

# Data Science for Agricultural Professionals

Marin L. Harbur

2022-08-19



# Table of Contents

<b>Preface</b>	<b>9</b>
Welcome . . . . .	9
R-language . . . . .	10
<b>1 Population Statistics</b>	<b>13</b>
1.1 Populations . . . . .	13
1.2 Case Study: Yield Map . . . . .	14
1.3 Distributions . . . . .	16
<b>2 Distributions and Probability</b>	<b>31</b>
2.1 Case Study . . . . .	31
2.2 The Normal Distribution Model . . . . .	33
2.3 The Z-Distribution . . . . .	38
<b>3 Sample Statistics</b>	<b>43</b>
3.1 Samples . . . . .	44
3.2 Case Study . . . . .	44
3.3 Distribution of Sample Means . . . . .	47
3.4 Central Limit Theorem . . . . .	50
3.5 Standard Error . . . . .	50
3.6 Degrees of Freedom . . . . .	51
3.7 The t-Distribution . . . . .	52
3.8 Confidence Interval . . . . .	59
3.9 Confidence Interval and Probability . . . . .	62

<b>4 Two-Treatment Comparisons</b>	<b>65</b>
4.1 Side-by-Side Trials . . . . .	65
4.2 Blocked Design . . . . .	67
4.3 Case Study . . . . .	69
4.4 Confidence Interval . . . . .	73
4.5 T-Test . . . . .	76
4.6 Conclusion . . . . .	77
<b>5 Understanding Statistical Tests</b>	<b>79</b>
5.1 Research Question . . . . .	79
5.2 The Model . . . . .	80
5.3 Hypotheses . . . . .	88
5.4 P-Value . . . . .	89
5.5 The P-Value and Errors . . . . .	91
5.6 One-Sided vs Two-Sided Hypotheses . . . . .	93
5.7 Exercise: Linear Additive Model . . . . .	96
5.8 Exercise: One-Sided Hypotheses . . . . .	102
5.9 Exercise: Type I and Type II Errors . . . . .	107
<b>6 Multiple Treatment Trials</b>	<b>109</b>
6.1 Case Study . . . . .	110
6.2 The Linear Additive Model . . . . .	112
6.3 Analysis of Variance . . . . .	117
6.4 The F statistic . . . . .	118
6.5 The ANOVA Table . . . . .	118
6.6 Visualizing How the Anova Table Relates to Variance . . . . .	123
6.7 Exercise: “Treatment Means” . . . . .	126
<b>7 Multiple Treatment Designs</b>	<b>135</b>
7.1 Randomized Complete Block Design . . . . .	136
7.2 Factorial Design . . . . .	142
7.3 Split-Plot Design . . . . .	161

<b>TABLE OF CONTENTS</b>	<b>5</b>
7.4 Linear Additive Model . . . . .	163
7.5 Conclusion . . . . .	166
7.6 Exercise: Randomized Complete Block Design . . . . .	167
7.7 Exercise: Factorial ANOVA . . . . .	176
7.8 Exercise: Split-Plot Design . . . . .	189
7.9 Exercise: Experimental Design . . . . .	198
<b>8 Means Separation and Data Presentation</b>	<b>207</b>
8.1 Case Study . . . . .	207
8.2 Least Significant Difference . . . . .	208
8.3 LSD Output in R . . . . .	210
8.4 Comparisonwise versus Experimentwise Error . . . . .	212
8.5 Tukey's Honest Significant Difference . . . . .	213
8.6 Linear Contrast . . . . .	216
8.7 Means Presentation . . . . .	224
8.8 Exercise: LSD and Tukey's HSD . . . . .	231
8.9 Exercise: Linear Contrasts . . . . .	238
8.10 Exercise: Means Tables . . . . .	250
<b>9 Messy and Missing Data</b>	<b>267</b>
9.1 Inspecting data for Normal Distributions . . . . .	268
9.2 Inspecting Data for Equal Variances . . . . .	275
9.3 Dealing with Messy Data . . . . .	282
9.4 Dealing with Missing Data . . . . .	287
9.5 Summary . . . . .	296
9.6 Exercise: Visual Data Inspection . . . . .	296
9.7 Exercise: Identifying Unequal Variances" . . . . .	302
9.8 Exercise: Transforming and Analyzing Data . . . . .	311
9.9 Exercise: Imputing Missing Data . . . . .	317

<b>10 Correlation and Simple Regression</b>	<b>323</b>
10.1 Correlation . . . . .	324
10.2 Correlation . . . . .	326
10.3 Regression . . . . .	336
10.4 Extrapolation . . . . .	351
10.5 Exercise: Scatterplots and Regression . . . . .	352
10.6 Exercise: Simple Linear Regression . . . . .	357
10.7 Exercise: Making Predictions from Regression Models. . . . .	363
<b>11 Nonlinear Relationships and Multiple Linear Regression</b>	<b>373</b>
11.1 Multiple Linear Regression . . . . .	374
11.2 Nonlinear Relationships . . . . .	388
11.3 Summary . . . . .	402
11.4 Exercise: Multiple Linear Regression . . . . .	402
11.5 Exercise: Avoiding Bloated Models . . . . .	406
11.6 Exercise: Tuning the Model . . . . .	410
11.7 Exercise: Nonlinear Regression . . . . .	419
<b>12 Spatial Statistics</b>	<b>433</b>
12.1 Projection (General) . . . . .	434
12.2 Shape Files . . . . .	443
12.3 Rasters . . . . .	453
12.4 Exercise: Shape Files . . . . .	463
12.5 Exercise: SSURGO . . . . .	471
12.6 Exercise: Rasters . . . . .	476
<b>13 Machine Learning</b>	<b>491</b>
13.1 Machine Learning . . . . .	491
13.2 Cluster Analyses . . . . .	492
13.3 k-Nearest-Neighbors . . . . .	507
13.4 Classification Trees . . . . .	528
13.5 Summary . . . . .	544

<b>14 Putting it all Together</b>	<b>547</b>
14.1 Scenario 1: Yield Map (Population Summary and Z-Distribution)	547
14.2 Scenario 2: Yield Estimate (Sampling t-Distribution) . . . . .	550
14.3 Scenario 3: Side-By-Side (t-Test) . . . . .	551
14.4 Scenario 4: Fungicide Trial (ANOVA CRD or RCBD) . . . . .	553
14.5 Scenario 5: Hybrid Response to Fungicide Trial (ANOVA Factorial or Split Plot) . . . . .	557
14.6 Scenario 6: Foliar Rate-Response Trial (Linear or Non-Linear Regression) . . . . .	560
14.7 Scenario 7: Application Map (Shapefiles and Rasters) . . . . .	566
14.8 Scenario 8: Yield Prediction (Multiple Linear Regression and other Predictive Models) . . . . .	570
14.9 Summary . . . . .	578



# Preface

## Welcome

Welcome to Data Science for Agricultural Professionals. I have written these course materials for a few reasons. First, it is my job. The more powerful motivation, however, was to write a guide that satisfied the following criteria:

- covers basic statistics used in reporting results from hybrid trials and other controlled experiments
- also addresses data science tools used to group environments and make predictions
- introduced students to R, an open-source statistical language that you can use after your studies at Iowa State University and can use without installing on your laptop, or using a VPN connection, which your work laptop may not allow.

I also wanted to develop a text that presented statistics around the situations in which you are most likely to encounter data:

- yield maps used by farmers and agronomists
- side-by-side or split-field trials used often at the retail level
- smaller-plot controlled experiments used in seed breeding and other product development
- rate recommendation trials for fertilizers and crop protection products
- fertilizer prediction maps
- decision support tools

I began my career as an a university researcher and professor, but in 2010 entered the private sector, working first in retail as a technical agronomist for a regional cooperative and then as a data scientist for a major distributor, developing product data and insights for a team of researchers and agronomists. In seeing how data were used at the retail and industry levels, I gained an appreciation for what areas of statistics were more often used than others.

What is important to the agricultural professional, in my experience, is data literacy and a basic ability to run analyses. It is easy, after years in the field, to lose skills gained as an undergraduate. My hope is that all of you upon completing this course will look at statistics you receive (from any source, but especially manufacturers) a little more critically. If you are involved in field research, I hope you will understand how to better layout and more creatively analyze field trials

I wanted to develop a reference that appreciated not all of us are mathematical

progedies, nor do we have flawless memories when it comes to formula. At the risk of redudancy, I will re-explain formulas and concepts – or at least provide links – so you aren't forever flipping back and forth trying to remember sums of squares, standard errors, etc. I am committed to making this as painless as possible.

## R-language

In this course, we will use the R language to summarise data, calculate statistics, and view trends and relationships in data. R is open-source (free) and incredibly versatile. I have used multiple languages, including SAS and SPSS, in my career; in my opinion, R is not only the cheapest, but the best, and I now use it exclusively and almost daily in my work.

R also has personal connections to Iowa State: Hadley Wickam, Carson Seivert, two authors of the R language, are Iowa State graduates.

We will use an application called R-Studio to work with R language. R Studio allows you to write and save scripts (code) to generate analyses. It also allows you to intersperse Markdown (html code) in between your statistical output so that you can explain and discuss your results. The beauty of Markdown is your report and your statistics are together in a single file; no cutting and pasting is needed between documents.

R is all about having options, and so with R-Studio you have the option of installing it on your laptop or, in case you are using a work laptop without administrative privileges, you can also use a cloud site, R-Studio Cloud, to work with R and save your work.

I know that for most of you R (and coding itself) will be a new experience. I am sure the idea of coding will intimidate many of you. To head off your anxiety as much as possible, I offer this: I understand that coding is challenging. I spend my days making mistakes, searching for bugs, and looking up how to do something for the umpteenth time. If something confuses you, that is normal.

But I can also assure you that you will likely have an easy time remembering what functions to use. In other words, which code to use will not be the problem. Most of your bugs will be due to misspellings, forgetting to close a parentheses, or referring to a dataset by the wrong name, e.g. “soy\_data” instead of “soybean\_data”. And you will get better at avoiding these mistakes over time – there is not more to learn, just practice. Our exercises in R will be designed to give you that practice, and introduce new functions as slowly as possible.

At the end of this course, you will have not only your completed work, but the course materials themselves as a resource from which you can borrow code for future projects. There is no shame in copying lines of codes (or whole chunks

of code) into your own original analyses. All of us data scientists do that, and it is one of the best ways to continue learning.

R is supported by many great books which may be accessed for free online, or purchased online for very reasonable prices. These may be used as references during the course, but also to continue learning for years to come. These include many references from bookdown.org:

- Introduction to Data Science: although there are many “Introduction to R” books, this one closely matches how we approach it in Agronomy 513. (<https://rafalab.github.io/dsbook/>)
- R Graphics Cookbook: a comprehensive explanation of how to create just about any plot you could ever imaging in R, mainly using the ggplot package. (<https://r-graphics.org/>)
- Geocomputation with R: The best book I have found for learning how to work with spatial data. (<https://geocompr.robinlovelace.net/>)
- Hands-On Machine Learning with R: I have not read this book, but it appears to be a robust and beautifully-illustrated guid to machine learning concepts in R. (<https://bradleyboehmke.github.io/HOML/>)



# Chapter 1

## Population Statistics

### 1.1 Populations

Almost every statistics text begins with the concept of a **population**. A population is the complete set of individuals to which you want to predict values. Let's dwell on this concept, as it is something that did not hit home for me right away in my career. Again, the population is all of the individuals for which you are interested in making a prediction. What do we mean by individuals? Not just people – individuals can plants, insects, disease, livestock or, indeed, farmers.

Just as important as what individuals are in a population is its extent. What do you want the individuals to represent? If you are a farmer, do you want to apply the data from these individuals directly to themselves, or will you use them to make management decisions for the entire field, or all the fields in your farm? Will you use the results this season or to make predictions for future seasons? If you are an animal nutritionist, will you use rations data to support dairy Herefords, or beef Angus?

If you are a sales agronomist, will you use the data to support sales on one farm, a group of farms in one area, or across your entire sales territory? If you are in Extension, will the individuals you measure be applicable to your entire county, group of counties, or state? If you are in industry like me, will your results be applicable to several states?

This is a very, very critical question, as you design experiments – or as you read research conducted by others. To what do or will the results apply? Obviously, an Iowa farmer should not follow the optimum planting date determined for a grower in North Carolina, nor should an Ohio farmer assume our pale clays will be as forgiving as the dark, mellow loam of southern Minnesota.

Drilling down, you might further consider whether research was conducted in

areas that have similar temperature or rainfall, how different the soil texture might be to the areas to which you want to apply results. At the farm level, you might ask how similar the crop rotation, tillage, or planting methods were to that operation. At the field level, you might wonder about planting date or the hybrid that was sown.

When you use the results from set of individuals to make predictions about other individuals, you are making inferences – you are using those data to make predictions, whether it be for that same field next month or next year, or for other locations (areas in a field, fields in a farm, counties, or states). When we speak of an inference space, then, that is the total group of individuals to which you will apply your results. Or, in another word, your population.

In summary, one of the most critical skills you can apply with data science has no formula and, indeed, little to do with math (at least in our application). It is to ask yourself, will my experiment represent the entire population in which I am interested? Indeed, one field trial likely will not address the entire population in which you are interested – it is up to you to determine the population to which you are comfortable applying those results.

In fact, statistics or data science done without this “domain” knowledge whether a dataset is useful or experimental results are reasonable can be disastrous. Just because a model fits one dataset very well or treatments are significantly different does not mean they should be used to make decisions. Your competency as an agronomic data scientist depends on everything you learn in your program of study. Soil science, crop physiology, and integrated pest management, to name just a few subjects, are as much a prerequisite as any math course you have taken.

In some cases all of the individuals in a population can be measured – in such case, we will use the basic statistics described in this unit. The yield map we will analyze in this unit is a loose example of a case where we can measure

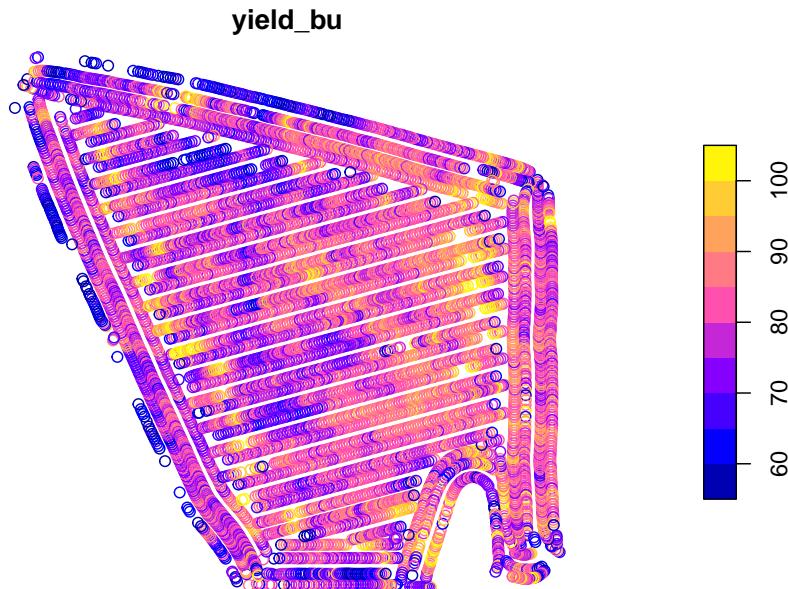
In most cases, however, it is not physically or financially feasible to measure all individuals in a population. In that case, subsets of the population, called samples, are used to estimate the range of individuals in a population.

## 1.2 Case Study: Yield Map

For our first case study, we will use a situation where every individual in our population can be measured: a single soybean field in central Iowa. In this case, yield data were gathered using a combine monitor. In case you don’t live and breathe field crops like me, combines (the machines that harvest grain) are usually equipped with a scale that repeatedly weighs grain as the combine moves across the field. The moisture of the grain is also recorded. These data are combined with measures of the combine speed and knowledge of the number

of rows harvested at once to calculate the yield per area of grain, adjusted to the market standard for grain moisture.

Almost all of you have seen a yield map somewhat like the one below. In this map, blue circles represent lower yields, while yellow and orange circles represent higher yields.



We will learn in the Exercises portion of this lesson how to create a map like this using just a few lines of code.

Each dataset has a structure – the way data are organized in columns and rows. To get a sense of the structure of our soybean dataset, we can examine the first 6 rows of the dataset using R.

```
##  
## Attaching package: 'kableExtra'  
  
## The following object is masked from 'package:dplyr':  
##  
##     group_rows
```

DISTANCE	SWATHWIDTH	VRYIELDVOL	Crop	WetMass	Moisture	Time
0.9202733	5	57.38461	174	3443.652	0.00	9/19/2016 4:45:46
2.6919269	5	55.88097	174	3353.411	0.00	9/19/2016 4:45:48
2.6263101	5	80.83788	174	4851.075	0.00	9/19/2016 4:45:49
2.7575437	5	71.76773	174	4306.777	6.22	9/19/2016 4:45:51
2.3966513	5	91.03274	174	5462.851	12.22	9/19/2016 4:45:54
3.1840529	5	65.59037	174	3951.056	13.33	9/19/2016 4:45:55

Each row in the above dataset is a **case**, sometimes called an **instance**. It is an single observation taken within a soybean field. That case may contain one or more **variables**: specific measurements or observations recorded for each case. In the dataset above, variables include *DISTANCE*, *SWATHWIDTH*, *VRYIELDVOL*, *Crop*, *WetMass*, and many others.

The two most important to us in this lesson are *yield\_bu* and *geometry*. That this dataset has a column named *geometry* indicates it is a special kind of dataset called a **shape file** – a dataset in which the measures are geo-referenced. That is, we know where on Earth these measurements were taken. The *geometry* column in this case identifies a point with each observation.

## 1.3 Distributions

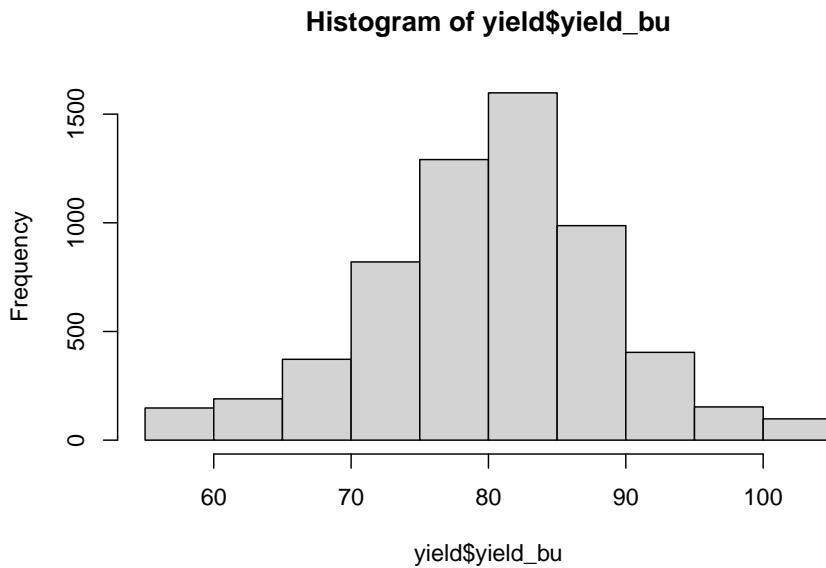
At this point, we have two options any time we want to know about soybean yield in this field. We can pull out this map or the complete dataset (which has over 6,500 observations) and look at try to intuitively understand the data. Or we can use statistics which, in a sense, provide us a formula for approximating the values in our dataset with just a few numbers.

A **distribution** describes the range of values that occur within a given variable. What is the range of values in our measured values? In this example, what are the highest and lowest yields we observed? What ranges of values occur more frequently? Many times, we want to see whether the distribution of one

### 1.3.1 Histograms

Before we get into the math required to generate these statistics, however, we should look at the shape of our data. What is the range of values in our measured values? In this example, what are the highest and lowest yields we observed? What ranges of values occur more frequently? Do the observed values make general sense?

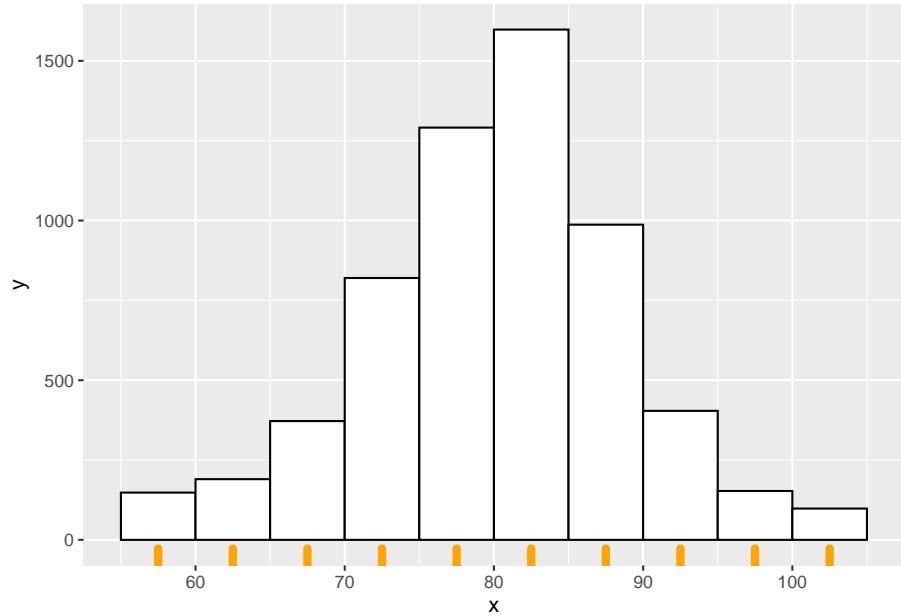
One of the easiest and most informative things for us to do is to create a particular bar chart known as a **histogram**.



In the histogram above, each bar represents range of values. This range is often referred to as a *bin*. The lowest bin includes values from 50 to 59.0000. The next bin includes values from 60 to 69.9999. And so on. The height of each bar represents the **frequency** within each range: the number of individuals in that population that have values within that range.

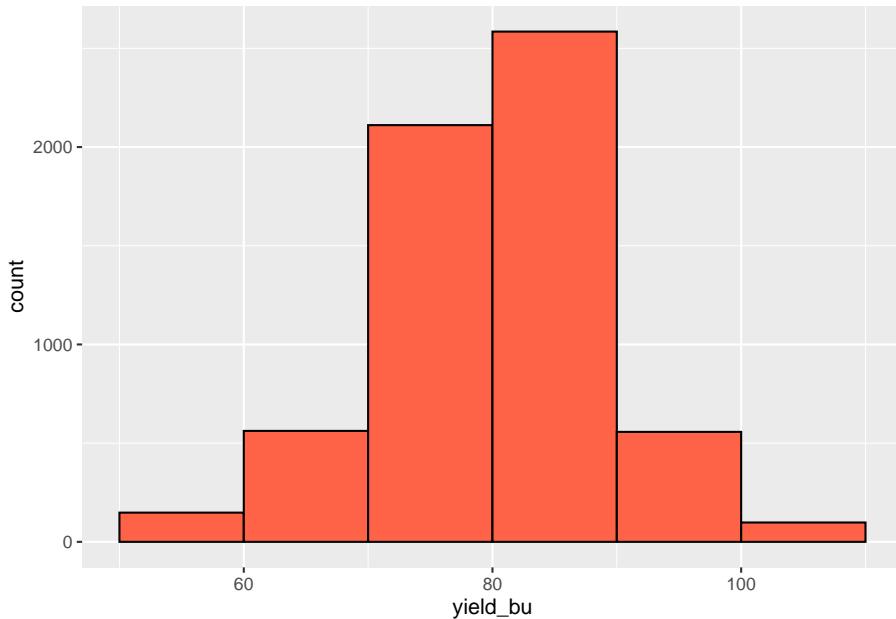
Each bin can also be defined by its **midpoint**. The midpoint is the middle value in each range. For the bin that includes values from 50 to 50.9999, the midpoint is 55. For the bin that includes values from 60 to 60.9999, the midpoint is 65.

In the plot above, the midpoint for each bar is indicated by the orange bar beneath it.



There are many ways in which we can draw a histogram – and other visualizations – in R. We will learn more in this course about an R package called `ggplot2`, which can create just about any plot you might imagine. Here is a simple taste:

```
ggplot(data=yield, aes(x=yield_bu)) +  
  geom_histogram(breaks=seq(50, 110, 10), fill="tomato", color="black")
```



Varying the *bin width* provides us with different perspectives on our distribution. Wide bins, which each include a greater range of values, will provide more gross representations of the data, while narrower bins will provide greater detail. When bins are too narrow, however, there may be gaps in our histogram where no values occur within particular bins.

Throughout this course, I have created interactive exercises to help you better visualize statistical concepts. Often, they will allow you to observe how changing the variables or the number of observations can affect a statistical test or visualization.

These exercises are located outside of this textbook. To access them, please follow links like that below. The exercises may take several seconds to launch and run in your browser. I apologize for their slowness – this is the best platform I have found to date.

Please click on the link below to open an application where you can vary the bin width and see how it changes your perspective:

<https://marin-harbur.shinyapps.io/01-app-histogram/>

### 1.3.2 Percentiles

We can also use **percentiles** to describe the values of a variable more numerically. Percentiles describe how values the proportional spread of values, from lowest to highest, within a distribution. To identify percentiles, the data are

numerically ordered (ranked) from lowest to highest. Each percentile is associated with a number; the percentile is the percentage of all data equal to or less than that number. We can quickly generate the 0th, 25th, 50th, and 75th, and 100th percentile in R:

```
summary(yield$yield_bu)
```

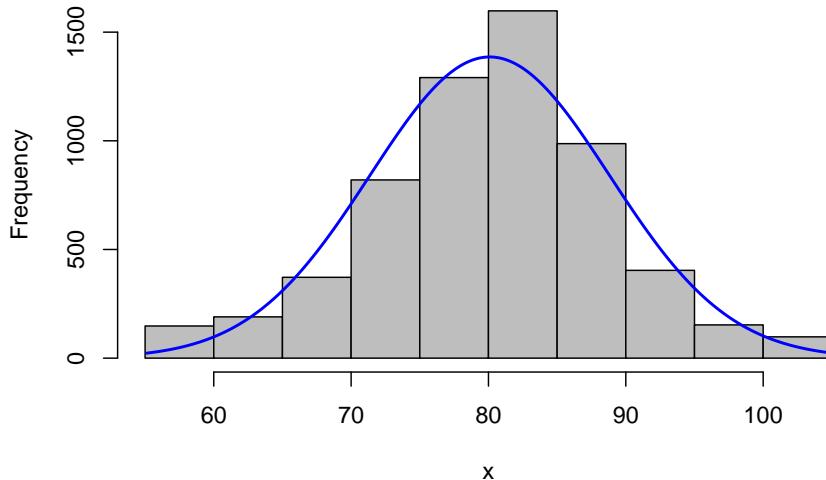
```
##      Min. 1st Qu. Median     Mean 3rd Qu.    Max.
##    55.12    74.96   80.62   80.09   85.44 104.95
```

This returns six numbers. The 0th percentile (alternatively referred to as the minimum) is 55.12 – this is the lowest yield measured in the field. The 25th percentile (also called the 1st Quartile) is 74.96. This means that 25% of all observations were equal to 74.96 bushels or less. The 50th percentile (also known as the median) was 80.62, meaning half of all observations were equal to 80.62 bushels or less. 75% of observations were less than 85.44, the 75th percentile (or 3rd quartile). Finally, the 100th percentile (or maximum yield) recorded for this field was 104.95.

We are now gaining a better sense of the range of observations that were most common. But we can describe this distribution with even fewer numbers.

### 1.3.3 Normal Distribution Model

Let's overlay a curve, representing the **normal distribution**, on our histogram. You have probably seen or heard of this curve before. Often it is called a *bell curve*; in school, it is the *Curve* that many students count on to bring up their grades. We will learn more about this distribution in *Lesson 2*.



In a perfect scenario, our curve would pass through the midpoint of each bar. This rarely happens with real-world data, and especially in agriculture. The data may be slightly **skewed**, meaning there are more individuals that measure above the mean than below, or vice versa.

In this example, our data do not appear skewed. Our curve seems a little too short and wide to exactly fit the data. This is a condition called **kurtosis**. No, kurtosis doesn't mean that our data stink; they are just more spread out or compressed than in a "perfect" situation.

No problem. We can – and should – conclude it is appropriate to fit these data with a normal distribution. If we had even more data, the curve would likely fit them even better.

Many populations can be handily summarized with the normal distribution curve, but we need to know a couple of statistics about the data. First, we need to know where the center of the curve should be. Second, we need to know the width or dispersion of the curve.

### 1.3.4 Measures of Center

To mathematically describe our distribution, we first need a **measure of center**. The two most common measures of center are the arithmetic mean and median. The **mean** is the sum of all observations divided by the number of observations.

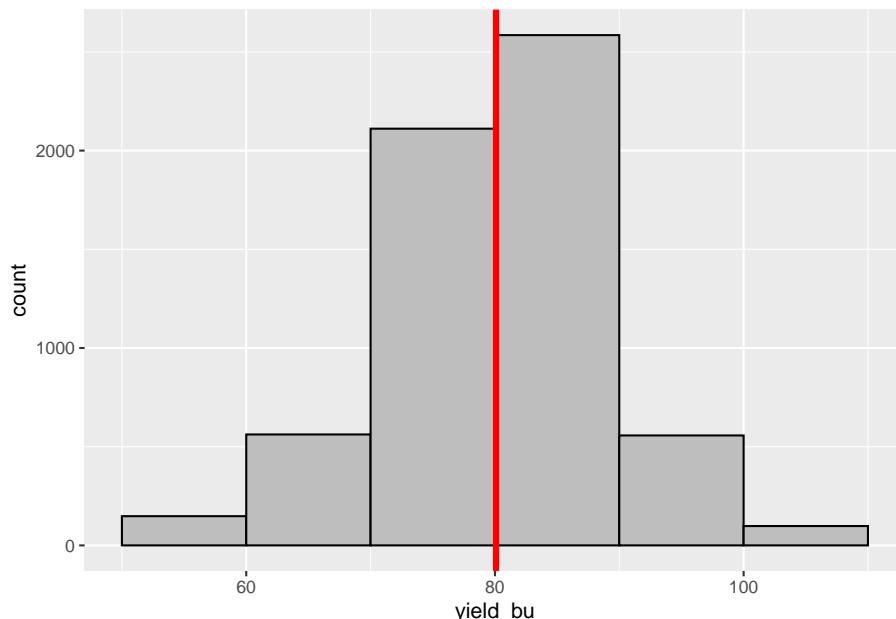
$$\mu = \frac{\sum x_i}{n}$$

The  $\mu$  symbol (a u with a tail) signifies the true mean of a population. The  $\sum$  symbol (the character next to  $x_i$  which looks like the angry insect alien from *A Quiet Place*) means “sum”. Thus, anytime you see the  $\sum$  symbol, we are summing the variable(s) to its right.  $x_i$  is the value  $x$  of the  $i$ th individual in the population. Finally,  $n$  is the number of individuals in the population.

For example, if we have a set of numbers from 1:5, their mean can be calculated as:

$$\frac{1 + 2 + 3 + 4 + 5}{5} = 3$$

The mean yield for our field is about 80.09 bushels per acre. This is represented by the red line in the histogram below.



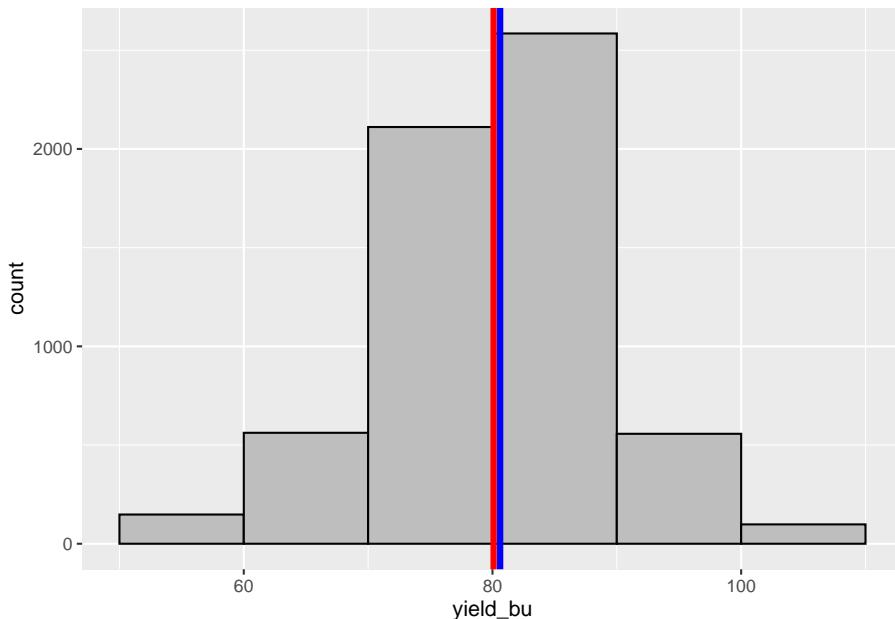
Earlier, you were introduced to the median. As discussed, the **median** is a number such that half of the individuals in the population are greater and half are less. If we have an odd number of individuals, the median is the “middle” number as the individuals are ranked from greatest to least.

$$\{1, 2, 3, 4, 5\} \text{median} = 3$$

If we have an even number of measures, the median is the average of the middle two individuals in the ranking:

$$\{1, 2, 3, 4, 5, 6\} \text{median} = 3.5$$

Let's add a blue line to our histogram to represent the median.



As you can see, they are practically identical. When the mean and median are similar, the number of individuals measuring greater and less than the mean are roughly equivalent. In this case, our data can be represented using the normal distribution.

We also need a statistic that tells us how wide to draw the curve. That statistic is called a measure of dispersion, and we will learn about it next.

### 1.3.5 Measures of Dispersion

To describe the spread of a population, we use one of three related **measures of dispersion**: sum of squares, variance, and standard deviation. Although there is a little math involved in these three statistics, please make yourself comfortable with their concepts because they are *very* important in this course. Almost every statistical test we will learn during this course is rooted in these measures of population width.

### 1.3.5.1 Sum of Squares

The first measure of population width is the **sum of squares**. This is the sum of the squared differences between each observation and the mean. The sum of squares of a measurement  $x$  is:

$$S_{xx} = (x_i - \mu)^2$$

Where again  $x_i$  is the value  $x$  of the  $i$ th individual in the population and  $\mu$  is the true mean of a population.

Why do we square the differences between the observations and means? Simply, if we were to add the unsquared differences they would add to exactly zero. Let's prove this to ourselves. Let's again use the set of numbers (1, 2, 3, 4, 5). We can measure the distance of each individual from the mean by subtracting the mean from it. This difference is called the residual.

```
sample_data = data.frame(individuals = c(1,2,3,4,5))

library(janitor)

## 
## Attaching package: 'janitor'

## The following objects are masked from 'package:stats':
## 
##     chisq.test, fisher.test

first_resid_table = sample_data %>%
  mutate(mean = 3) %>%
  mutate(residual = individuals - mean) %>%
  mutate(mean = as.character(mean))

first_resid_totals = data.frame(individuals = "Total",
                                 mean = "",
                                 residual = sum(first_resid_table$residual))

first_resid_table %>%
  rbind(first_resid_totals) %>%
  kbl()
```

individuals	mean	residual
1	3	-2
2	3	-1
3	3	0
4	3	1
5	3	2
Total		0

The first column of the above dataset contains the individual observations. The second column contains the population mean, repeated for each observation. The third column is the residuals, which are calculated by subtracting each observed value from the population mean.

And if we sum these residuals we get zero.

$$(-2) + (-1) + (0) + (+1) + (+2) = 0$$

Let's now do this with our field data. The number of residuals (almost 6800) is too many to visualize at once, so we will pick 20 at random.

```
set.seed(080921)
yield_sample = data.frame(yield = sample(yield$yield_bu, 10))

second_resid_table = yield_sample %>%
  mutate(yield = round(yield,2),
        mean = round(mean(yield),2),
        residual = yield-mean)

second_resid_totals = data.frame(yield = "Total",
                                  mean = "",
                                  residual = sum(second_resid_table$residual))

second_resid_table %>%
  rbind(second_resid_totals) %>%
  kbl()
```

yield	mean	residual
83.61	76.52	7.09
86.82	76.52	10.30
68.39	76.52	-8.13
81.91	76.52	5.39
80.75	76.52	4.23
57.06	76.52	-19.46
62.58	76.52	-13.94
86.6	76.52	10.08
80.05	76.52	3.53
77.42	76.52	0.90
Total		-0.01

If we sum up all the yield residuals, we get -0.04. Not exactly zero, but close. The difference from zero is the result of rounding errors during the calculation.

The sum of squares is calculated by squaring each residual and then summing the residuals. For our example using the set (1, 2, 3, 4, 5):

```
first_squares_table = first_resid_table %>%
  mutate(square = residual^2)

first_squares_totals = data.frame(individuals = "Total",
                                   mean = "",
                                   residual = "",
                                   square = sum(first_squares_table$square))

first_squares_table %>%
  rbind(first_squares_totals) %>%
  kbl()
```

individuals	mean	residual	square
1	3	-2	4
2	3	-1	1
3	3	0	0
4	3	1	1
5	3	2	4
Total			10

And for our yield data:

```
second_squares_table = second_resid_table %>%
  mutate(square = round(residual^2, 2)) %>%
  mutate(residual = round(residual, 2))

second_squares_totals = data.frame(yield = "Total",
```

```

mean = "",
residual = "",
square = sum(second_squares_table$square))
second_squares_table %>%
  rbind(second_squares_totals) %>%
  kbl()

```

yield	mean	residual	square
83.61	76.52	7.09	50.27
86.82	76.52	10.3	106.09
68.39	76.52	-8.13	66.10
81.91	76.52	5.39	29.05
80.75	76.52	4.23	17.89
57.06	76.52	-19.46	378.69
62.58	76.52	-13.94	194.32
86.6	76.52	10.08	101.61
80.05	76.52	3.53	12.46
77.42	76.52	0.9	0.81
Total			957.29

### 1.3.5.2 Variance

The sum of squares helps quantify spread: the larger the sum of squares, the greater the spread of observations around the population mean. There is one issue with the sum of squares, though: since the sum of square is derived from the differences between each observation and the mean, it is also related to the number of individuals overall in our population. In our example above, the sum of squares was 10.

Now, let's generate a dataset with two 1s, two 2s, two 3s, two 4s, and two 5s:

```

first_squares_table = first_resid_table %>%
  mutate(square = residual^2)

double_squares_table = first_squares_table %>%
  rbind(first_squares_table)

double_squares_totals = data.frame(individuals = "Total",
                                    mean = "",
                                    residual = "",
                                    square = sum(double_squares_table$square))

double_squares_table %>%

```

```
rbind(double_squares_totals) %>%
  kbl()
```

individuals	mean	residual	square
1	3	-2	4
2	3	-1	1
3	3	0	0
4	3	1	1
5	3	2	4
1	3	-2	4
2	3	-1	1
3	3	0	0
4	3	1	1
5	3	2	4
Total			20

You will notice the sum of squares increases to 20. The spread of the data did not change: we recorded the same five values. The only difference is that we observed each value twice.

The moral of this story is this: given any distribution, the sum of squares will always increase with the number of observations. Thus, if we want to compare the spread of two different populations with different numbers of individuals, we need to adjust our interpretation to allow for the number of observations.

We do this by dividing the sum of squares,  $S_{xx}$  by the number of observations,  $n$ . In essence, we are calculating an “average” of the sum of squares. This value is the variance,  $\sigma^2$ .

$$\sigma^2 = \frac{S_{xx}}{n}$$

We can calculate the variance as follows.

In our first example, the set  $\{1,2,3,4,5\}$  had a sum of squares of 10. in that case, the variance would be:

$$\frac{10}{5} = 2$$

In our second example, the set  $\{1,2,3,4,5,1,2,3,4,5\}$  had a sum of squares of 20. In that example, the variance would be

$$\frac{20}{10} = 2$$

As you can see, the variance is not affected by the size of the dataset, only by the distribution of its values.

Later on in this course, we will calculate the variance a little differently:

$$\sigma^2 = \frac{S_{xx}}{n-1}$$

$n-1$  is referred to as the **degrees of freedom**. We use degrees of freedom when we work with samples (subsets) of a population. In this unit, however, we are working with populations, so we will not worry about those.

### 1.3.5.3 Standard Deviation

Our challenge in using the variance to describe population spread is it's units are not intuitive. When we square the measure we also square the units of measure. For example the variance of our yield is measured in units of bushels<sup>2</sup>. Wrap your head around that one. Our solution is to report our final estimate of population spread in the original units of measure. To do this, we calculate the square root of the variance. This statistic is called the standard deviation,  $\sigma$ .

$$\sigma = \sqrt{(\sigma^2)}$$

For the dataset {1,2,3,4,5}, the sum of squares is 10, the variance 2, and the standard deviation is:

$$\sigma = \sqrt{2} = 1.4$$

For our yield dataset, the sum of squares is 957.29, and based on 10 observations. Our variance is therefore:

$$\sigma^2 = \frac{957.29}{10} = 9.57 \text{ bushels}^2 / \text{acre}^2$$

Our sum of squares is :

$$\sqrt{9.57 \text{ bushels}^2 / \text{acre}^2} = 3.09 \text{ bushels} / \text{acre}$$

That is enough theory for this first week of class. The remainder of this lesson will focus on introducing you to **RStudioCloud**.



## Chapter 2

# Distributions and Probability

In this unit we will continue with the **normal distribution model** introduced in the previous unit. As you will recall, the normal distribution is a symmetrical curve that represents the frequency with which individuals with different particular measured values occur in a population.

The peak of the curve is located at the population mean,  $\mu$ . The width of the curve reflects how spread out other individuals are from the mean. We learned three ways to measure this spread: the sum of squares, the variance, and the standard deviation. These statistics can roughly be thought of as representing the sums of the squared differences, the average of the squared distances, and the distances presented in the original units of measure. These four statistics – mean, sum of squares, variance, and standard deviation – are among the most important you will learn in this course.

### 2.1 Case Study

This week, we will continue to work with the Iowa soybean yield dataset introduced to us in Unit 1.

Here again is the structure of this dataset:

```
head(yield)
```

```
## Simple feature collection with 6 features and 12 fields
## Geometry type: POINT
## Dimension: XY
```

```

## Bounding box: xmin: -93.15033 ymin: 41.66641 xmax: -93.15026 ymax: 41.66644
## Geodetic CRS: WGS 84
##   DISTANCE SWATHWIDTH VRYIELDVOL Crop  WetMass Moisture           Time
## 1 0.9202733          5 57.38461 174 3443.652    0.00 9/19/2016 4:45:46 PM
## 2 2.6919269          5 55.88097 174 3353.411    0.00 9/19/2016 4:45:48 PM
## 3 2.6263101          5 80.83788 174 4851.075    0.00 9/19/2016 4:45:49 PM
## 4 2.7575437          5 71.76773 174 4306.777    6.22 9/19/2016 4:45:51 PM
## 5 2.3966513          5 91.03274 174 5462.851   12.22 9/19/2016 4:45:54 PM
## 6 3.1840529          5 65.59037 174 3951.056   13.33 9/19/2016 4:45:55 PM
##   Heading VARIETY Elevation           IsoTime yield_bu
## 1 300.1584 23A42 786.8470 2016-09-19T16:45:46.001Z 65.97034
## 2 303.6084 23A42 786.6140 2016-09-19T16:45:48.004Z 64.24158
## 3 304.3084 23A42 786.1416 2016-09-19T16:45:49.007Z 92.93246
## 4 306.2084 23A42 785.7381 2016-09-19T16:45:51.002Z 77.37348
## 5 309.2284 23A42 785.5937 2016-09-19T16:45:54.002Z 91.86380
## 6 309.7584 23A42 785.7512 2016-09-19T16:45:55.005Z 65.60115
##   geometry
## 1 POINT (-93.15026 41.66641)
## 2 POINT (-93.15028 41.66641)
## 3 POINT (-93.15028 41.66642)
## 4 POINT (-93.1503 41.66642)
## 5 POINT (-93.15032 41.66644)
## 6 POINT (-93.15033 41.66644)

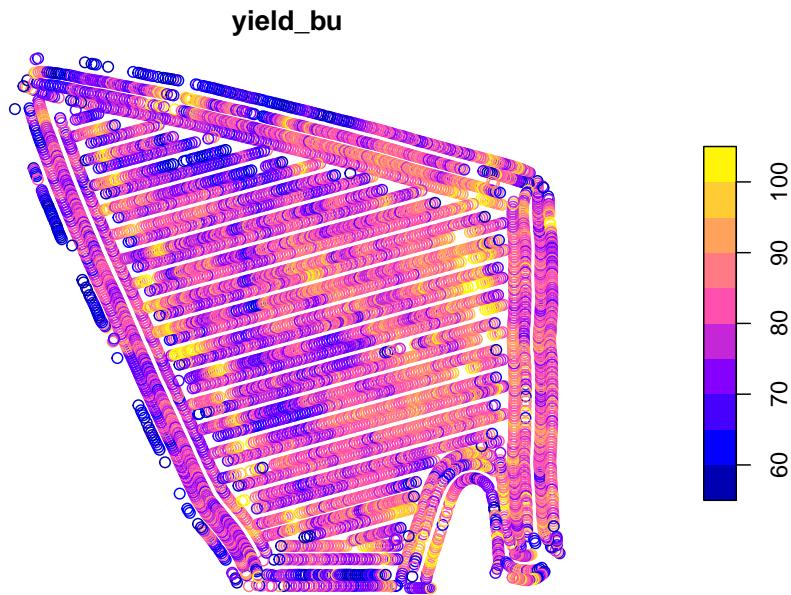
```

And here is the map of the field:

```

library(sf)
plot(yield["yield_bu"])

```



## 2.2 The Normal Distribution Model

Mean, sum of squares, variance, and standard deviation are so important because they allow us to reconstruct the normal distribution model. Before we go further, what is a **model**? Is it frightening we are using that term already in the *second chapter of this text???*

Models can be very complex, but in there essence they all have the following in common: they are simplified representations of reality. No, that doesn't mean that models are "fake" (SMH). It means that they summarize aspects of data, both measured and predicted. The normal distribution model describes the relationship between the values of individuals and how frequently they appear in the population. The model is useful because we can use it to approximately reconstruct the dataset at any time by knowing just two things about the original dataset – its mean and its standard deviation.

### 2.2.1 The Bell Curve

The normal distribution curve is often referred to as the *bell curve*, since it is taller in the middle and flared on either side. This shape reflects the tendency of measures within many populations to occur more frequently near the population mean than far from it. Why does this occur and how do we know this?

As agronomists, we can reflect on what it takes to produce a very good – or very bad crop. For a very good crop, many factors need to coincide: temperature, precipitation, soil texture, rate of nitrogen mineralization, proper seed singulation (spacing during planting), pest control, and hybrid or variety selection, to name just a few. In a typical season or within a field, we might optimize a few of these factors, but the possibility of optimizing every one is exceedingly rare. Thus, if we are measuring yield, measures near the mean yield will occur more frequently. Extremely high yields will occur less frequently.

Conversely, very low yields require we manage a crop very badly or that catastrophic weather conditions occur: a hailstorm, flood, or tornado. A frost at exactly the wrong time during seed germination or, in corn, excessive heat or a drought during pollination or grain fill. A planter box running out of seed or a fertilizer nozzle jamming. These things do occur, but less frequently.

The distribution of individuals around the mean is also a the result of measurement inaccuracies. Carl Friedrich Gauss, who introduced the normal distribution model, showed that it explained the variation among his repeated measurements of the position of stars in the sky. All measurements of continuous data (those that can be measured with a ruler, a scale, a graduated cylinder, or machine) have variation – we use the term **accuracy** to explain their variation around the population mean.

### 2.2.2 Distribution and Probability

Some areas of mathematics like geometry and algebra identify theorems: consistent, proven relationships between variables. In statistics, however, we typically solve for the **probability** that one or more variables are have a given value or range of values. To be more specific, most of the statistical tests we will learn can be reduced to the probability that a particular value is observed in a population. These probabilities include:

- that a given value or mean is observed for a population; we calculate this using the *normal distribution*.
- that the difference between two treatments is not zero; we calculate this using a *t-Test* or *Least Significant Difference* test.
- that a value will be observed in one variable, given a specific value of another variable; we calculate this using *Linear Regression*.
- that the spread of individual measures in a population is better predicted by treatment differences than random variation; we calculate this using an *F-test* and *Analysis of Variance*)

Each of these probabilities is calculated from a distribution – the frequency with which individuals appear in a population. Another way of stating this is that

probability is the proportion of individuals in a population that are expected to have values within a given range. Examples could include:

- the proportion of individual soybean yield measurements, within one field, that are less than 65 bushels per acre
- the proportion of individual corn fields that have an average less than 160 bushels per acre
- the proportion of trials in which the difference between two treatments was greater than zero
- the proportion of observations in which the actual crop yield is greater than that predicted from a regression model

### 2.2.3 Probability and the Normal Distribution Curve

Probability can be calculated as the proportion of the area underneath the normal distribution that corresponds to a particular range of values. We can visualize this, but first we need to construct the normal distribution curve for our soybean field.

We need just two statistics to construct our distribution curve: the mean and standard deviation of our yield. In the last lesson, we learned how easily these both can be calculated in R:

```
library(muStat)
yield_mean = mean(yield$yield_bu)
yield_sd = stdev(yield$yield_bu, unbiased = FALSE)

# to see the value of yield_mean and yield_sd, we just run their names in our code
yield_mean

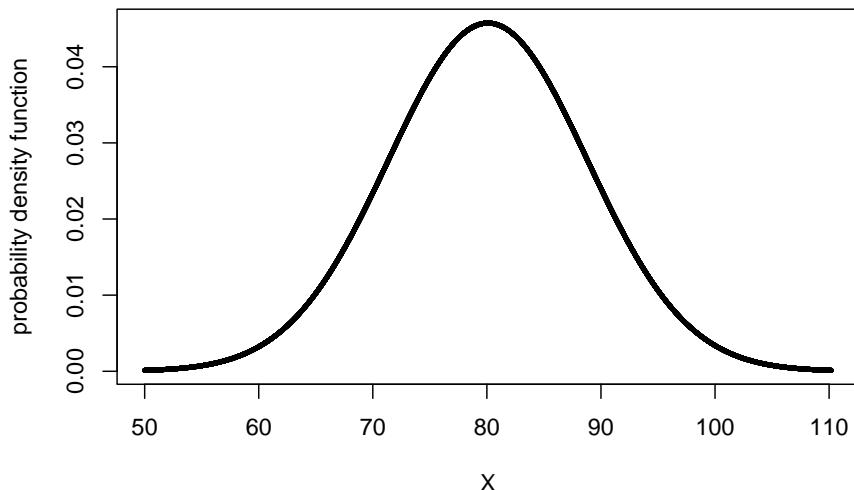
## [1] 80.09084

yield_sd
```

```
## [1] 8.72252
```

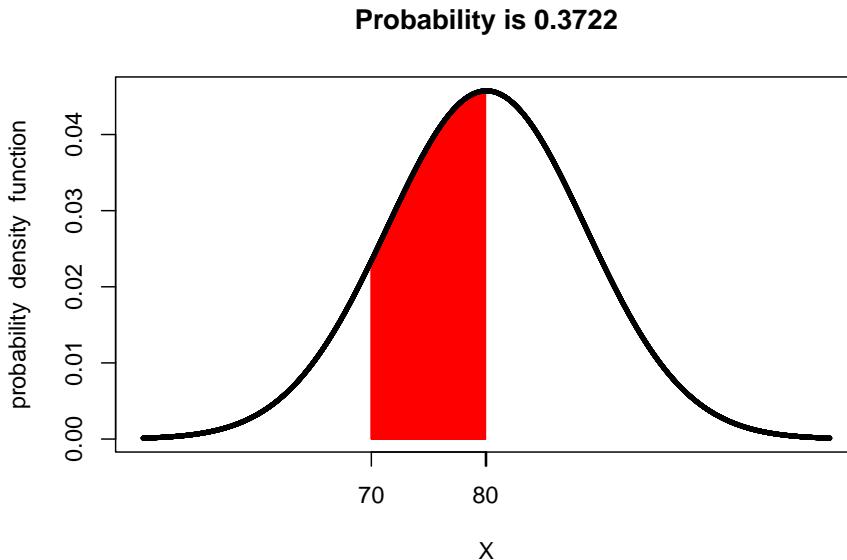
In this lesson's exercise section, we will learn to use R to construct the distribution curve for any population, given the population mean and population standard deviation.

```
library(fastGraph)
plotDist("dnorm", yield_mean, yield_sd)
```



Let's now shade the area underneath the normal curve corresponding to X values from 70 - 80. This area will represent the proportion of the population where individuals were measured to have values between 70 and 80 bushels. We will use a function in R called `shadeDist` to do this. You will learn more about this function this lesson in the exercise section.

```
shadeDist(xshade=c(70,80), ddist = "dnorm", yield_mean, yield_sd, lower.tail = FALSE)
```



Pretty cool, huh? The red area is the proportion of the soybean yield population that was between 70 and 80 bushels/acre. At the top of the output, `shadeDist` has also reported the proportion of the curve represented by that area, which it has labelled *Probability*. The probability in this case is 0.3722. What does that number mean?

The total area of the curve is 100%, or 1.0000. The proportion of the area under the curve that corresponds with yields from 70 to 80 bushels, then, is 37.22 percent of the area. This means that 37.22 percent of the individuals in our yield population had values from 70 and 80 bushels

But wait a second – why is R using the term *Probability*? Think of it this way. Imagine you sampled 1000 individuals from our population. If 37.22 percent of our individuals have values from 70 to 80 bushels, then about 37% of the individuals in your sample should have values from 70 to 80 bushels. In other words, there is a 37% probability that any individual you select, at random, from the population will have a value from 70 to 80 bushels.

Let's test this. Let's randomly sample 1000 individuals from our population. Then lets count the number of individuals that have yields between 70 and 80 bushels. For the curious, this is how we do this in R. We will run three lines of R code to do this:

- First, use the `set.seed()` function to specify a certain point in the population where R will begin sampling. R uses an algorithm to choose samples. This is not quite the same thing as random sampling – by setting a seed, we ensure that we can generate the same “random” set in the future should

we need to. Our seed number can be any numeric value; in this case, the date this section was revised.

- Second, use the `sample` function to randomly sample our population and return a vector of those numbers.
- Third, use the `subset` function to subset our data into those that meet logical conditions and return a dataframe or vector.
- Fourth, use the `length` function to count the number of observations.

For everyone else, just understand this is how we came up with the sample of 1000 individuals.

```
# 1) set seed
set.seed(081521)

# 2) take sample
yield_sample = sample(yield$yield_bu, 1000)

# "yield_sample >=70 & yield_sample <=80 tells it to only include measures from 70 to 80

# 3) Subset data into values between 70 and 80
yield_subset = subset(yield_sample, yield_sample >=70 & yield_sample <=80)

# 4) Count number of samples in subset
length(yield_subset)

## [1] 353
```

Our sample had 1000 individuals. 353, or 35.3%, were had yields between 70 and 80 bushels/acre.

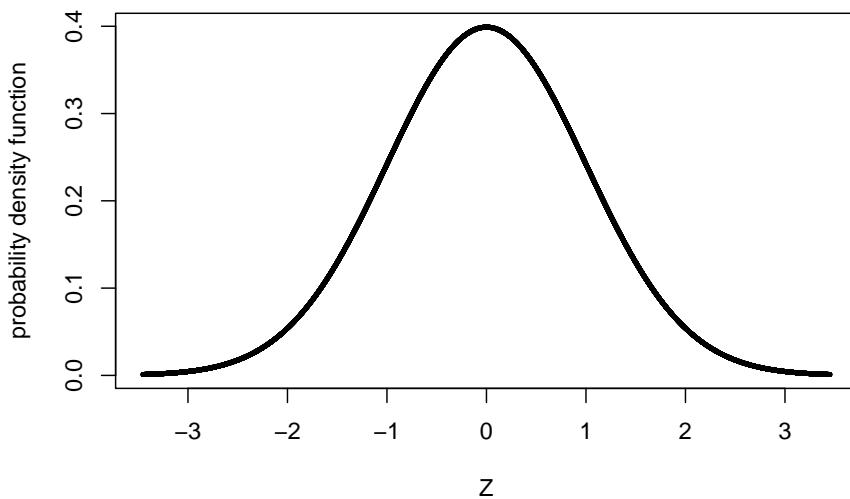
Is the proportion predicted by the normal distribution curve exactly that of the actual population? No. The normal distribution curve is, after all, a model – it is an approximation of the actual population. In addition, our sample is a subset of the population, not a complete accounting.

We will talk more about sampling in the next unit.

## 2.3 The Z-Distribution

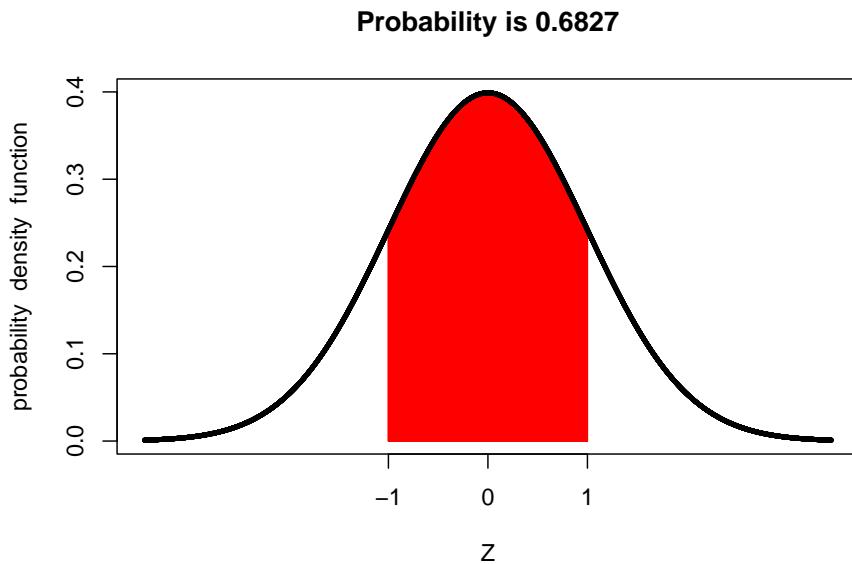
The relationship between probability and the normal distribution curve is based on the concept of the Z-distribution. In essence, the **Z-distribution** describes a

normal distribution curve with a population mean of 0 and a standard deviation of 1.

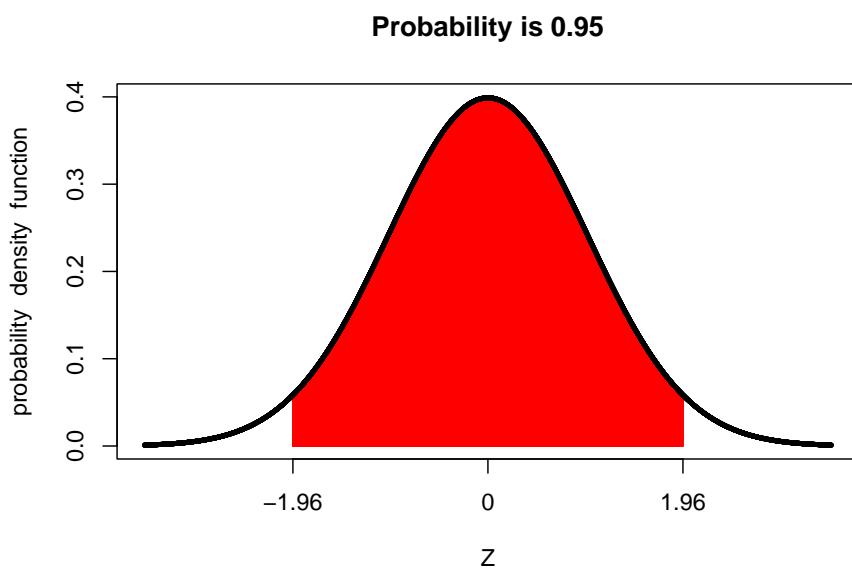


The Z-distribution helps us understand how probability relates to standard deviation in a normal distribution, regardless of the nature of a study or its measurement units.

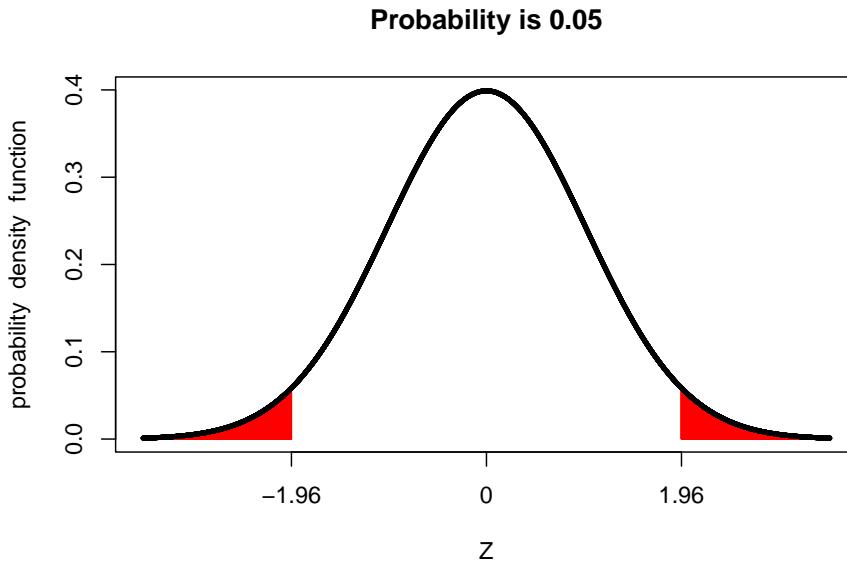
For example, the proportion of a population within one standard deviation of the mean is about 68 percent:



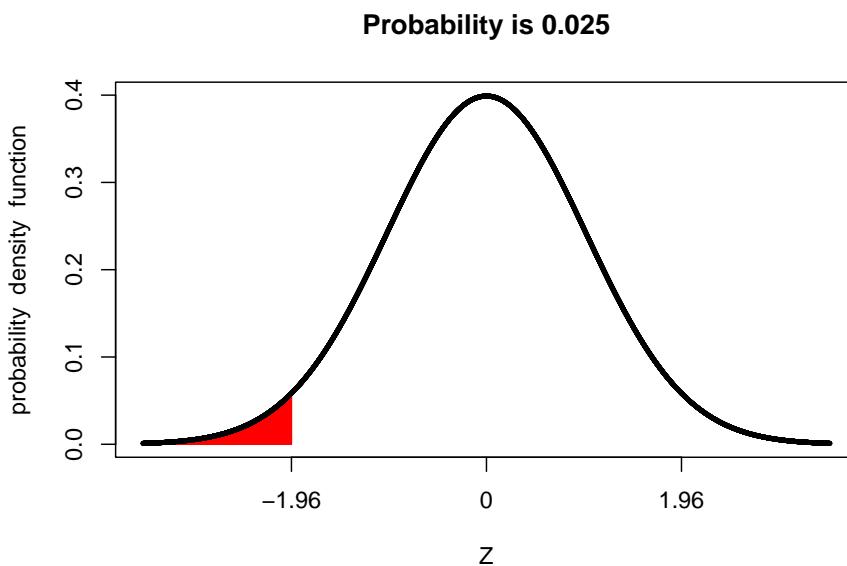
Similarly, the proportion of a population within 1.96 standard deviations of the mean is about 95 percent:



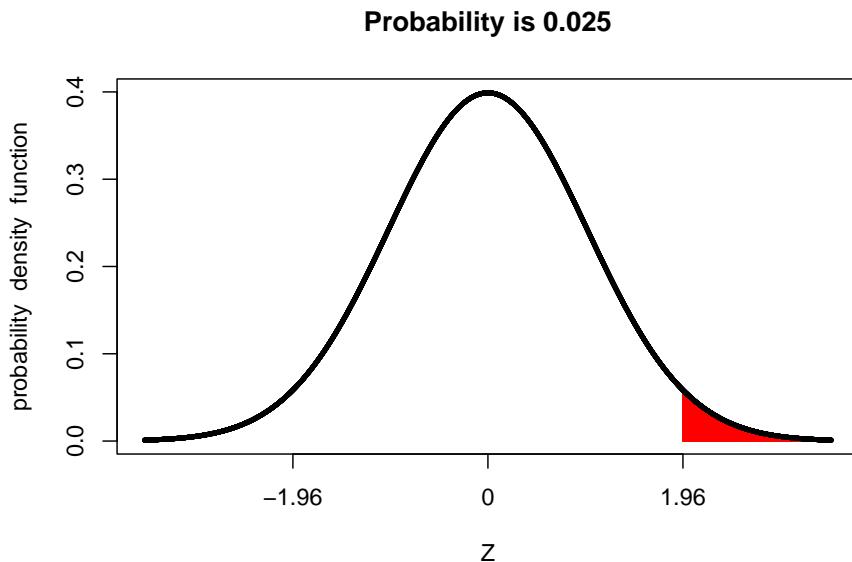
Conversely, the proportion of a population beyond 1.96 standard deviations from the mean is about 5 percent.



We refer to the upper and lower ends of the distribution as **tails**. In a normal distribution we would expect about 2.5% of observations to less than -1.96 standard deviations of the mean.



And 2.5% of the population to be more than +1.96 above the mean:



### 2.3.1 Important Numbers: 95% and 5%

Above we learned that 95% of a normal distribution is between 1.96 standard deviations of the mean, and that 5% of a normal distribution is outside this range. Perhaps these numbers sound familiar to you. Have you ever seen results presented with a 95% confidence interval? Have you ever read that two treatments were significantly different at the  $P=0.05$  level?

For population statistics, the normal distribution is the origin of those numbers. As we get further into this course, we will learn about additional distributions –  $t$  and  $F$  – and the unique statistical tests they allow. But the concept will stay the same: identifying whether observed statistical values are more likely to occur (i.e., within the central 95% of values expected in a distribution), or whether the values are unusual (occurring in the remaining 5%).

## Chapter 3

# Sample Statistics

In the previous two units, we studied populations and how to summarize them with statistics when the *entire* population was measured. In other words, the measure of center (mean) and measure of spread (standard deviation) were the summary of all observations.

In the case of a yield monitor map, these are appropriate statistics. In most every other agricultural reality, however, we cannot measure every individual in a population. Instead, we only have enough resources to collect a **sample** the population, that is, measure a subset of individuals from the population. In this case, we cannot measure the exact population mean or population standard deviation of the population. Instead, we can only estimate them using our **sample mean** or **sample standard deviation**.

This, of course, raises questions. Was the sample representative of the population? Would another random sample result in a similar estimate of the population mean or population standard deviation? And, perhaps, how much could our sample mean deviate from the population mean?

In other words, there is always uncertainty that statistics calculated from samples represent the true values of a population. You might even say we lack complete *confidence* that a sample mean will closely estimate the population mean.

Enter statistics. By taking multiple sets of samples, and calculating their means, we can use the differences among those sample means to estimate the distribution of sample means around the true population mean. Indeed, this is a fundamental concept of research and statistics – using the measured variance of samples to determine how accurate they are in predicting population statistics.

## 3.1 Samples

To measure the variation of sample means, we need at least two samples to compare. Ideally we can gather even more. As we will see, the more samples included in our estimates of the population mean, the more accurate we are likely to be.

A second comment, which may seem intuitive, but at the retail level may be overlooked, is randomization. Samples – for example individual plants or areas where yield will be measured – are ideally selected at random. In reality, the plants or areas selected for measures may be less than random. When I used to count weed populations, we used square quadrats (frames) to consistently define the area that was measured. We would throw them into different areas of the plot and count weeds where ever they landed.

The most important thing about selecting samples, however, is that the researcher work to minimize bias. Bias is when the samples selected consistently overestimate or underestimate the population mean. The most egregious example of this would be a researcher who consistently and purposely sampled the highest- or lowest-measuring parts of a field.

But bias can enter in other ways. For example, if our weed populations were very uneven, our thrown quadrat might be more likely to skid to a stop in weedy areas. A researcher might unconsciously choose taller plants to sample. In August, we might be tempted to sample a corn field from the edge than walk deep into that sweltering, allergenic hell.

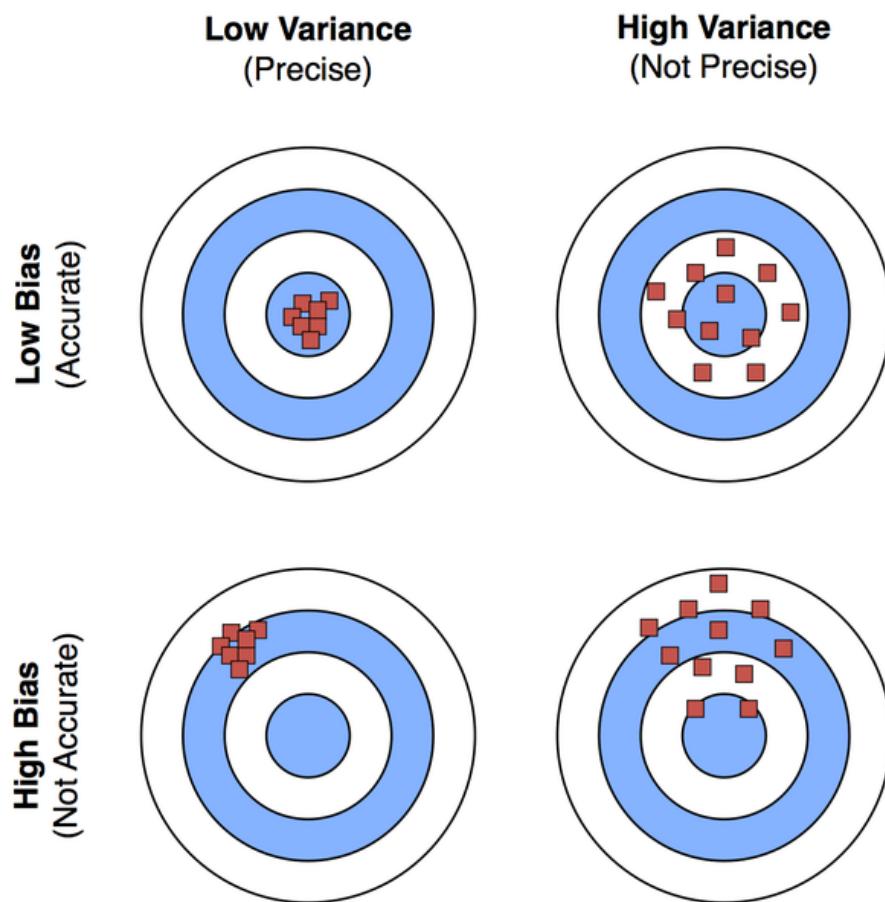
Remember, our goal is to generate estimates of population values that are as precise and accurate as our resources allow. **Precise** means our sample means have a low variance around the population mean. **Accurate** means our sample means are equivalently scattered above and below the population mean.

## 3.2 Case Study

Once more, we will work with the Iowa soybean yield dataset from Units 1 and 2.

Let's review the structure of this dataset:

```
## Simple feature collection with 6 features and 12 fields
## Geometry type: POINT
## Dimension: XY
## Bounding box: xmin: -93.15033 ymin: 41.66641 xmax: -93.15026 ymax: 41.66644
## Geodetic CRS: WGS 84
##   DISTANCE SWATHWIDTH VRYIELDVOL Crop  WetMass Moisture           Time
## 1  0.9202733          5  57.38461   174 3443.652      0.00 9/19/2016 4:45:46 PM
```



This work by Sebastian Raschka is licensed under a  
Creative Commons Attribution 4.0 International License.

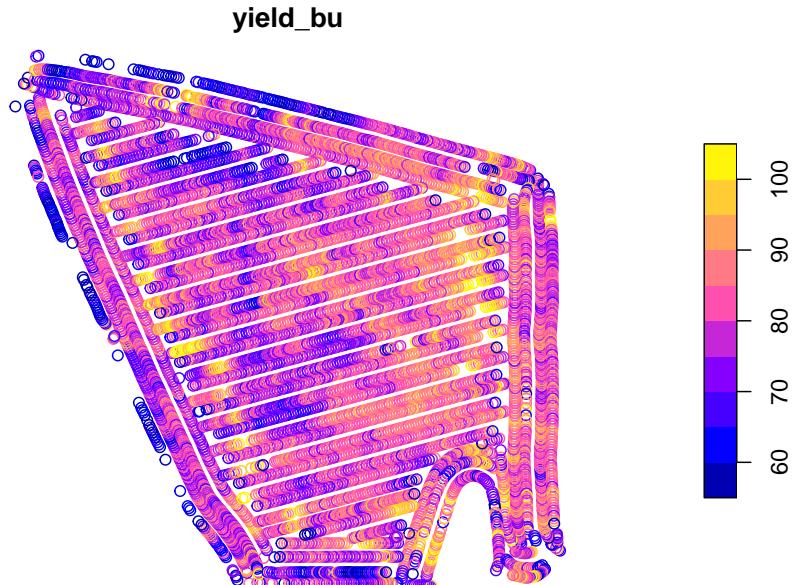
Figure 3.1: Accuracy versus Bias

```

## 2 2.6919269      5  55.88097 174 3353.411    0.00 9/19/2016 4:45:48 PM
## 3 2.6263101      5  80.83788 174 4851.075    0.00 9/19/2016 4:45:49 PM
## 4 2.7575437      5  71.76773 174 4306.777    6.22 9/19/2016 4:45:51 PM
## 5 2.3966513      5  91.03274 174 5462.851   12.22 9/19/2016 4:45:54 PM
## 6 3.1840529      5  65.59037 174 3951.056   13.33 9/19/2016 4:45:55 PM
##   Heading VARIETY Elevation           IsoTime yield_bu
## 1 300.1584 23A42 786.8470 2016-09-19T16:45:46.001Z 65.97034
## 2 303.6084 23A42 786.6140 2016-09-19T16:45:48.004Z 64.24158
## 3 304.3084 23A42 786.1416 2016-09-19T16:45:49.007Z 92.93246
## 4 306.2084 23A42 785.7381 2016-09-19T16:45:51.002Z 77.37348
## 5 309.2284 23A42 785.5937 2016-09-19T16:45:54.002Z 91.86380
## 6 309.7584 23A42 785.7512 2016-09-19T16:45:55.005Z 65.60115
##   geometry
## 1 POINT (-93.15026 41.66641)
## 2 POINT (-93.15028 41.66641)
## 3 POINT (-93.15028 41.66642)
## 4 POINT (-93.1503 41.66642)
## 5 POINT (-93.15032 41.66644)
## 6 POINT (-93.15033 41.66644)

```

And map the field:



In Unit 2, we learned to describe these data using the normal distribution model. We learned the area under the normal distribution curve corresponds to the proportion of individuals within a certain range of values. We also discussed

how this proportion allows inferences about probability. For example, the area under the curve that corresponded with yields from 70.0 to 79.9 represented the proportion of individuals in the yield population that fell within that yield range. But it also represented the probability that, were you to measure individual points from the map at random, you would measure a yield between 70.0 and 79.9.

### 3.3 Distribution of Sample Means

In the last unit, we sampled the yield from 1000 locations in the field and counted the number of observations that were equal to or greater than 70 and equal to or less than 80.

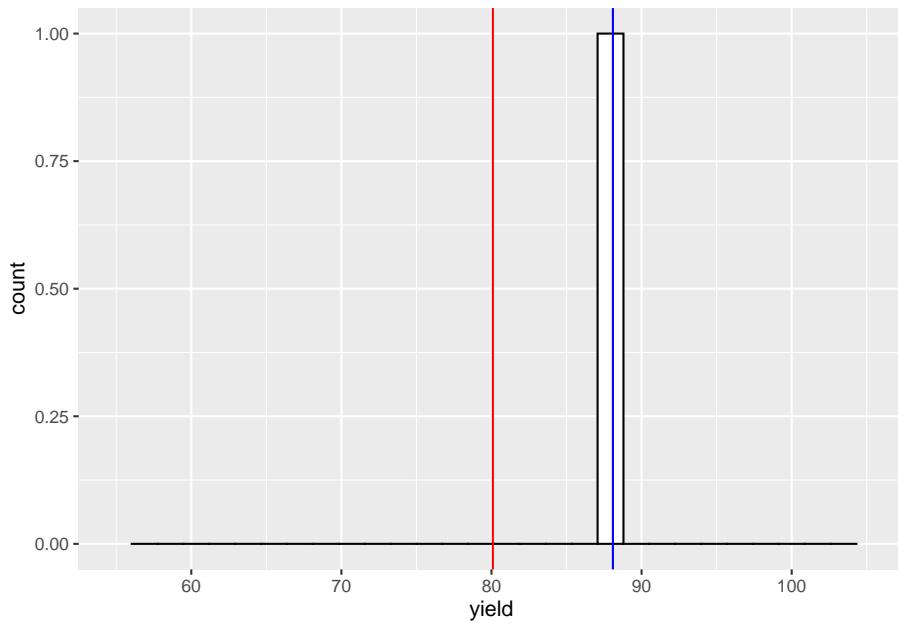
What would happen if we only sampled from one location. What would be our sample mean and how close would it be to the population mean?

In the histograms below, the red vertical line marks the population mean. The blue line marks the sample mean.

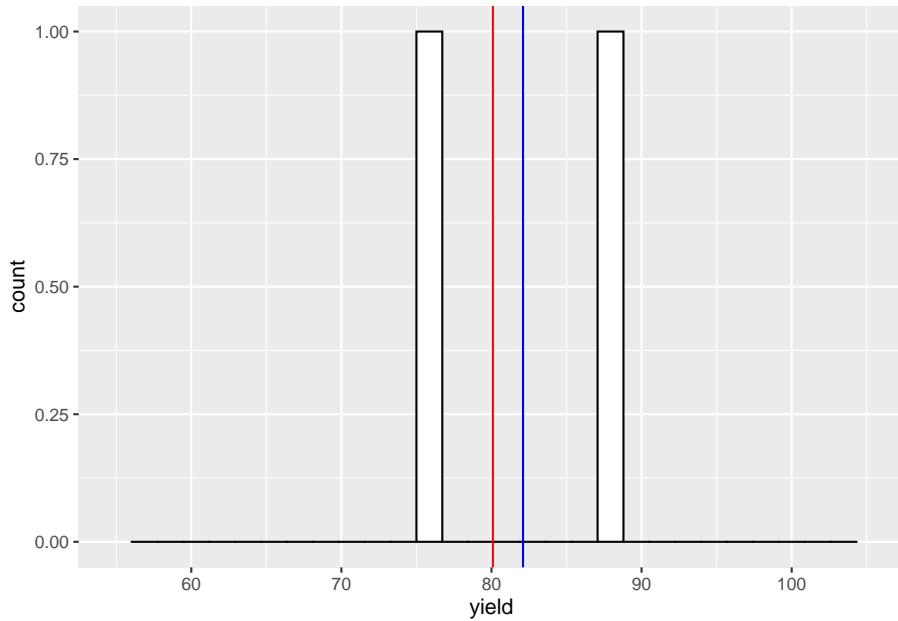
```
set.seed(1771)
yield_sample = sample(yield$yield_bu, 1) %>%
  as.data.frame()
names(yield_sample) = c("yield")
ggplot(yield_sample, aes(x=yield)) +
  geom_histogram(fill="white", color="black") +
  geom_vline(xintercept = mean(yield$yield_bu), color = "red") +
  geom_vline(xintercept = mean(yield_sample$yield), color = "blue") +
  lims(x=c(55,105))

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

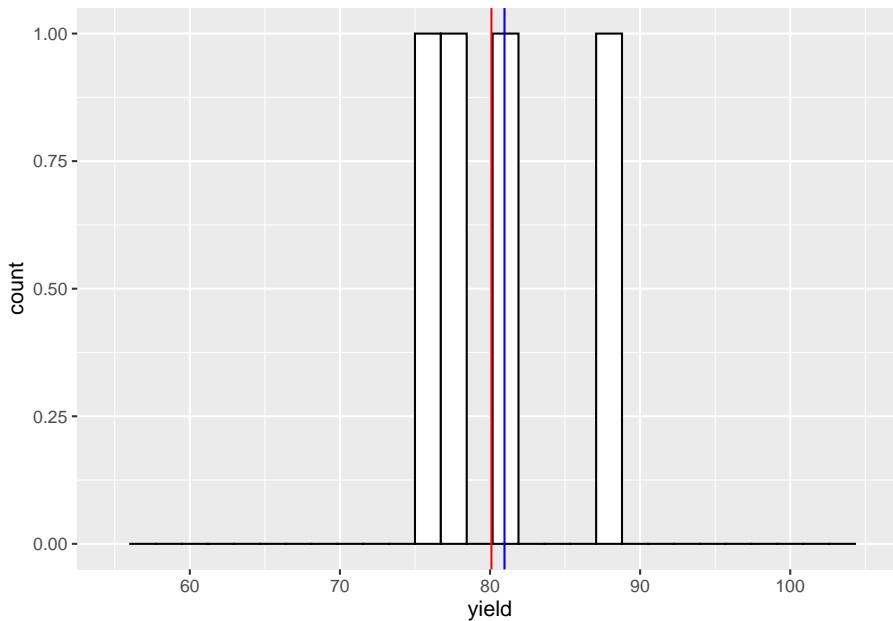
## Warning: Removed 2 rows containing missing values (geom_bar).
```



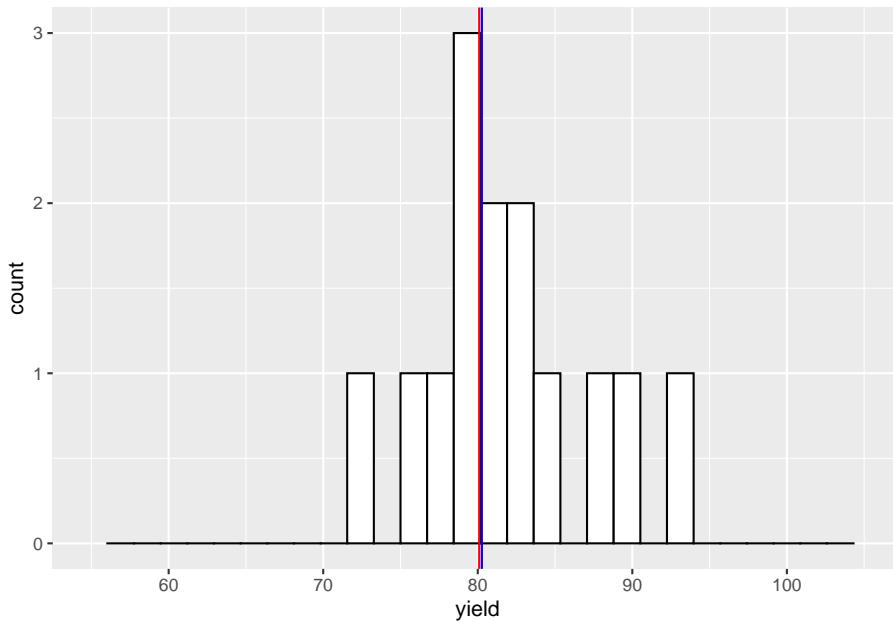
With one sample, our sample mean is about 8 bushels above the population mean. What would happen if we sampled twice?



Our sample mean is now about 2 bushels greater than the population mean. What would happen if we sampled four times?



Our sample mean is now about 1 bushel greater than the population mean. What would happen if we sampled 15 times?



The sample mean and population mean are almost equal.

Click on this link to access an app to help you further understand this concept: [https://www.rsimonjones.com/statistics/sampling-distribution-of-sample-means/app/](#)

[https://marin-harbur.shinyapps.io/03-sampling\\_from\\_normal\\_distn/](https://marin-harbur.shinyapps.io/03-sampling_from_normal_distn/)

### 3.4 Central Limit Theorem

The **Central Limit Theorem** states that sample means are normally distributed around the population mean. This concept is powerful because it allows us to calculate the probability that that a sample mean is a given distance away from the population mean. In our yield data, for example, the Central Limit Theorem allows us to assign a probability that we would observe a sample mean of 75 bushels/acre, if the population mean is 80 bushels/acre. More on how we calculate this in a little bit.

In our yield dataset, the population data are approximately normally distributed. It makes sense that the sample means would be normally distributed, too. But the Central Limit Theorem shows us that our sample means are likely to be normally distributed even if the population *does not* follow a perfect normal distribution.

Let's take this concept to the extreme. Suppose we had a population where every value occurred with the same frequency. This is known as a uniform distribution. Click on the following link to visit an app where we can explore how the sample distribution changes in response to sampling an uniform distribution:

<https://marin-harbur.shinyapps.io/03-sampling-from-uniform-distn/>

You will discover that the sample means are normally distributed around the population mean even when the population itself is not normally distributed.

### 3.5 Standard Error

When we describe the spread of a normally-distributed population – that is, all of the individuals about which we want to make inferences – we use the population mean and standard deviation.

When we sample (measure subsets) of a population, we again use two statistics. The **sample mean** describes the center of the samples.. The spread of the sample means is described by the **standard error of the mean** (often abbreviated to **standard error**). The standard error is related to the standard deviation as follows:

$$SE = \frac{\sigma}{\sqrt{n}}$$

The standard error, SE, is equal to the standard deviation ( $\sigma$ ), divided by the square root of the number of samples ( $n$ ). This denominator is very important

– it means that our standard error shrinks as the number of samples increases.  
Why is this important?

A sample mean is an estimate of the true population mean. The distribution of sample means the range of possible values for the population mean. I realize this is a fuzzy concept. This is the key point: by measuring the distribution of our sample sample means, we are able to describe the probability that the population mean is a given value.

To better understand this, please visit this link:

<https://marin-harbur.shinyapps.io/03-sample-distn/>

If you take away nothing else from this lesson, understand whether you collect 2 or 3 samples has tremendous implications for your estimate of the population mean. 4 samples is much better than 3. Do everything you can to fight for those first few samples. Collect as many as you can afford, especially if you are below 10 samples.

## 3.6 Degrees of Freedom

In Unit 1, the section on variance briefly introduced **degrees of freedom**, the number of observations in a population or sample, minus 1. Degrees of Freedom are again used below in calculating the **t-distribution**. So what are they and why do we use them? Turns out there are two explanations.

In the first explanation, *degrees of freedom* refers to the number of individuals or samples that can vary independently given a fixed mean. So for an individual data point to be free, it must be able to assume any value within a given distribution. Since the population mean is a fixed number, only  $n - 1$  of the data have the freedom to vary. The last data point is determined by the value of all the other data points and the population mean.

Confusing, huh? Who starts measuring samples thinking that the data point is fixed, in any case? But if you think about it, the purpose of the sample is approximate a real population mean out there – which is indeed fixed. It's just waiting for us to figure it out. So if our sample mean is equal to the population mean (which we generally assume), then the sample mean is also fixed. But it is a very weird way of thinking.

Yet this is the answer beloved by all the textbooks, so there you go.

The second answer I like better: samples normally *underestimate* the true population variance. This is because the sample variance is calculated from the distribution of the data around the sample mean. Sample data will always be closer to the sample mean – which is by definition based on the data themselves – than the population mean.

Think about this a minute. Your sample data could be crazy high or low compared to the overall population. But that dataset will define a mean, and the variance of the population will be estimated from that mean. In many cases, it turns out that using  $n - 1$  degrees of freedom will increase the value of the sample variance so it is closer to the population variance.

## 3.7 The t-Distribution

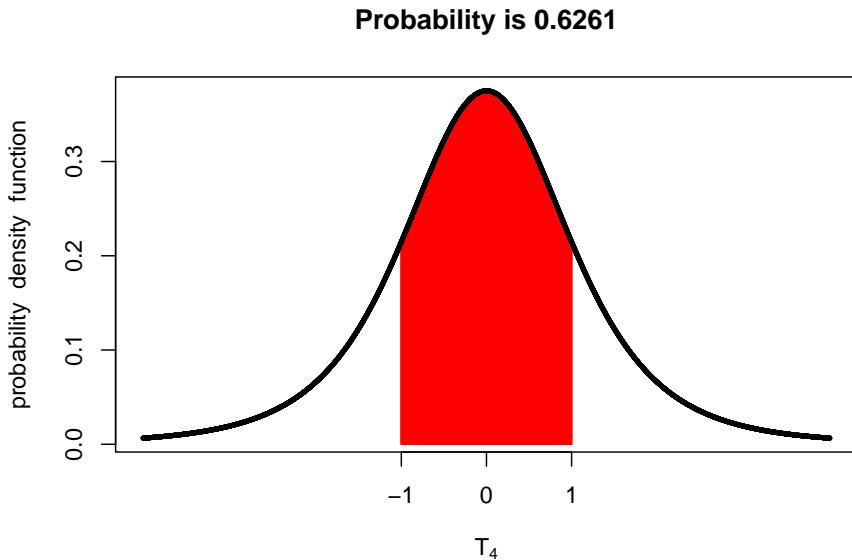
In the last unit, we used the *Z-distribution* to calculate the probability of observing an individual of a given value in a population, given its population mean and standard deviation. Recall that about 68% of individuals were expected to have values within one standard deviation, or  $Z$ , of the population mean. Approximately 95% of individuals were expected to have values within 1.96 standard deviations of the population mean. Alternatively, we can ask what the probability is of observing individuals of a particular or greater value in the population, given its mean and standard deviation.

We can ask a similar question of our sample data: what is the probability the population mean is a given value or greater, given the sample mean? As with the Z-distribution, the distance between the sample mean and hypothesized population mean will determine this probability.

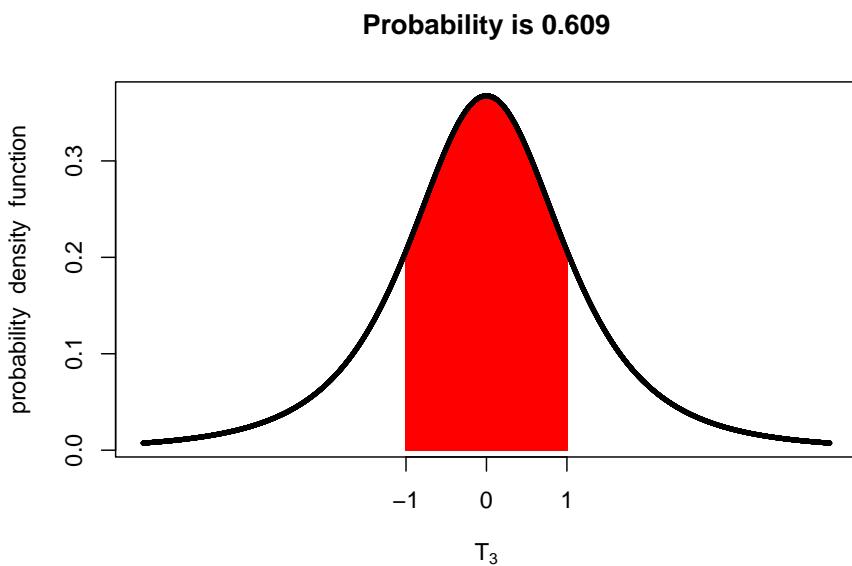
There is one problem, however, with using the Z-distribution: it is only applicable when the population standard deviation is *known*. When we *sample* from a population, we do not know its true standard deviation. Instead, we are estimating it from our samples. This requires we use a different distribution: the t-distribution.

Unlike the Z-distribution, the **t-distribution** changes in shape as the number of samples increases. Notice in the animation above that, when the number of samples is low, the distribution is wider and has a shorter peak. As the number of samples increases, the curve becomes narrower and taller. This has implications for the relationship between the distance of a hypothetical population mean from the sample mean, and the probability of it being that distant.

We can prove this to ourselves with the help of the `shadeDist()` function we used in Unit 2. You will learn how to plot the t-distribution in an exercise this week.

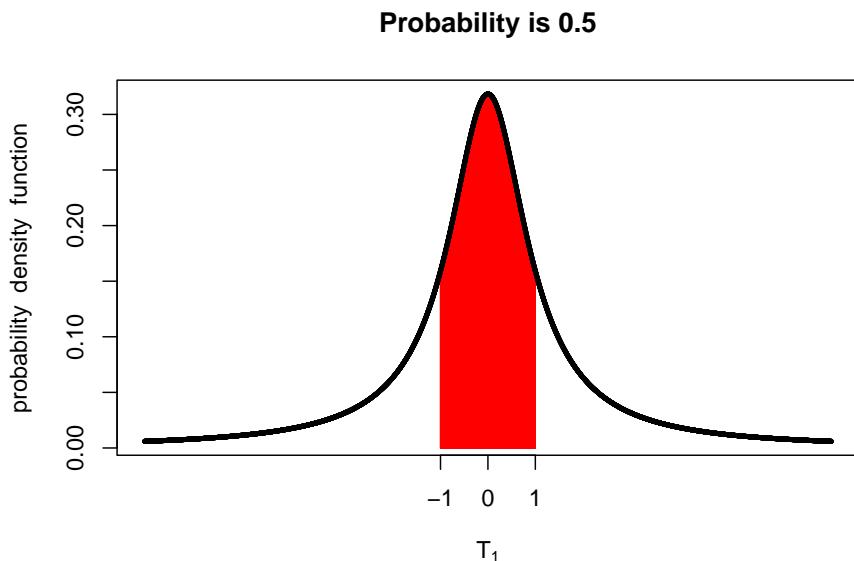


With 4 degrees of freedom, there is about a 63% probability the population mean is within 1 standard error of the mean. Let's decrease the sample mean to 3 degrees of freedom



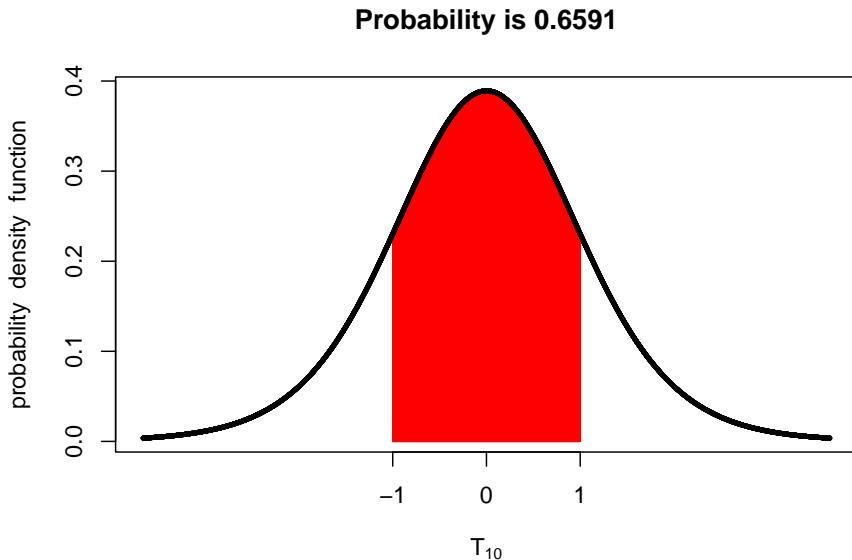
With only 3 degrees of freedom (4 samples), there is only a 61% probability the

population mean is within one standard error of the mean. What if we only had one degree of freedom (two samples)?

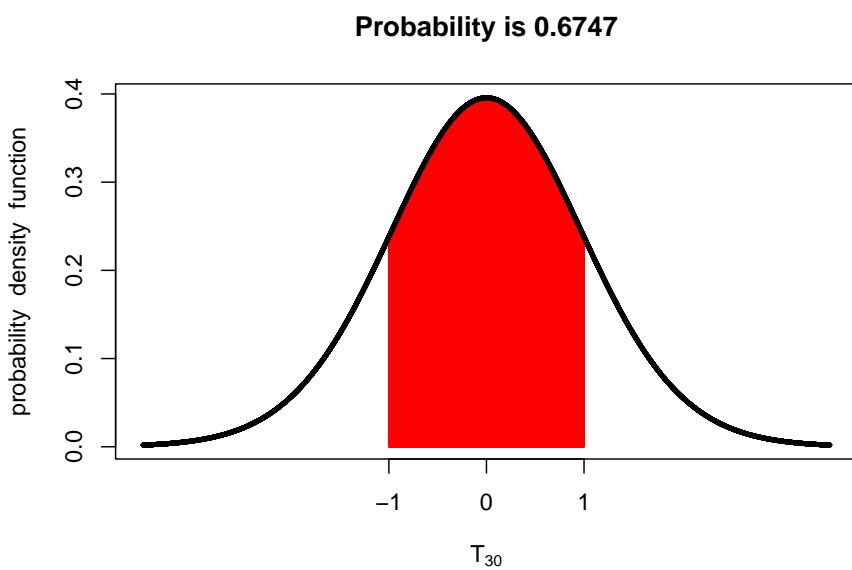


You should see the probability that the population mean is within 1 standard error of the sample mean falls to 50%.

If we have 10 degrees of freedom (11 samples), the probability increases to about 66%.

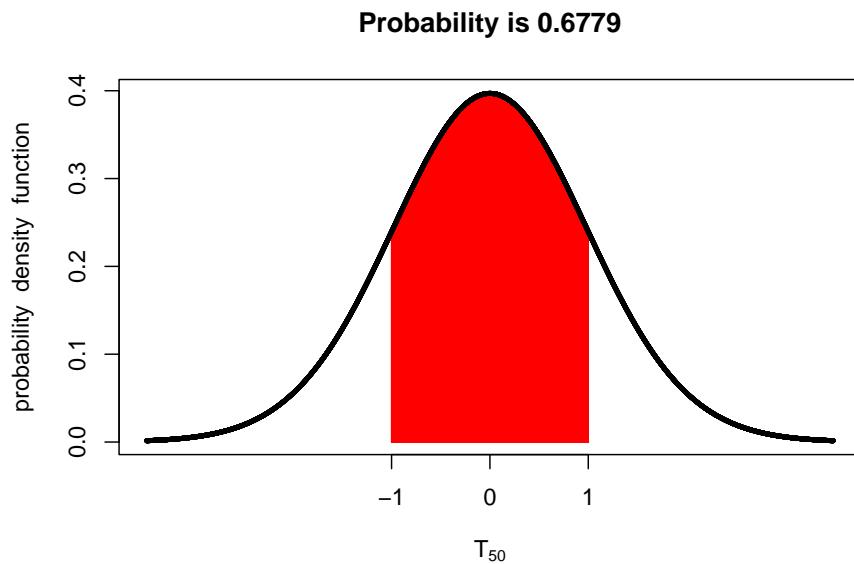


With 30 degrees of freedom the probability the population mean is within 1 standard error of the sample mean increases to 67%.

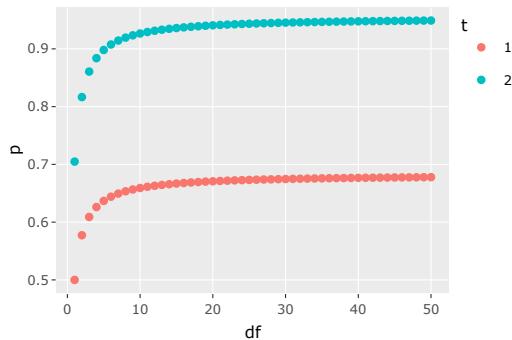


With 50 degrees of freedom (51 samples) the probability is about 68%. At this point, the t-distribution curve approximates the shape of the z-distribution

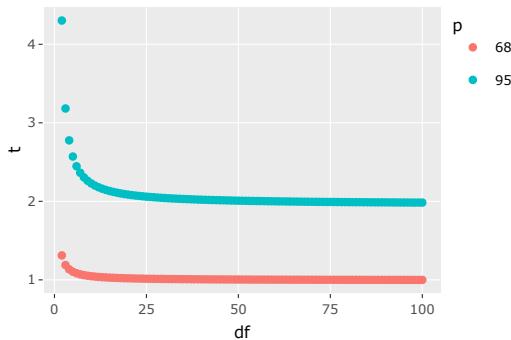
curve.



We can sum up the relationship between the t-value and probability with this plot. The probability of the population mean being within one standard error of the population mean is represented by the red line. The probability of the population mean being within 2 standard errors of the mean is represented by the blue line. As you can see, the probability of the population mean being within 1 or 2 standard errors of the sample mean *increases* with the degrees of freedom (df). Exact values can be examined by tracing the curves with your mouse.



Conversely, the t-value associated with a given proportion or probability will also decrease as the degrees of freedom increase. The red line represents the t-values that define the area with a 68% chance of including the population mean. The blue line represents the t-values that define the area with a 95% chance of including the population mean. Exact values can be examined by tracing the curves with your mouse. Notice the t-value associated with a 68% chance of including the population mean approaches 1, while the t-value associated with a 95% chance approaches about 1.98.



*Takeaway:* the number of samples affects not only the standard error, but the t-distribution curve we use to solve for the probability that a value will occur, given our sample mean.

## 3.8 Confidence Interval

The importance of the number of samples the standard error, and the t-distribution becomes even more apparent with the use of confidence interval. A **confidence interval** is a range of values around the sample mean that are selected to have a given probability of including the true population mean. Suppose we want to define, based on a sample size of 4 from the soybean field above, a range of values around our sample mean that has a 95% probability of including the true sample mean.

The 95% confidence interval is equal to the sample mean, plus and minus the product of the standard error and t-value associated with 0.975 in each tail:

$$CI = \bar{x} + t \times se$$

Where  $CI$  is the confidence interval,  $\bar{x}$  is the sample mean,  $t$  is determined by desired level of confidence (95%) and degrees of freedom, and  $se$  is the standard error of the mean

Since the t-value associated with a given probability in each tail decreases as the degrees of freedom increase, the confidence interval narrows as the degrees of freedom increase – even when the standard error is unaffected.

Lets sample our yield population 4 times, using the same code we did earlier. Although I generally try to confine the coding in this course to the exercises, I want you to see how we calculate the confidence interval:

1. Let's set the seed using `set.seed(1776)`. When R generates what we *call* a random sample, it actually uses a very complex algorithm that is generally unknown to the user. The `seed` determines where that algorithm starts. By setting the seed, we can duplicate our random sample the next time we run our script. That way, any interpretations of our sample will not change each time it is recreated.

```
# setting the seed the same as before means the same 4 samples will be pulled
set.seed(1776)
```

2. Now, let's take 4 samples from our population using the `sample()` function. That function requires has arguments. First, `yield$yield_bu` tells R to sample the `yield_bu` column of the `yield` data.frame. The second argument, `size=4`, tells R to take four samples.

```
# collect 4 samples
yield_sample = sample(yield$yield_bu, size=4)
#print results
yield_sample
```

```
## [1] 82.40863 71.68231 73.43349 81.27435
```

3. Next, we can calculate our sample mean and sample standard deviation. We will assign these values to the objects `sample_mean` and `sample_sd`.

```
sample_mean = mean(yield_sample)
sample_sd = sd(yield_sample)

sample_mean
```

```
## [1] 77.1997
```

```
sample_sd
```

```
## [1] 5.427144
```

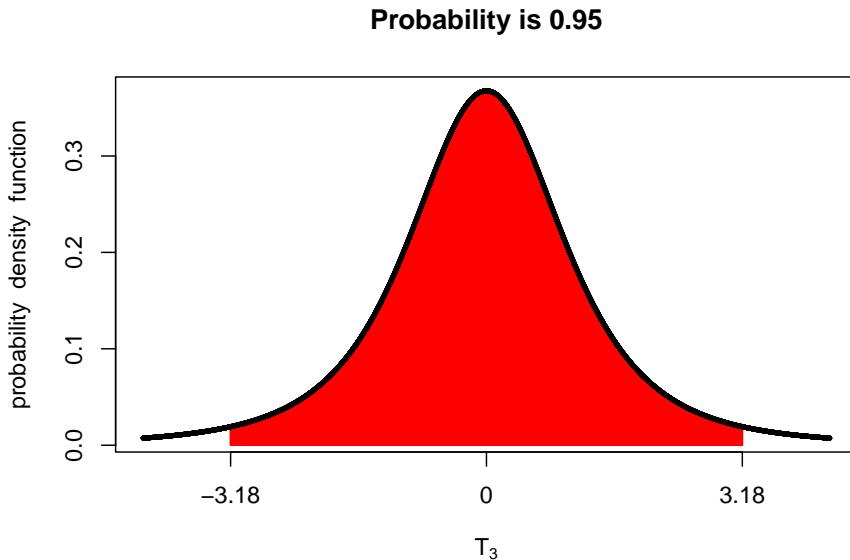
4. Finally, we can calculate the standard error, by dividing `sample_sd` by the square root of the number of observations in the sample, 4 . We assign this value to the object `sample_se`.

```
sample_se = sd(yield_sample)/sqrt(4)

sample_se
```

```
## [1] 2.713572
```

5. At this point, we have everything we need to calculate the confidence interval except the t-value. The middle 95% of the of the population will look like this:



There will be about 2.5% of the distribution above this range and 2.5% below it. We can calculate

```
# t-value associated with 3 df
upper_t = qt(p=0.975, df=3)
upper_t
```

```
## [1] 3.182446
```

6. Our lower limit is the t-value *below* which only 2.5% of the t-distribution occurs.

```
## [1] -3.182446
```

You will notice that "lower\_t", the t-value that measures from the sample mean to the lower limit

7. We can then calculate our upper confidence limit. The upper limit of the confidence interval is equal to:

$$\text{Upper CL} = \bar{x} + t \cdot SE$$

Where  $\bar{x}$  is the sample mean,  $t$  is the t-value associated with the upper limit, and  $SE$  is the standard error of the mean.

```
upper_limit = sample_mean + upper_t*sample_se
upper_limit
```

```
## [1] 85.83549
```

8. We can repeat the process to determine the lower limit.

```
lower_limit = sample_mean + lower_t
lower_limit
```

```
## [1] 74.01725
```

9. Finally, we can put this all together and express it as follows. The confidence interval for the population mean, based on the sample mean is:

$$CI = 81.6 \pm 3.2$$

We can also express the interval by its lower and upper confidence limits.

$$(78.5, 85.4)$$

We can confirm this interval includes the true population mean, which is 80.1.

### 3.9 Confidence Interval and Probability

Lets return to the concept of **95% confidence**. This means if we were to collect 100 sets of 4 samples each, 95% of them would estimate confidence intervals that include the true population mean. The remaining 5% would not.

Click on this link to better explore this concept:

<https://marin-harbur.shinyapps.io/03-confidence-interval/>

Again, both the standard error and the t-value we use for calculating the confidence interval decrease as the number of samples increase, so the confidence interval itself will decrease as well.

Click on the link below to explore how the size (or range) of a confidence interval changes with the number of samples from which it is calculated:

<https://marin-harbur.shinyapps.io/03-sample-size-conf-interval/>

As the number of samples increases, the confidence interval shrinks. 95 out of 100 times, however, the confidence interval will still include the true population

mean. In other words, as our sample size increases, our sample mean becomes less biased (far to either side of the population mean), and its accuracy (the proximity of the sample mean to the population mean) increases. In conclusion, the greater the number of samples, the better our estimate of the population mean.

In the next unit, we will use these concepts to analyze our first experimental data: a side by side trial where we will use the confidence interval for the difference between two treatments to test whether they are different.



## Chapter 4

# Two-Treatment Comparisons

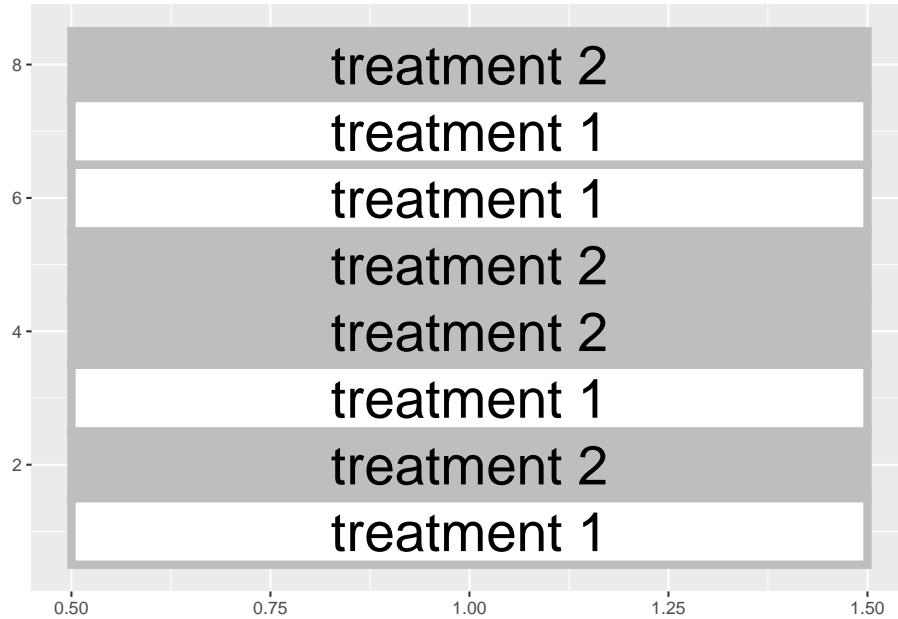
Until now, we have worked with a single population. To recap our progress: -

- In Unit 1, we learned a **population** is a complete group of individuals for which we wanted to develop a summary or prediction. We learned to describe the center of this population with the population mean and its spread using the sum of squares, variance, and standard deviation.
- In Unit 2, we used the **normal distribution** model to describe the pattern with which individuals in many populations are spread around the mean. Individuals closer in value to the population mean were predicted to occur more frequently than those further in value. Based on the frequency with which values were predicted to occur, we calculated the probability an individual from that population would fall within a given range of values.
- In Unit 3, we worked with **samples**, subsets drawn from a population. We saw how sample means are normally distributed, regardless of the population distribution from which they come. The standard error describes the spread of individual samples around the sample mean. This distribution was modeled with the t-distribution, which is wider when the number of samples is low and taller when the number of samples was greater.

### 4.1 Side-by-Side Trials

In this unit, we will finally put our statistical knowledge to work to test treatment differences. We will work with a simple but important experimental design

– the two-treatment comparison. Here in Ohio, this is referred to as a side-by-side trial, but you may have a different term for it where you work. If you work in retail agronomy, you have probably conducted these trials. Typically, you would split one or more fields into treated and untreated fields. For example, you might “stripe” an individual field with treated and untreated areas:



Or, you might divide multiple fields in half like this:



In either case, a side-by-side trial deals with two treatments and can be analyzed using a t-test. In these next two units we will compare different designs for side-by-side (or paired) trials, use R to randomly assign treatments, understand how the t-distribution allows us to test for significance, and run these tests in R.

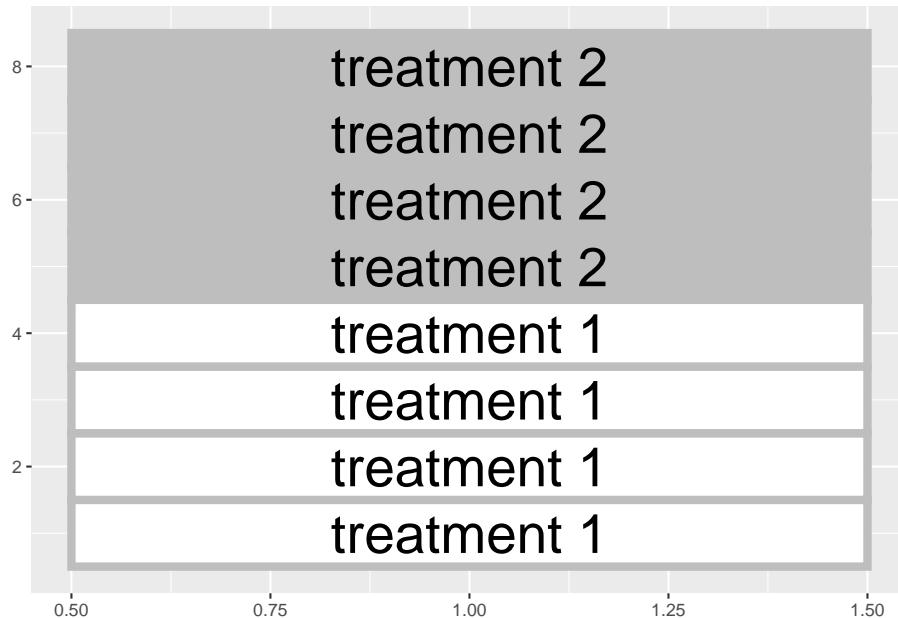
## 4.2 Blocked Design

In Unit 1 we learned the hallmarks of a designed trial are the **randomization** and **replication** of treatments. Each treatment should be observed several times in different experimental units. In our work, often **experimental unit** is a fancy term for a plot, half a field, or a pot.

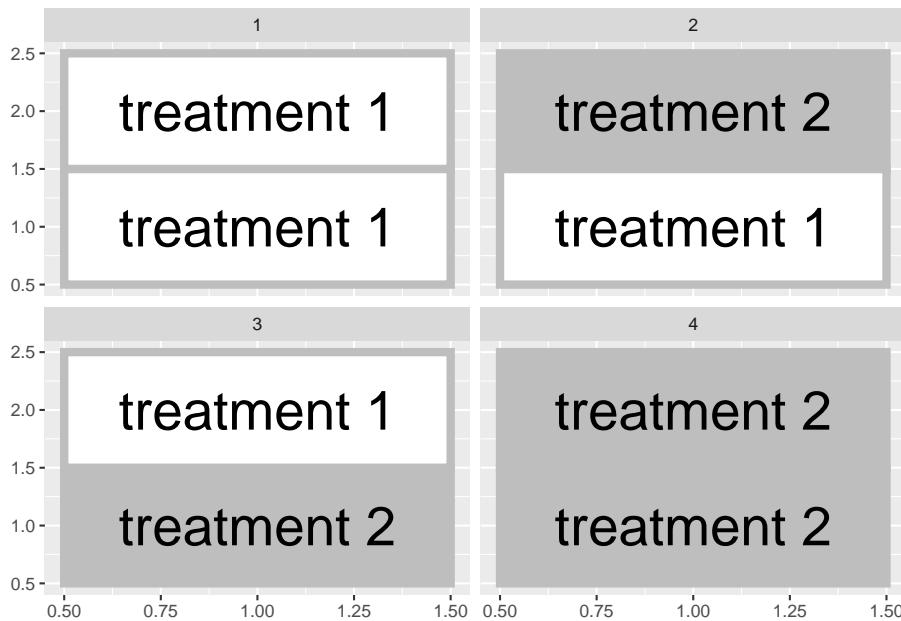
Observing the treatment several times has two benefits. First, the average of those observations – the sample mean – will likely be closer to the true population average for that treatment than the individual observations. Second, variation among the observations can be used to give us a sense how much environmental factors – in contrast with our treatment – cause our observations to vary.

It is also important to randomize treatments in order to reduce intentional or unintentional biases that might skew our interpretation of results. For example, one treatment might be biased by always putting it to the north of another treatment or, more insidiously, in the better half of a field. Deliberate randomization reduces the probability of either scenario.

That said, reality sometimes intervenes. Soil types change across a field, as do microclimates around field edges and management histories (e.g. an old feedlot). Though randomization reduces the likelihood that treatments are concentrated in one areas, it may not produce as even a distribution of treatments across the experimental area as we would like. We could conceivably end up with all replicates of a treatment level concentrated in one half of the field:



Similarly, if we are using multiple fields, both halves of a field could receive the same treatment in a randomized design



**Blocked** experimental designs place a restriction on the random assignment of treatments to plots. Instead of assigning treatments randomly within a field or across a state. We instead force both treatments to occur within a field section or in the same field, so that our treatment maps look more those we showed in the first two figures of this unit.

Here is another way to think about this:

- Our statistical test this week is based on comparing samples from two populations, the control population and the population receiving starter.
- We want to design our experiment so that aside from the effect of the treatment, the two populations are as identical as possible.

The blocked approach, in general, helps create two populations that are similar.

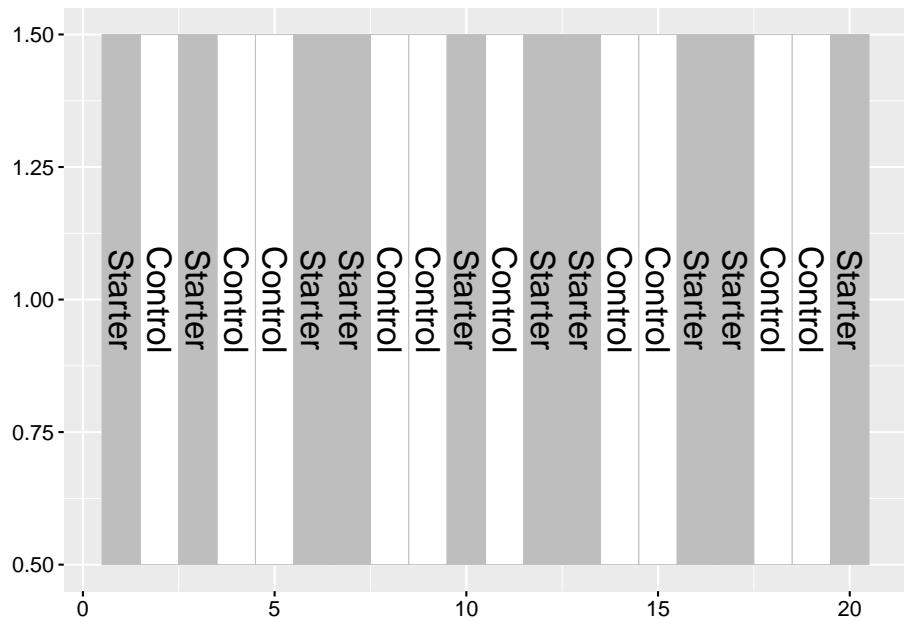
In the exercise `exercise_randomizing_plots` in R, you will learn how to design your own blocked two-treatment trial.

## 4.3 Case Study

An in-furrow corn starter (*6-24-6 plus micronutrients*) was tested against a control (*no starter*) in a trial in western Ohio. Treatments were blocked (paired) so that each treatment occurred once per block. Plot numbers are given on the

x-axis of the map below. There were 10 blocks. The data are in a data.frame named “corn\_starter”.

```
corn_starter = read.csv("data-unit-4/corn_starter.csv")
```



Here are the first six rows of our dataset.

```
head(corn_starter)
```

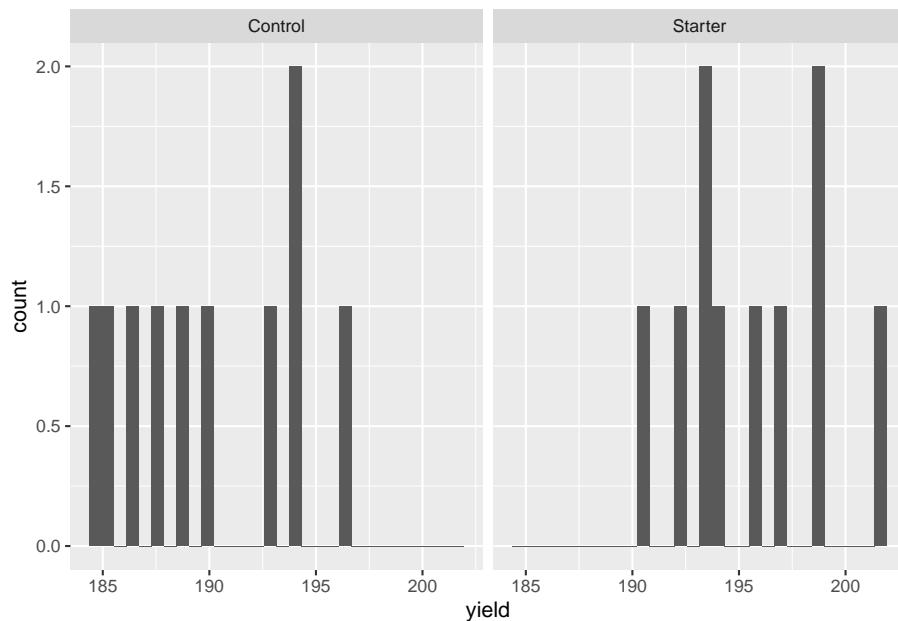
```
##   block plot treatment yield
## 1     1    11   Starter 193.4
## 2     1    12   Control 194.2
## 3     2    21   Starter 192.2
## 4     2    22   Control 189.0
## 5     3    31   Control 193.8
## 6     3    32   Starter 194.2
```

Note the column headings: *treatment* refers to the treatment level (Control or Starter) and *yield* refers to the measured yield.

Let’s make a quick histogram plot of the data using ggplot. For the first time, in this unit, we are working with samples from two populations: the control and treatment populations. We will have separate histograms for each set of samples.

```
ggplot(data = corn_starter, aes(x=yield)) +
  geom_histogram() +
  facet_grid(~treatment)

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



The mean yield of the sample from the population that received starter is greater than the mean yield of the sample of the population that received the control (no starter).

```
corn_starter %>%
  group_by(treatment) %>%
  summarise(yield = mean(yield))

## # A tibble: 2 x 2
##   treatment    yield
##   <chr>        <dbl>
## 1 Control      190.
## 2 Starter      196.
```

Instead of studying the control and treatment sample sets independently, however, we can restructure our dataset to analyze it another way. Data set structure describes how our columns and rows are organized. Do our treatment

levels, for example, differ down rows or across columns? In the dataset above, for example, our two treatment levels (*control* and *starter*) vary among rows in the *treatment* column.

When we **restructure** a dataset, we rearrange it so that values that formally varied among rows vary across columns, or vice versa. If you have every used a *PivotTable* in *Excel*, you have restructured data. In **exercise\_restructuring\_data** in RStudio, you will learn how to similarly restructure data in R.

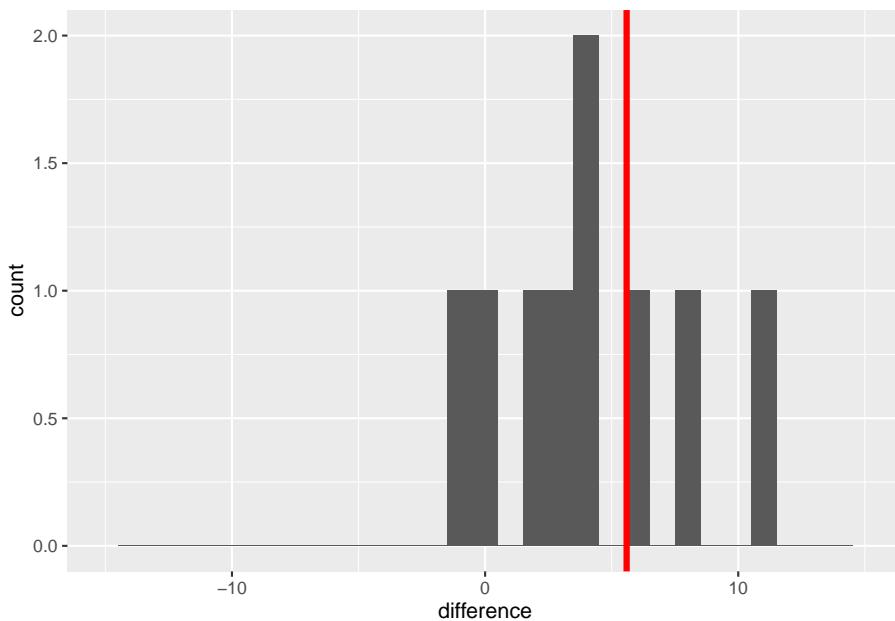
In our case study, we will restructure the treatment column by separating its values among two new columns, *control* and *starter*. We will create a new *difference* column by subtracting the yield of the control plot from the yield of the starter plot in each block. By doing this, each block is now summarized by one value, the difference between treatments.

```
##   block Control Starter difference
## 1      1    194.2   193.4     -0.8
## 2      2    189.0   192.2      3.2
## 3      3    193.8   194.2      0.4
## 4      4    186.4   190.5      4.1
## 5      5    196.4   198.7      2.3
## 6      6    189.8   193.4      3.6
## 7      7    187.6   196.0      8.4
## 8      8    185.3   196.7     11.4
## 9      9    184.5   201.5     17.0
## 10     10   192.6   198.9      6.3
```

In this new population, individuals with a positive value indicate a block where the starter out-yielded the control. Negative differences indicate the control out-yielded the starter. We can plot these differences in a histogram, also.

```
## Warning: Removed 1 rows containing non-finite values (stat_bin).

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



Most of the individuals in this new population are greater than 0. The population mean, indicated by the vertical red line, is 5.6 bushels/acre. So far, our starter shows promise: a 5.6 bushel/acre mean *increase* in yield. Of course, this immediately raises the next questions:

- How consistently is there a positive response to this starter product?
- If we repeated this experiment, would we likely see a difference equal to zero or – even worse – one less than zero, in which case, yield actually *decreased* with the use of starter fertilizer?

We can use the t-distribution to answer this question, either by calculating the confidence interval of the mean difference, 5.6 bushels/acre, or testing the difference between the control and treatment means using a t-test.

## 4.4 Confidence Interval

In Unit 3, we worked with the sample mean, which is based on individual samples from an individual population. We defined the confidence interval around a sample mean, based on the variation in sample values around that sample mean.

We can use the same approach now on our sample differences: we will calculate the confidence interval around the mean difference between our control and treatment. If that interval contains zero or negative values, that means it is

possible that control and treatment yields are the same, or even that the control yields more than the treatment.

Recall that to calculate the confidence interval we need two values:

- the minimal t-value that: is associated with our degrees of freedom and is expected to encompass 95% of the distribution of sample means
- the standard error of the difference.

We calculate the minimal t-value using the `qt()` function in R.

```
min_t = qt(0.975, df=9)
min_t
```

```
## [1] 2.262157
```

Remember, degrees of freedom is equal to  $n-1$ . Our new population has 10 degrees of freedom, therefore, `df` equals 9 above. The minimal t-value is about 2.26 .

We also need to know the standard error of the population. We will calculate that the same as we did for a single variable in Unit 3. Given our sample set is composed of treatment differences, though, we will call this statistic the *standard error of the difference*.

First, we calculate the standard deviation of the treatment differences using the `sd()` function:

```
sd = sd(corn_differences$difference)
sd
```

```
## [1] 5.404001
```

Next, we divide the standard deviation by the square root of the number of differences. This is equal to the number of blocks.

Then we divide the standard deviation by the square root of the number of observations (10) to get the standard error of the difference:

```
sed = sd/sqrt(10)
sed
```

```
## [1] 1.708895
```

Finally, since the confidence interval is constructed with the sample mean as its center, we calculate the sample mean:

```
sample_mean = mean(corn_differences$difference)
```

Our sample mean is 5.59 and the standard error of the difference is about 1.71  
We additionally calculated pop\_mean, the mean of our population.

We can now calculate the confidence interval, using `sample_mean`, `min_t`, and `sed`. To calculate the lower limit, we subtract the product of `min_t` and `sed` from the sample mean:

```
lower_limit = sample_mean - (min_t * sed)
lower_limit
```

```
## [1] 1.724211
```

Similarly, the upper limit is calculated by adding the product of `min_t` and `sed` to the sample mean:

```
upper_limit = sample_mean + (min_t * sed)
upper_limit
```

```
## [1] 9.455789
```

It is common practice to present this confidence interval in parentheses, with the lower and upper limits separated by commas.

```
CI = paste0("(", lower_limit, ",",
           upper_limit, ")")
CI
```

It is also good practice to round your final results so that your answers have either the same number of decimal places as the original measures. By doing this, we more fairly reflect the precision of the original measurements.

```
CI = paste0("(", round(lower_limit,1), ",",
           round(upper_limit,1), ")")
CI
```

The 95% confidence interval ranges from 1.7 to 9.5 bushels/acre and, notably, does not include zero. Since zero is outside the 95% confidence interval, there is greater than a 95% probability the population mean is not equal to zero. Another way of saying this is there is less than a 5% probability that the population mean is equal to zero. Or, finally, the population mean is significantly different from zero at the 5% (or 0.05) level.

Going our population was composed of the difference of starter and control, we can say the starter effect is significantly different from the control at the 5% level.

Learn how to construct your own confidence interval in `exercise_confidence_interval` in RStudio.

## 4.5 T-Test

An alternative to the confidence interval is the t-test. The first step of the t-test is to calculate our observed t-value:

$$t = \frac{\bar{x} - \mu}{SED}$$

Where  $\bar{x}$  is the observed population mean,  $\mu$  is the hypothesized population mean (usually 0, meaning no treatment difference), and  $SED$  is the standard error of the difference.

In our example above:

$$t = \frac{5.59 - 0}{1.71} = 3.27$$

Our observed t-value is about 3.27. If there were no difference between the starter and the control, of course,  $t$  would be equal to zero. So there is a difference between our observed mean and zero of  $t = 3.27$ . Using R, we can quickly calculate the probability of observing a value  $t = 3.27$  above – or below our population mean of 5.59.

```
library(fastGraph)
shadeDist(xshade=c(-3.27, 3.27), "dt", parm2 = 9, lower.tail = TRUE)
```

Note that this time with this function we used the argument “lower.tail=TRUE”. This tells R to calculate the probability of values futher than  $t = 3.27$  both above and below the mean.

Again there were 10 observed differences in our population, so there were 9 degrees of freedom. The value that is returned is the probability the population mean is actually zero, given the population mean. The probability of this t-value (sometimes abbreviated as  $Pr \geq |t|$ ) is very low, less tan 0.01, or 1%.

Learn how to conduct your own t-test in `exercise_t_test` in RStudio.

## 4.6 Conclusion

In this unit, we learned to create the experimental design for a two-treatment test and compare two treatments using the confidence interval of the treatment differences and the t-test.

Before the first plot is planned, however, we must consider what is the objective of the experiment. Specifically, what question is it designed to address? What are the potential answers to that question, and how should they be tested to determine which answer is most probably correct?

In Unit 5, we will learn about the nuances of developing research questions and expressing them in hypotheses we can then test with experimental data.



## **Chapter 5**

# **Understanding Statistical Tests**

In the last unit, we introduced the concept of statistical testing and, although I endeavor to make this course as practical and painless as possible, I believe it worthwhile to spend a unit on some of the theory of statistical testing. This will help reinforce what we have learned so far in this course, and prepare us for the other statistical tests that lie ahead. Otherwise, it is easy to become ambiguous about what we are really testing, and unclear in reporting our results.

In the last unit, we discussed experimental design and quickly jumped into data analysis. This unit, we will walk through the thought processes that surround our trial, including: - identifying our research question - creating a model from our question - identifying the hypotheses our data will be used to test - recognizing that we can mistakenly accept or reject these hypotheses - understanding how the confidence interval and p-value describe our measured difference - incorporating pre-existing knowledge into our hypotheses and tests

This list seems intimidating, but we will take our time and break these down into as much plain language as statistics will allow.

### **5.1 Research Question**

As I have likely said before in this course, the first think you must have to design an experiment is a clear, testable research question. The question should be answerable using quantitative data and specific about what those data will measure. Here are some examples of bad and good questions:

Bad: Is this fertilizer better than another? Good: Does corn treated with 6-24-6 fertilizer at planting differ in yield from corn that is not treated with an

in-furrow starter.

Bad: Is the winter wheat variety MH666 (“the Beast”) different than variety MH007 (“the Bond”)? Good: Does variety MH666 differ in test weight from variety MH007 ?

Bad: Does herbicide deposition agent “Stick-It!” perform differently than agent “Get-Down!” ? Good: Do “Stick-It!” and “Get-Down!” differ in the number of live weeds two weeks after their application with glyphosate?

Remember to be clear about what we are measuring. Otherwise, it is unclear whether we are testing fertilizer affects on corn yield or moisture at harvest. We don’t know whether we are comparing winter wheat yield or head scab. We don’t know whether we are measuring the effect of our deposition agent on weed survival or crop injury.

## 5.2 The Model

The word “model” probably makes you shudder and think of a crowded blackboard filled with mathematical equations.

Yes, models can be quite complex. All of you have worked with models, however, and most of you should recall this one:

$$y = b + mx$$

Where  $y$  is the vertical coordinate of a point on a graph,  $x$  is its horizontal coordinate, and  $b$  is the Y-intercept (where the line crosses the  $y$ -axis). The most interesting variable is often  $m$ , the slope. The slope is the unit increase in  $y$  with each unit increase in  $x$ .

Suppose we took a field of corn and conducted a side-by-side trial where half of the plots were sidedressed with 150 lbs treated with an N stabilizer. The other half were sidedressed with 150 lbs actual N plus 1 unit of nitrogen stabilizer. The mean yield of plots treated with N plus nitrogen stabilizer was 195 bu and the mean yield of plots treated with N alone was 175 bu. How could we express this with a slope equation?

First, let’s state this as a table. We will express the N stabilizer quantitatively. The “No Stabilizer” treatment included zero units of N stabilizer. The “Stabilizer” treatment received 1 unit of stabilizer.

```
library(tidyverse)
```

```
## -- Attaching packages ----- tidyverse 1.3.1 --
```

```

## v ggplot2 3.3.5      v purrr   0.3.4
## v tibble  3.1.6      v dplyr    1.0.8
## v tidyverse 1.2.0     v stringr  1.4.0
## v readr   2.1.2      vforcats  0.5.1

## Warning: package 'ggplot2' was built under R version 4.1.3

## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()   masks stats::lag()

set.seed(1776)
`No Stabilizer` = rnorm(4, mean=175, sd = 5)
`Stabilizer` = rnorm(4, mean = 195, sd=5)

original_data = cbind(`No Stabilizer`, `Stabilizer`) %>%
  as.data.frame()

means_table = original_data %>%
  gather(Treatment, Yield) %>%
  group_by(Treatment) %>%
  summarise(Yield = mean(Yield)) %>%
  ungroup() %>%
  mutate(Yield = round(Yield, 1)) %>%
  mutate(Nitrogen = c(0,1)) %>%
  select(Treatment, Nitrogen, Yield) %>%
  column_to_rownames(var = "Treatment")

mean_yield = mean(means_table$Yield)

means_table

##           Nitrogen Yield
## No Stabilizer      0 177.7
## Stabilizer        1 198.0

# yield = c(175, 195, 185)
# nitrogen = c(0, 1, 0.5)
# original_data = cbind(nitrogen, yield) %>%
#   as.data.frame()
# rownames(original_data) = c("No Stabilizer", "Stabilizer", "Mean")
#
# original_data

```

In creating this table, we also calculated the mean stabilizer rate and corn yield across all plots. These are are population means for the field.

Now, let's express the stabilizer rate and yield little differently, this time by their differences from their population mean. In half of the plots, the N stabilizer rate was 0.5 less than the population mean of 187.9; in the other half, the rate was 0.5 greater. Similarly, the yield in half of the plots was about 10 bushels less than the population mean of 188.9; in the other half of the plots, it was 10 bushels greater. Our table now looks like this:

```
centered_data = means_table %>%
  scale(scale = FALSE) %>%
  as.data.frame() %>%
  mutate(Yield = round(Yield, 1))

centered_data

##           Nitrogen Yield
## No Stabilizer     -0.5 -10.2
## Stabilizer       0.5  10.2

#
# yield = c(-10, 10, 0)
# nitrogen = c(-0.5, 0.5, 0)
# centered_data = cbind(nitrogen, yield) %>%
#   as.data.frame()
# rownames(centered_data) = c("No Stabilizer", "Stabilizer", "Mean")
#
# centered_data
```

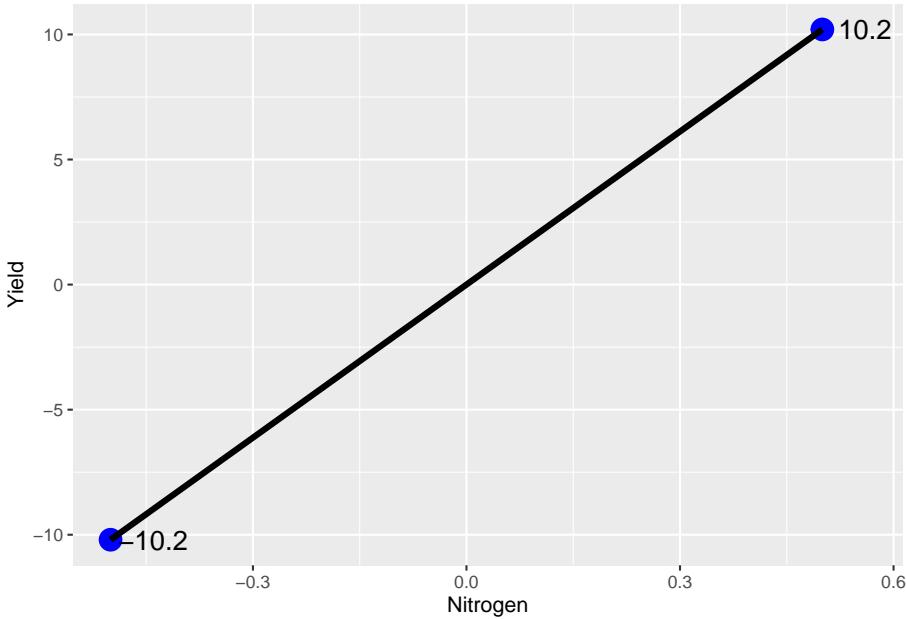
What we have just done is a statistical process called *center scaling*. Centering expresses measures by their distance from the population mean, instead of as absolute values.

Now let's plot this out using our line equation.  $y$  equals yield.  $x$  equals nitrogen rate.  $b$  equals the mean yield, the y-intercept, which is zero in our centered data.

```
library(ggpubr)

means_plot = centered_data %>%
  ggplot(aes(x = Nitrogen, y=Yield)) +
  geom_point(color="blue", size =5) +
  geom_line(size = 1.5)
```

```
means_plot +
  geom_text(aes(x = Nitrogen + 0.06, y = Yield, label = Yield), size = 5)
```



Ta da: our line plot. If we were to write this as a linear equation, it would be:

$$Yield = 0 + 20 * Stabilizer$$

Thus, as the N stabilizer rate increase from 0 (no stabilizer) to 1 (stabilizer), yield increases 20 bushels.

### 5.2.1 Treatment Effect

Another way of expressing the effect of the treatment levels is in terms of their distance from the mean yield across all plots. Where sidedressed with nitrogen alone, our mean yield is equal to the population mean minus 10. Conversely, where we sidedressed with nitrogen plus stabilizer, our yield is the population mean *plus* 10.2. We can express these treatment effects as:

$$Unstabilized : T_0 = -10.2$$

$$Stabilized : T_1 = +10.2$$

Our mean yield when corn is sidedressed with N without stabilizer is then equal to the mean yield across all plots plus the treatment effect:

$$\text{Unstabilized} : Y_0 = \mu + T_0$$

$$\text{Stabilized} : Y_1 = \mu + T_1$$

We can prove this to ourselves by plugging in the actual yields for  $Y$  and  $\mu$  and the actual treatment effects for  $T_0$  and  $T_1$ :

$$\text{Unstabilized} : 175 = 185 + (-10.2)$$

$$\text{Stabilized} : Y_1 = 185 + (+10.2)$$

### 5.2.2 Error Effect

The treatment effect is known as a *fixed* effect: we assume it will be consistent across all plots within our trial. That said, will every plot that receives nitrogen plus stabilizer will yield 195 bushels? Will every field sidedressed with nitrogen without stabilizer yield 175?

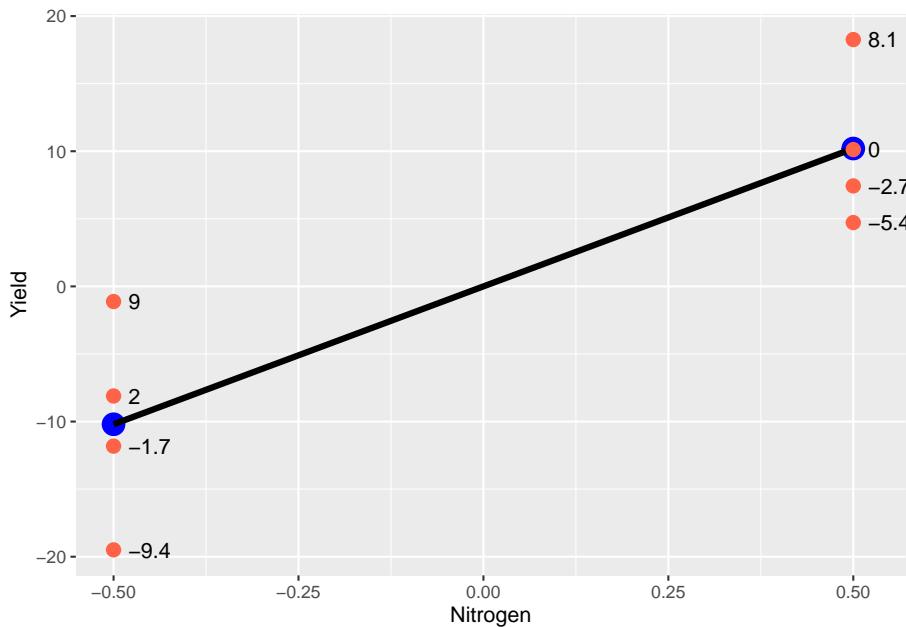
Of course not. Any yield map will show variation in yield within a field, even when the entire field has been managed uniformly. Differences in soil texture and microclimates, inconsistencies in field operations, and inaccuracies in measuring equipment contribute to variations in the values recorded. These variations will also add to or subtract from the mean yield across all plots.

We can visualize this in the plot below.

```
centered_original_data_by_trt = original_data %>%
  gather(Nitrogen, Yield) %>%
  mutate(Nitrogen = gsub("No Stabilizer", -0.5, Nitrogen)) %>%
  mutate(Nitrogen = gsub("Stabilizer", 0.5, Nitrogen)) %>%
  mutate(mu = mean(Yield)) %>%
  group_by(Nitrogen) %>%
  mutate(T = mean(Yield) - mu) %>%
  ungroup() %>%
  mutate(E = Yield - T - mu) %>%
  mutate(Effect = T + E) %>%
  mutate(Nitrogen = as.numeric(Nitrogen)) %>%
  mutate(E = round(E, 1))

means_plot + geom_point(data = centered_original_data_by_trt, aes(x = Nitrogen, y=Effect))
geom_text(data = centered_original_data_by_trt, aes(x = Nitrogen+0.02, y = Effect, label = Nitrogen))
```

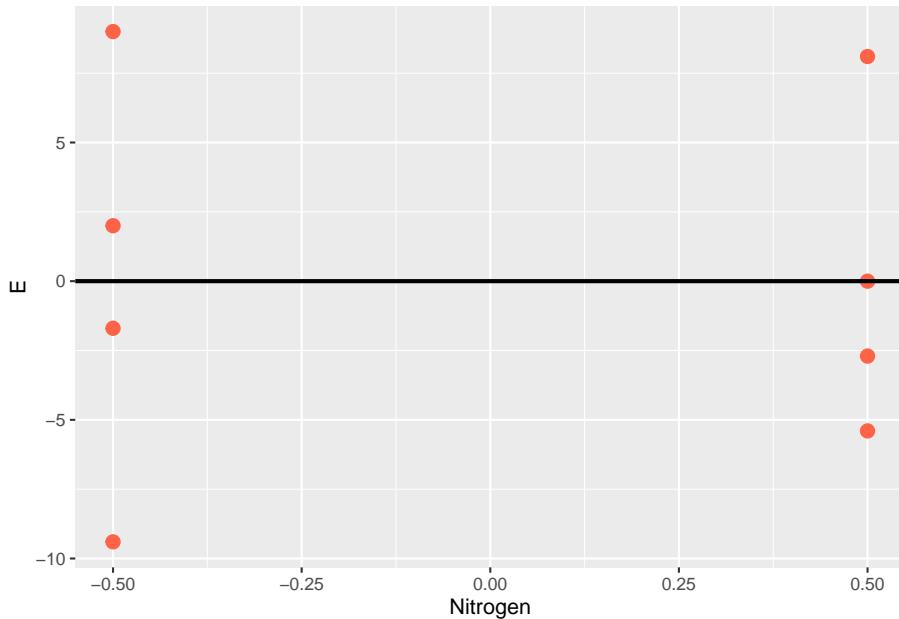
## Warning: Ignoring unknown aesthetics: label



The blue points still represent the treatment mean, and the black line represents the difference between treatments. The red points are the original data – note how they are distributed around each treatment mean. Any individual observation is going to add to or subtract from its treatment mean. The value which each point adds to the treatment mean is shown to the right of the point. This is the error effect for that observation.

Sometimes it is easier to view the experimental unit effect another way, by excluding the treatment effect so that just the effects are plotted around their mean of zero:

```
centered_original_data_by_trt %>%
  ggplot(aes(x = Nitrogen, y=E)) +
  geom_point(size = 3, color="tomato") +
  geom_hline(aes(yintercept = 0), size=1)
```



This kind of plot is often called a *residual plot*, because the error can be thought of as the unexplained, leftover (ie “residue”) effect after the population mean and treatment effects are accounted for. When a correct model is fit to the data, about half the observations for each treatment should be greater than zero, and half below zero. The residual plot is a valuable tool to inspect and verify this assumption.

The yield observed in each plot, then, will be the sum of three values: - the mean yield across all plots - the effect of the treatment applied to that plot - the combined effect of environment, field operations, and measurements

This model can be expressed as:

$$Y_{ij} = \mu + T_i + \epsilon_{ij}$$

Where: -  $Y_{ij}$  is the yield of the  $i^{th}$  treatment level in the  $j^{th}$  block -  $\mu$  is the yield mean across all plots -  $T_i$  is the effect of the  $i^{th}$  level of stabilizer -  $\epsilon_{ij}$  is the *random* effect associated with the plot in the  $j^{th}$  block that received the  $i^{th}$  level of stabilizer

For example, a plot in the 3rd block that received nitrogen treated with stabilizer ( $T_1$ ) would be indicated by the equation:

$$Y_{13} = \mu + T_1 + \epsilon_{13}$$

If the error for this plot,  $\epsilon_{13}$ , was -2, the observed yield would be:

$$Y_{13} = 185 + 10 - 2 = 193$$

Why bother with the linear model when we simply want to know if one treatment yields more than the other? There are two reasons. First, although in agriculture we often think of field trials as testing differences, what we are really doing is using the data from those trials to *predict* future differences. In my opinion, this is one of the key differences between classical statistics and data science. Classical statistics describes what has happened in the past. Data science predicts what will happen in the future.

The linear model above is exactly how we would use data from this trial to predict yields if the product is used under similar conditions. Adding the stabilizer to nitrogen during sidedress will increase the mean yield for a field by 10 bushels. But any given point in that field will have a yield that is also determined by the random effects that our model cannot predict: soil and microclimate, equipment, and measurement errors.

Second, the linear model illustrates what statistics will test for us. Ultimately, every statistical test is a comparison between fixed and random effects: what explains the differences in our observations more: the fixed effects (our treatment) or random effects (error)? In our current example, we can visualize this as follows:

```
centered_original_data = original_data %>%
  gather(Nitrogen, Yield) %>%
  mutate(Nitrogen = gsub("No Stabilizer", -0.5, Nitrogen)) %>%
  mutate(Nitrogen = gsub("Stabilizer", 0.5, Nitrogen)) %>%
  mutate(Yield = Yield - mean(Yield)) %>%
  mutate(Nitrogen = as.numeric(Nitrogen))

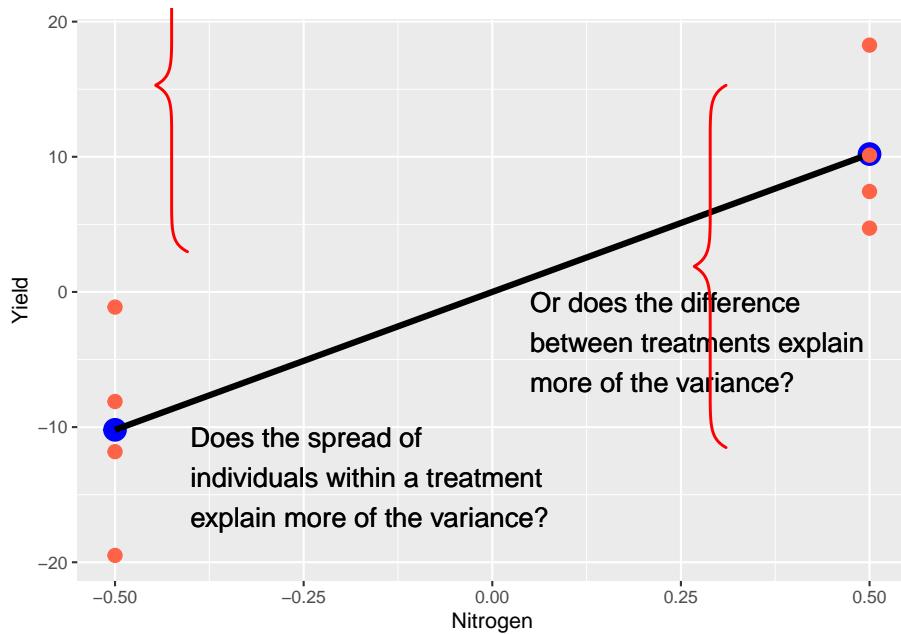
library(ggpubr)

library(pBrackets)
library(grid)

means_plot + geom_point(data = centered_original_data, aes(x = Nitrogen, y=Yield), size=3, color=
  geom_text(aes(x=-0.40, y=-10), label="Does the spread of\nindividuals within a treatment\nexplain\nthe difference?", hjust=0, vjust=1, size=5) +
  geom_text(aes(x=0.05, y=0), label="Or does the difference\nbetween treatments explain\nmore of\nthe difference?", hjust=0, vjust=1, size=5)

grid.locator(unit="native")
```

```
grid.brackets(95, 200, 95, 370, h=0.05, lwd=2, col="red")
grid.brackets(370, 100, 370, 285, h=0.05, lwd=2, col="red")
```



The purpose of a trial is to measure both types of effects and render a verdict. Which hypotheses are important, as we will now see.

### 5.3 Hypotheses

Before we design any experiment, however, we have to define our research question. In the case of a side-by-side trial, the question is generally: “Is one treatments better than the other? This question then needs to be translated into hypotheses.

Outside of statistics, a hypothesis is often described as “an educated guess.” Experiments are designed, however, to test two or more hypotheses. We may casually describe a side-by-side trial as comparing two treatments, but the data are formally used as evidence to test two, opposing hypotheses:

- $H_0$ : The difference between the two treatments is zero.
- $H_a$ : The difference between the two treatments is not zero.

The first hypothesis,  $H_0$ , is called the *null hypothesis*. The second hypothesis,  $H_a$ , is the *alternative hypothesis*. Typically, we tend to focus our effort on

gathering enough evidence to support the alternative hypothesis: after all this work, we typically want to see a treatment difference! But we need to remember the null hypothesis may also be supported.

This process, like the linear model ahead, probably seems overly-formal at first. But like the linear model, it helps us to understand what statistics really tell us. We cannot prove either of these hypotheses. The world is full of one-off exceptions that will prevent either hypothesis from being universal truths. Our science is about comparing the evidence for each hypothesis, and selecting the hypothesis that is probable enough to meet our standards.

To illustrate this, look no further than the t-test we used in the last unit:

$$t = \frac{\bar{x} - \mu}{SED}$$

Recall that  $\bar{x}$  was our treatment difference,  $\mu$  was the hypothesized treatment difference (zero), and  $SED$  was the standard error of the difference. The numerator is our treatment effect on plot yield. The denominator quantifies the random effects on plot yield. As this ratio increases, so does  $t$ . As  $t$  increases, the probability that the true population difference is zero decreases.

Another nuance of hypotheses is this, especially when it comes to the alternative hypothesis. If the evidence fails to support the alternative hypothesis, that does not mean it is wrong. The fixed (treatment) effect we observed was real. But the random effect was so great we could not rule out the differences we observed were the result of chance.

Simply put, our confidence interval was too big to rule out the true difference between treatments was actually zero. There was too much variation among plots. In a trial with a lower standard error of the difference, our  $t$ -value would have been greater, and the probability that the true difference between treatments was zero would be lesser.

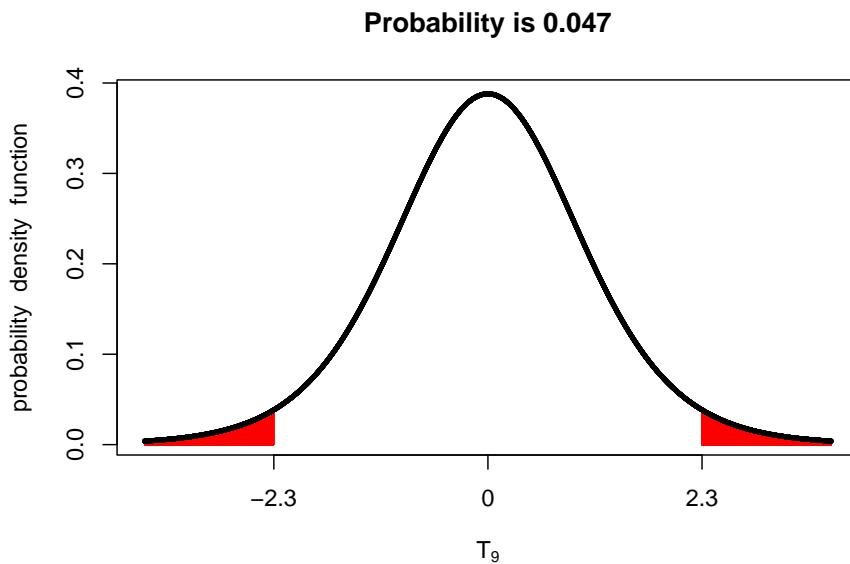
Statistics is not about finding the truth. It is about quantifying the probability an observed difference is the result of chance. Lower probabilities suggest less chance in our observations, and the greater likelihood this difference will be observed if the trial is repeated by someone else, in a laboratory, or in a production field.

## 5.4 P-Value

The P-Value is always a source of confusion in statistics. What does it mean? What is so magical about the 0.05, or 5%, cutoff for declaring populations different? Even if you think you've mastered the P\_value concept already, let's review it one more time.

The P-value, as applied to a distribution, is the probability of observing a value with a given (or greater) difference from the population mean. For example, if we have a t-distribution with a mean of 0 and a standard error of 1, the probability we will, the probability we will observe a value 2.3 standard errors away than the mean, given a population size of 4, is 0.047, or 4.7%.

```
library(fastGraph)
shadeDist(xshade=c(-2.3, 2.3), "dt", parm2 = 9, lower.tail = TRUE)
```



What does a P-value of 0.047 mean? It means the probability of us measuring this value, by dumb luck, when the true population mean is 0, is about 4.7%. Put another way, given the standard error we observed, if the true population mean was zero, we would observe a value equal to or more than 2.3 units away from the mean in less than 5 out of 100 trials.

If this concept sounds the same as that described for a confidence interval, that's because it is the same principle. If we constructed a 95% confidence interval around the sample mean of 2.3, we would see it excludes zero.

Knowing this, we have three options. We can conclude, for now, that the population mean really was zero and we were just very lucky (or unlucky) in our sampling.

We could also repeat the trial multiple times, to see what other values occur. If this is a field trial, that will incur additional research expenses. Even worse, it means delaying a recommendation for several seasons.

Our final option would be to conclude that our population mean is probably *not* zero. If the probability of observing a sample mean 2.3 or more units away from the mean, when the true population mean is zero, is 4.7% or less, then we can also say that the probability that the population mean has a value of zero or less, given our sample mean of 2.3, is 4.7% or less. Given this knowledge, we may conclude the true population mean is different from zero.

This is the power – and beauty! – of statistics. By properly designing an experiment (with sufficient replication and randomization), we can estimate the variability of individuals within a population. We can then use these estimates to test the probability of a hypothetical population mean, given the variability in our sample. And from that, we decide whether one population (which, for example, may have received a new product) is different from the other (which was untreated).

## 5.5 The P-Value and Errors

There are two kinds of P-value: the P-Value we measure, and the maximum P-Value we will accept before determining an observed difference between populations is insignificant. This maximum P-Value is referred to as *alpha* ( $\alpha$ ). You have probably seen statistical summaries that included whether treatments were “different at the  $P \leq 0.05$  level. In this case, the  $\alpha$  is 0.05, or 5%.

Before we discuss why an alpha of 0.05 or 5% is so often used for statistical tests, we need to understand how it relates to the likelihood we will reach the correct inference when comparing populations. You see, once we have gathered our data, calculated the variance in those populations (or, in the case of the paired t-test, the variance in their differences), and run our test(s), we will conclude either that the two populations are the same, or that they are different.

Either conclusion may be right. Or it may be wrong. And since we rarely measure entire populations, we never know their exact population means. We work with probabilities. In the above example, there was a 4.7% chance we could observe a sample mean 2.3 units from a true population of zero. That means there is a 95.3 % (100 - 4.7) chance we would not see that value by chance. But there is still a chance. In other words, there is still a chance we could conclude the population mean is not zero, when in fact it is.

When we infer the mean of one population is significantly different from another (whether the second mean be measured or hypothesized), when in fact the two population means are equal, we commit a *Type I Error*. One example would be concluding the yield of one population, having received additional fertilizer, yielded more than an unfertilized population, when in fact their yields were equal. Another example would be concluding there is a difference in the percent of corn rejected from two populations, each treated with a different insecticide.

The P-value is the probability of making that Type I Error: of observing a sample mean so improbable enough that it leads us to conclude two populations are different, when for all purposes they are the same. If we are worried that recommending a product to a grower that does not increase yield will cost us their business, then we are worried about making a Type I Error. Outside of agriculture, if we are worried about releasing a treatment for COVID-19 that does not work and will leave users unprotected, we are worried about a Type I Error.

If we are really, really worried about wrongly inferring a difference between populations, we might use an even lower P-value. We might use  $P=0.01$ , which means we will not infer two treatments are different unless the mean difference we observe has less than a 1% probability of being observed by chance. This might be the case if the product we recommend to a grower is not \$10 per acre, but \$30. If our COVID treatment is very expensive or has serious side effects, we might insist on an even lower alpha, say  $P=0.001$ , or 0.1%.

So why not always use an alpha of 0.01 or 0.001 to infer whether two populations are different?

There is a second error we can make in the inferences from our research: we can conclude two populations are not different, when in fact they are. In this case, we observed, by chance, a sample mean from one population that was very close to the mean (hypothesized or measured) of another population, when in fact the two population means are different.

For example, a plant breeder might conclude a there performance of a new hybrid is similar to an existing hybrid, and fail to advance it for further testing. An agronomist might erroneously conclude there is no difference in plants treated with one of two micronutrient fertilizers, and fail to recommend the superior one to a grower.

Even more dangerously, a health researcher might conclude there is no difference in the incidence of strokes between a population that receives a new medication and an untreated population, and erroneously conclude a that mdeciation is safe.

Thus, there is a tradeoff betwteen Type I and Type II errors. By reducing the alpha used as critierial to judge whether an one value is significantly different from another, we reduce the likelihood of a Type I error, but increase the likelihood of a Type II error.

In agronomic research, we conventionally use an alpha of 0.05. I cannot explain why we use that particular alpha, other than to suggest it provides an acceptable balance between the risks of Type I and Type II errors for our purposes. It is a value that assures most of the time we will only adopt new practices that very likely to increase biomass or quality, while preventing us wrongly rejecting other practices. In my research, I might use an alpha of 0.05 in comparing hybrids for commercial use. But I might use an alpha of 0.10 or 0.15 if I was doing

more basic work in weed ecology where I was testing a very general hypothesis to explain their behavior.

To make things simple, we will use an alpha of 0.05 for tests in this course, unless stated otherwise.

## 5.6 One-Sided vs Two-Sided Hypotheses

So far, we have treated our hypotheses as:

$H_0$ : there is no difference between two populations, each treated with a different input or practice  
 $H_a$ : there is a difference between two populations, each treated with a different input or practice

We could casually refer to these as “no difference” and “any difference”. But often in side-by-side trials, we have an intuitive sense (or hope) that one population will be “better” than another. If we are the yield of the two populations, one planted with an older hybrid and the other with a newer hybrid, we may be trying to determine whether the new hybrid is likely to yield more. If we are comparing the number of infected plants in populations treated with different fungicides, we may hope the population that receives the new technology will have fewer diseased plants than the population that receives the older technology..

Is this bias? Yes. Is it wrong? I would argue no. The whole purpose of product development, in which I work, is to identify better products. Better hybrids, better micronutrients, better plant growth regulators. If we were equally satisfied whether a new product performed significantly better or significantly worse than an older product – well, in that case, I’d better look into teaching another section of this course.

It is okay to have this kind of bias, so long as we keep our research honest. Proper experimental design, including replication and randomization of plots in the field, or pots in the greenhouse, will go a long way towards that honesty. So will selection of a P-value that acceptably minimizes Type I errors, so that we don’t advance a new product which isn’t highly likely to perform better in future trials, or in the grower’s field.

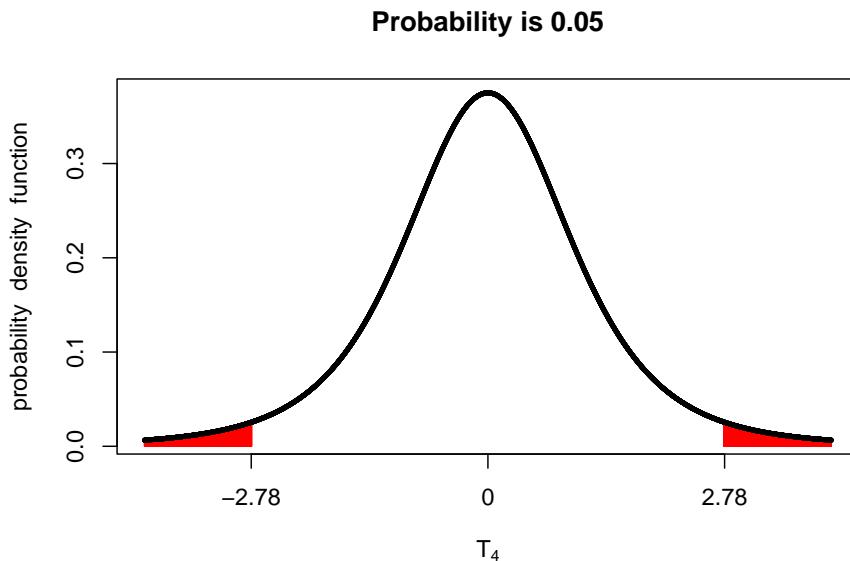
When we have this kind of bias, or interest, however, it also changes our hypotheses. Our null hypothesis is that the population treated with the new product will not perform better than the population treated with the old product. Our alternative hypothesis is the population treated with the new product will perform better.

If we are comparing yield in corn populations treated with a new fungicide, our hypotheses will be:

- Ho: The yield of the population that receives the new fungicide will be equal too – or lesser – than the yield of the population that receives the old fungicide.
- Ha: The yield of the population that receives the new fungicide will be greater than the yield of the population that receives the old fungicide.

The reason these are called one-sided hypotheses is because we are only interested in one-side of the normal distribution. In the two-sided hypotheses we have worked with, we would only care if the yield of the two populations (one receiving the old fungicide, the other receiving the new fungicide) were different. To visualize this

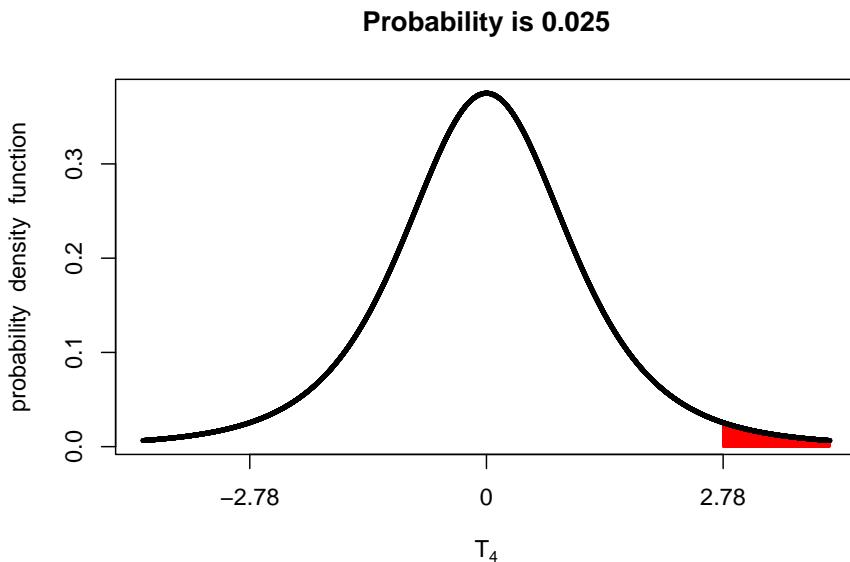
```
library(fastGraph)
alpha_05_2side = qt(0.975, 4)
shadeDist(xshade=c(-alpha_05_2side, alpha_05_2side), "dt", parm2 = 4, lower.tail = TRUE)
```



If either  $t \leq -2.78$  or  $t \geq 2.78$  (either of the red areas above), we declare the two populations different. In other words, the observed t-value can occur in either of the two tails and be significant. Accordingly, we refer to this as a two-tailed test.

In testing a one-sided hypothesis, we only care if the difference between the population that received the new fungicide and the population that received the old fungicide (ie new fungicide - old fungicide) has a t-value of 1.98 or greater. We would visualize this as:

```
library(fastGraph)
alpha_05_1side = qt(0.975, 4)
shadeDist(xshade=alpha_05_1side, "dt", parm2 = 4, lower.tail = FALSE)
```

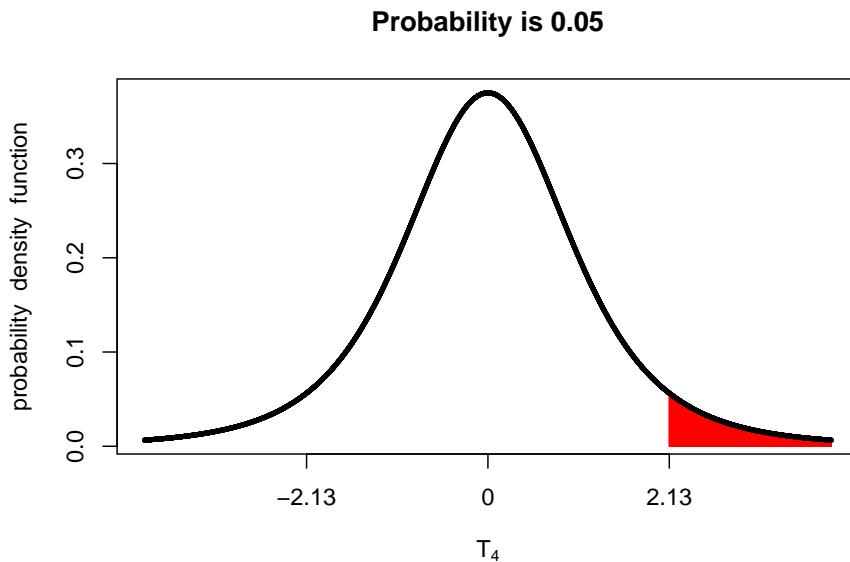


Only if the measured t-value falls in the upper tail will the population that receives the new fungicide be considered significantly better than the population that received the old fungicide. We therefore – you guessed it! – refer to this test as a one-tailed test.

In using a one-tailed test, however, we need to use a different t-value to achieve our alpha (maximum p-value for significance). If you look at the plot, only 2.5% of the distribution is in the upper tail. If we leave this as it is, we will only conclude the populations are different if their P-value is equal to or less than 2.5%. Reducing our P-value from 5% to 2.5% will, indeed, reduce our probability of Type I errors. But it will increase our likelihood of Type II errors.

If we are going to conduct a one-tailed test with an alpha of 0.05, we need to adjust the percentage of the distribution in the upper tail to 5% of the distribution:

```
alpha_05_1side = qt(0.95, 4)
shadeDist(xshade=alpha_05_1side, "dt", parm2 = 4, lower.tail = FALSE)
```



The implication is that the minimum difference between populations to be significant at an  $\alpha=0.05$  is lesser than for a two-tailed test.

A common first response to the one-tailed test is: “Isn’t that cheating? Aren’t we just switching to a one-tailed test so we can nudge our difference passed the goal line for significance”? And, indeed, if you switch to a one-tailed test for that reason alone, it is cheating. That is why it is important we declare our hypotheses before we begin our trial. If we are going to run a one-tailed test, it needs to be based on a one-sided hypothesis that declares, from the beginning, we are only testing the difference in one direction, either because we have intuition that the difference will be one-sided, or we have a practical reason for only being interested in a positive or negative difference between populations.

## 5.7 Exercise: Linear Additive Model

The linear additive model may seem a very esoteric concept in data science, but it is critical to understand how we make predictions from statistical models, and how we test whether those models significantly explain the variance in our observations. It also underscores an important concept in statistical modelling: the assumption that the different effects that combine to explain an observed value are *additive*.

### 5.7.1 Case Study: Barley Effects

The barley dataset is the same we used in the other exercises, except I have calculated the population mean, mu, treatment effect, and error effect. Ignore the last four columns – those were calculated only to create the chart you will run below.

```
barley_effects = read.csv("data-unit-5/exercise_data/barley_effects_table.csv")
barley_effects
```

##	block	trt	y	plot	mu	trt_mean	trt_effect	error_effect	trt_bar_min	
## 1		1	new	291.1	1	312.26	339.06	26.8	-47.96	312.26
## 2		1	old	223.9	2	312.26	285.46	-26.8	-61.56	-26.80
## 3		2	new	321.4	3	312.26	339.06	26.8	-17.66	312.26
## 4		2	old	249.9	4	312.26	285.46	-26.8	-35.56	-26.80
## 5		3	new	399.3	5	312.26	339.06	26.8	60.24	312.26
## 6		3	old	330.8	6	312.26	285.46	-26.8	45.34	-26.80
## 7		4	new	362.6	7	312.26	339.06	26.8	23.54	312.26
## 8		4	old	349.8	8	312.26	285.46	-26.8	64.34	-26.80
## 9		5	new	320.9	9	312.26	339.06	26.8	-18.16	312.26
## 10		5	old	272.9	10	312.26	285.46	-26.8	-12.56	-26.80
##						trt_bar_max	err_bar_min	err_bar_max		
## 1						339.06	-47.96	0.0		
## 2						0.00	-88.36	-26.8		
## 3						339.06	-17.66	0.0		
## 4						0.00	-62.36	-26.8		
## 5						339.06	339.06	399.3		
## 6						0.00	312.26	357.6		
## 7						339.06	339.06	362.6		
## 8						0.00	312.26	376.6		
## 9						339.06	-18.16	0.0		
## 10						0.00	-39.36	-26.8		

### 5.7.2 Plotting the Effects

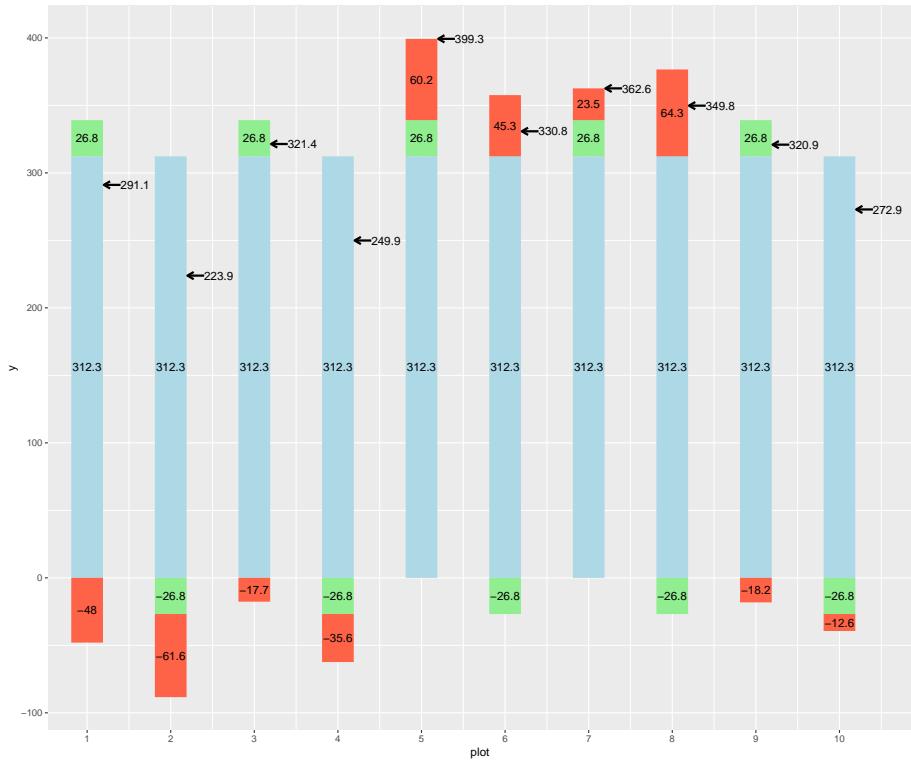
Below is the code to create a bar plot. This kind of bar plot is *stacked* – it shows multiple measures for each observation (plot), stacked on top each other. Normally, a stacked bar plot can be created in a couple of lines of code – the intricate code below was so I could customize the order of stacking for each plot. Run the code and observe the plot.

```
ggplot(barley_effects) +
  geom_segment(aes(x=plot, y=0, xend=plot, yend=mu), color="lightblue", size=14) +
  geom_segment(aes(x=plot, xend=plot, yend=trt_bar_min, yend=trt_bar_max), color="lightgreen", size=14)
```

```

geom_segment(aes(x=plot, xend=plot, y=err_bar_min, yend=err_bar_max), color="tomato")
# geom_point(aes(x=plot, y=y), size=4) +
geom_text(aes(x=plot, y=mu/2, label= round(mu,1))) +
geom_text(aes(x=plot, y=trt_bar_min+(trt_bar_max-trt_bar_min)/2, label=round(trt_eff,
geom_text(aes(x=plot, y=err_bar_min+(err_bar_max-err_bar_min)/2, label=round(error_eff,
geom_segment(aes(x=plot+0.4, xend=plot+0.2, y=y, yend=y), arrow = arrow(length = unit(10, "pt"),
geom_text(aes(x=plot+0.4, y=y, label=round(y,1)), hjust=0) +
scale_x_continuous(breaks=c(1:10))

```



If this plot is too big to fit comfortably on this screen, remember you can knit this code using the menu immediately above this window, and either view it in HTML or print it out to use.

What I have tried to do with this plot is to visualize the additive effects of the population mean (light blue bar), treatment effect (green bar), and error effect (red bar). The sum of their effects (the observed value) is shown by a black arrow to the right of each bar.

The blue bars are the same, 312.3, reflecting that the population mean for the whole trial does not change: every plot starts out with this same value, with the effects then adding or subtracting from its observed value.

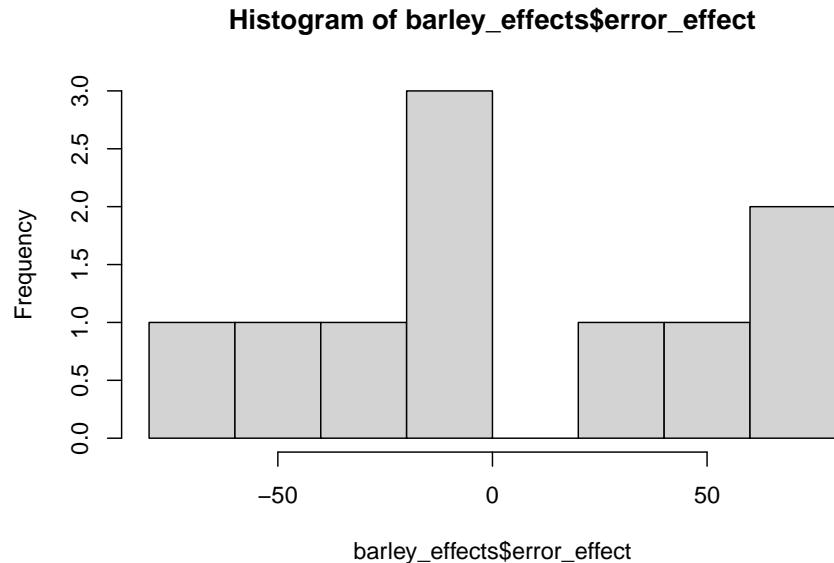
Looking at this plot, you can see half of the green bars have a treatment effect of 26.8, while the other half have a treatment effect of -26.8. In a two treatment trial, the treatment effects will be exactly opposite for the two levels of treatment. If we look at the plot above, we can see the positive treatment effect is associated with the new genotype, and the negative effect with the old genotype.

The red bars represent the error. This is the unexplained variance in observed value. Part of it reflects how, try as hard as we might, not every plot has exactly the same environment. Soil types and microclimates differ. Plot lengths may differ slightly among plots. One plot may be planted more evenly than others. A fertilizer applicator might skip. An ear might bounce out of the combine. Note that the errors are negative in some cases and positive in others. In addition, they are not consistent – their values are random.

The treatment effect above is an example of a *fixed effect* – it has a specific, consistent effect, based on the level of treatment.

The error effect above is an example of a *random effect*. Its value varies from plot to plot. Its variation usually follows a normal distribution with a mean close to zero. Lets run the histogram below and check.

```
hist(barley_effects$error_effect)
```



```
mean(barley_effects$error_effect)
```

```
## [1] 1.775489e-16
```

### 5.7.3 Examining Individual Plots

We can see

Recall our linear model is:  $Y_{ij} = \mu + T_i + e(i)j$

$Y_{ij}$  is the observed value for plot  $j$  that has received treatment  $i$ .  $\mu$  is the population mean.  $T_i$  is the treatment effect associated with level  $i$ .  $e(i)j$  is the error term. So let's plug a few plots into this equation and see how it works.

In plot 2:  $- Y_{ij} = 223.9 - \mu = 312.3$ , -  $T_i$ , associated with the old hybrid =  $-26.8 - e(i)j$ , associated with plot 2, =  $-61.6$

Our model for plot 2, then is:  $223.9 = 312.3 - 26.8 - 61.6$

What about for plot 5?

$Y_{ij} = 399.3$   $\mu = 312.3 - T_i$ , associated with the new hybrid =  $+26.8 - e(i)j$ , associated with plot 5 =  $+60.2$

Our model for plot 5, then is:  $399.3 = 312.3 + 26.8 + 60.2$

What about for plot 8?

$Y_{ij} = 349.8$   $\mu = 312.3 - T_i$ , associated with the old hybrid =  $-26.8 - e(i)j$ , associated with plot 8 =  $+64.3$

Our model for plot 8, then is:  $349.8 = 312.3 - 26.8 + 64.3$

What is interesting in this last example is that the positive error “masks” the negative effect of the treatment – the yield with the old hybrid, in this case, is greater than plots 3 and 9, which received the old hybrid! This underscores why it is important to break down the effects behind an observation to better understand the true contribution of each treatment level.

### 5.7.4 Practice: Groudnut

Let's load and plot the groundnut data from the other exercises this unit. Like the barley data above, we have added several columns in order to calculate treatment and error effects and to draw the plot below.

```
groundnut_effects = read.csv("data-unit-5/exercise_data/groundnut_effects_table.csv")
groundnut_effects

##   block row col trt    y dry plot      mu trt_mean trt_effect error_effect
## 1     B1  4   2   A 5.2 3.3     1 3.225     4.30     1.075     0.90
## 2     B1  4   6   C 2.4 1.4     2 3.225     2.15    -1.075     0.25
## 3     B2  3   1   C 1.7 0.9     3 3.225     2.15    -1.075    -0.45
## 4     B2  3   6   A 4.8 3.0     4 3.225     4.30     1.075     0.50
```

```

## 5   B3   2   3   A 2.4 1.4   5 3.225   4.30   1.075   -1.90
## 6   B3   2   6   C 2.5 1.5   6 3.225   2.15   -1.075   0.35
## 7   B4   1   3   C 2.0 1.0   7 3.225   2.15   -1.075   -0.15
## 8   B4   1   5   A 4.8 3.1   8 3.225   4.30   1.075   0.50
##   trt_bar_min trt_bar_max err_bar_min err_bar_max
## 1      3.225        4.3     4.300     5.200
## 2     -1.075        0.0     3.225     3.475
## 3     -1.075        0.0    -1.525    -1.075
## 4      3.225        4.3     4.300     4.800
## 5      3.225        4.3    -1.900     0.000
## 6     -1.075        0.0     3.225     3.575
## 7     -1.075        0.0    -1.225    -1.075
## 8      3.225        4.3     4.300     4.800

```

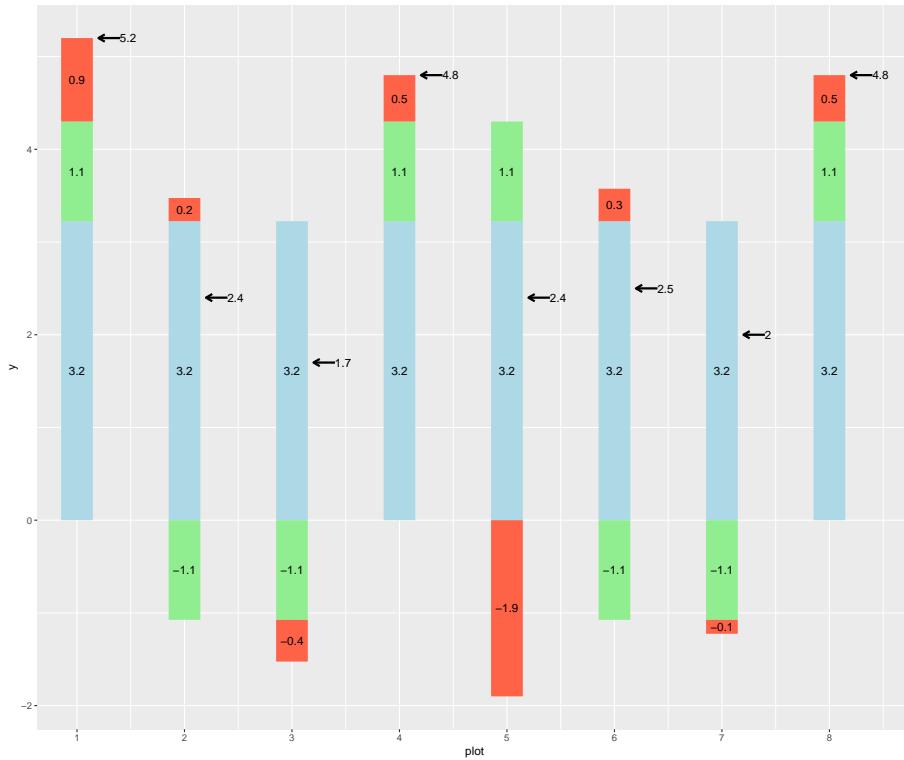
### 5.7.5 Plotting the Effects

Below is the code to create a bar plot. This kind of bar plot is *stacked* – it shows multiple measures for each observation (plot), stacked on top each other. Normally, a stacked bar plot can be created in a couple of lines of code – the intricate code below was so I could customize the order of stacking for each plot. Run the code and observe the plot.

```

ggplot(groundnut_effects) +
  geom_segment(aes(x=plot, y=0, xend=plot, yend=mu), color="lightblue", size=14) +
  geom_segment(aes(x=plot, xend=plot, y=trt_bar_min, yend=trt_bar_max), color="lightgreen", size=14) +
  geom_segment(aes(x=plot, xend=plot, y=err_bar_min, yend=err_bar_max), color="tomato", size=14) +
  # geom_point(aes(x=plot, y=y), size=4) +
  geom_text(aes(x=plot, y=mu/2, label= round(mu,1))) +
  geom_text(aes(x=plot, y=trt_bar_min+(trt_bar_max-trt_bar_min)/2, label=round(trt_effect,1))) +
  geom_text(aes(x=plot, y=err_bar_min+(err_bar_max-err_bar_min)/2, label=round(error_effect,1))) +
  geom_segment(aes(x=plot+0.4, xend=plot+0.2, y=y, yend=y), arrow = arrow(length = unit(0.01, "np"))
  geom_text(aes(x=plot+0.4, y=y, label=round(y,1)), hjust=0) +
  scale_x_continuous(breaks=c(1:10))

```



Model plots 1, 5, and 6, using the linear model as we did above.

## 5.8 Exercise: One-Sided Hypotheses

We learned in the lesson there are times when it is appropriate to use a one-sided hypothesis. A one-sided hypothesis specifies how two treatments will rank in a trial, for example that variety B will have greater yield than variety A:

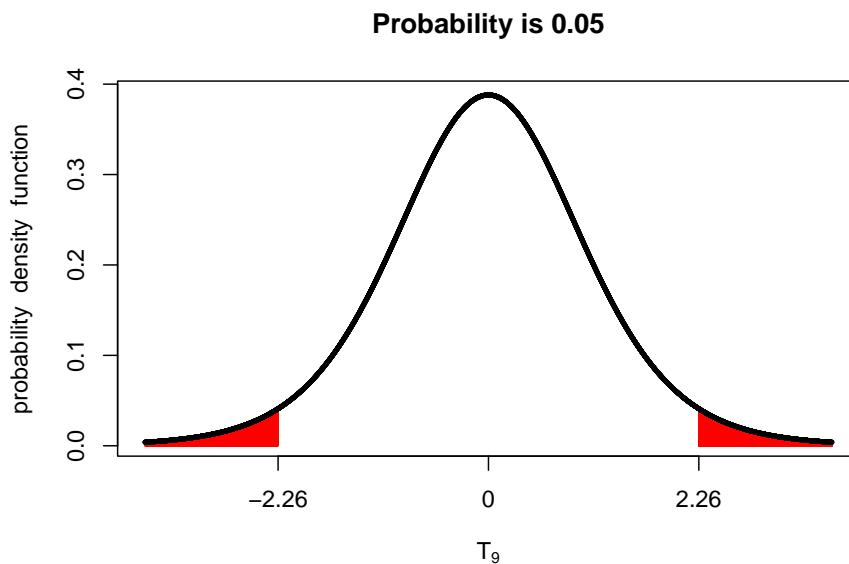
$$H_0: A \geq B \quad H_a: A < B$$

A two-sided hypothesis, in contrast, only specifies that variety A and variety B will be different:  $H_0: A = B$   $H_a: A \neq B$

As we learned in the lecture, the one sided t-test requires a lesser difference for significance than the two-sided test. Given 9 degrees of freedom, and a standard error of the difference of 1, for example, a difference equal to or greater than 2.26 – or equal to or less than -2.26 – between treatments would need to be observed between treatments for the two-sided test to be significant.

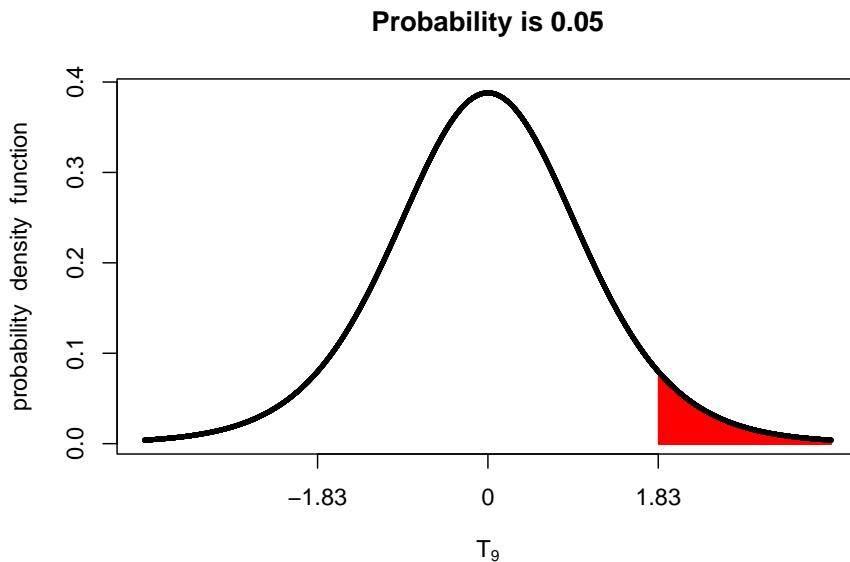
```
library(tidyverse)
library(fastGraph)
```

```
alpha_05_2side = qt(0.975, 9)
shadeDist(c(-alpha_05_2side, alpha_05_2side), "dt", parm2 = 9, lower.tail = TRUE)
```



In a one-sided test, a lower difference, between treatments, 1.83, is required for significance at the  $p < 0.05$  level.

```
alpha_05_1side = qt(0.95, 9)
shadeDist(xshade=alpha_05_1side, "dt", parm2 = 9, lower.tail = FALSE)
```



### 5.8.1 Case Study: Groundnut

In this study, the wet weight of groundnut, in kg/plot, was measured for two genotypes, coded A and C. The plots were paired.

```
groundnut = read.csv("data-unit-5/exercise_data/groundnut.csv")
head(groundnut)

##   block row col gen wet dry
## 1     B1   4   2   A 5.2 3.3
## 2     B1   4   6   C 2.4 1.4
## 3     B2   3   1   C 1.7 0.9
## 4     B2   3   6   A 4.8 3.0
## 5     B3   2   3   A 2.4 1.4
## 6     B3   2   6   C 2.5 1.5

groundnut %>%
  group_by(gen) %>%
  summarise(wet = mean(wet))

## # A tibble: 2 x 2
##   gen      wet
##   <fct>  <dbl>
## 1 A       4.00
## 2 C       2.00
```

```
##   <chr> <dbl>
## 1 A      4.3
## 2 C      2.15
```

### 5.8.2 One-Sided T-Test

In the last unit, we learned to use the `t.test()` function to conduct a paired two-sided t-test. Let's first analyze the groundnut data that way.

Our hypotheses are

```
t.test(wet ~ gen, groundnut, paired=TRUE)

##
## Paired t-test
##
## data: wet by gen
## t = 2.854, df = 3, p-value = 0.0649
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
## -0.2474194 4.5474194
## sample estimates:
## mean of the differences
##                      2.15
```

We see we have a p-value of 0.0649. The two genotypes do not produce different wet weights of groundnuts at the  $p < 0.05$  level of significance.

Now let's run the one-sided test. To specify our hypothesis properly, we need to know which treatment will be the *subtractant*: the number that is subtracted. This is really important. In R, the treatment which comes *second* in alphabetical order is subtracted from the treatment that comes *first*.

Let's say our hypotheses are these:  $H_0: A \geq C$   $H_a: A < C$

To tell R to run the `t.test` this way, we add the `alternative = "less"` argument to our `t-test`. If A is greater than C, we will have a positive difference, so we specify `alternative = "greater"`.

```
t.test(wet ~ gen, groundnut, paired=TRUE, alternative = "greater")
```

```
##
## Paired t-test
##
## data: wet by gen
```

```
## t = 2.854, df = 3, p-value = 0.03245
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
## 0.3771502      Inf
## sample estimates:
## mean of the differences
##                           2.15
```

We can now see the p-value for the test is 0.03 – genotype C produces a greater wet weight of groundnut than genotype A at the  $p < 0.05$  level of significance.

### 5.8.3 Practice: Barley

In this study, yield of a new and old genotype were compared. Treatments were paired.

```
barley = read.csv("data-unit-5/exercise_data/barley.csv")
```

- 1) Run a two-sided t.test to compare the yield of the two genotypes. You should get a p-value of 2.158e-06.
- 2) Run a one-sided t.test to test the hypothesis the new hybrid yields greater than the old hybrid. Going by alphabetical order, R will subtract the mean of “old” from “new”. Given our hypothesis that the yield of the “new” genotype will be greater than that of the “old”, our difference will again be positive. Again, use the “alternative=”greater”\* argument with the *t.test()* function. Your answer should have a p-value = 1.079e-06.

### 5.8.4 Practice: Strawberry

The yield of two strawberry genotypes was tested in a paired treatment design.

```
strawberry = read.csv("data-unit-5/exercise_data/strawberry.csv")
```

- 1) Test the difference between genotypes using a two-sided test. You should get p-value = 0.055.
- 2) Test the hypothesis that genotype F is greater than genotype R1. Since R will subtract R1 from F, our difference will be positive. You should get p-value = 0.0275.

## 5.9 Exercise: Type I and Type II Errors

Please use the following link to test whether you understand the difference between Type I and Type II errors.



## Chapter 6

# Multiple Treatment Trials

Here is our course, so far, in a nutshell:

- statistics is all about populations
- when we compare treatments, we are actually comparing populations that have been exposed to different products or management
- when we can measure every individual in that population, we can use the Z-distribution to describe their true population means and variance
- most of the time in agricultural research, however, we must use samples (subsets from those populations) to *estimate* their population means and variance
- when we use the sample mean to estimate the population mean, we use the t-distribution to describe the distribution of sample means around the mean of their values
- the distribution of sample means can be used to calculate the probability (the p-value) the true population mean is a hypothetical value
- in a paired t-test of two populations, we create a new population of differences, and calculate the probability its mean is zero
- proper hypotheses and alphas (maximum p-values for significance) can reduce the likelihood we conclude populations are different when they are likely the same, or the same when they are likely different

I include this brutal distillation so you can see how the course has evolved from working with complete populations to samples, from “true” or “certain” estimates of population means to estimates, from working with single populations to comparing differences between two populations.

In the last two units, we learned how to design and evaluate the results of side-by-side trials: trials in fields or parts of fields were divided into two populations that were managed with different treatments or practices. This was a practical, powerful, jumping-off point for thinking about experiments and their analyses.

Let's face it, however: if we only compared two treatments per experiment in product testing or management trials, our knowledge would advance at a much slower pace. So in the next three units, we will learn how to design and evaluate trials to test multiple *categorical* treatments. By *categorical*, we mean treatments we identify by name, not quantity. Treatments that, in general, cannot be ranked. Hybrids are a perfect example of categorical treatments. Herbicides, fungicides, or fertilizers that differ in brand name or chemical composition are also categorical treatments. Comparisons of cultural practices, such as tillage systems or crop rotations are categorical treatments as well.

## 6.1 Case Study

For our case study, we will look at a comparison of four hybrids from the Marin Harbur seed company. We will compare hybrids MH-052672, MH-050877, MH-091678, and MH-032990, which are not coded by relative maturity or parent lines, but represent some of the owner's favorite Grateful Dead concerts:

- MH-052672 has a great early disease package and is ideal for environments that have heavy morning dews - MH-050877 is very resilient in poor fertility soils and among the last to have its leaves fire under low nitrogen conditions. It can also grow well on everything from flatlands to mountains.
- MH-091678 has a very strong root base that will not be shaken down by late-season wind storms - MH-032990 offers a very rich performance and responds well to additional inputs. (This one catches growers' eyes at every field day – not to blow our own horn.)

```
library(tidyverse)

## -- Attaching packages ----- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr    1.0.8
## v tidyr   1.2.0     v stringr  1.4.0
## v readr   2.1.2     vforcats 0.5.1

## Warning: package 'ggplot2' was built under R version 4.1.3

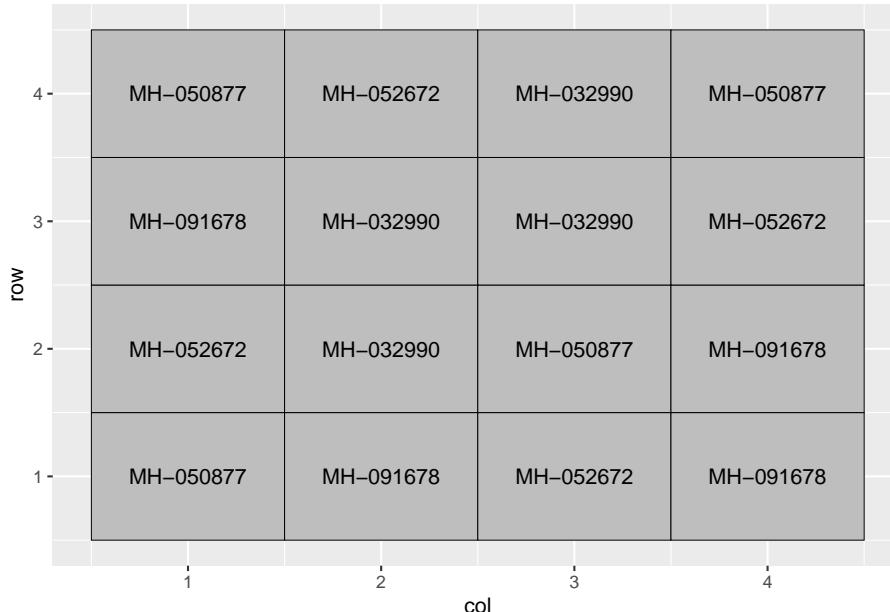
## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()   masks stats::lag()

hybrid_data = read.csv("data-unit-6/grateful_data.csv")
head(hybrid_data)
```

```
##   plot_number   Hybrid Yield col row
## 1           1 MH-050877 189.5   1   1
## 2           2 MH-052672 186.4   1   2
## 3           3 MH-091678 196.9   1   3
## 4           4 MH-050877 191.2   1   4
## 5           5 MH-091678 191.8   2   1
## 6           6 MH-032990 198.9   2   2
```

The hybrids were grown in a field trial with four replications as shown below.

```
hybrid_data %>%
  ggplot(aes(x=col, y=row, label=Hybrid)) +
  geom_tile(fill="grey", color="black") +
  geom_text(aes(label = Hybrid))
```



The arrangement of treatments above is a *completely randomized design*. This means the hybrids were assigned at random among all plots – there was no grouping or pairing of treatments as in our side-by-side trials earlier.

This design – to be honest – is used more often in greenhouse or growth chamber research where the soil or growing media are more uniform. Still, it is the most appropriate design to start with in discussing multiple treatment trials.

## 6.2 The Linear Additive Model

In order to understand how we will analyze this trial, let's plot out our data.

```

hybrid_rank = hybrid_data %>%
  group_by(Hybrid) %>%
  summarise(Yield = mean(Yield)) %>%
  ungroup() %>%
  arrange(Yield) %>%
  mutate(rank = row_number()) %>%
  select(-Yield)

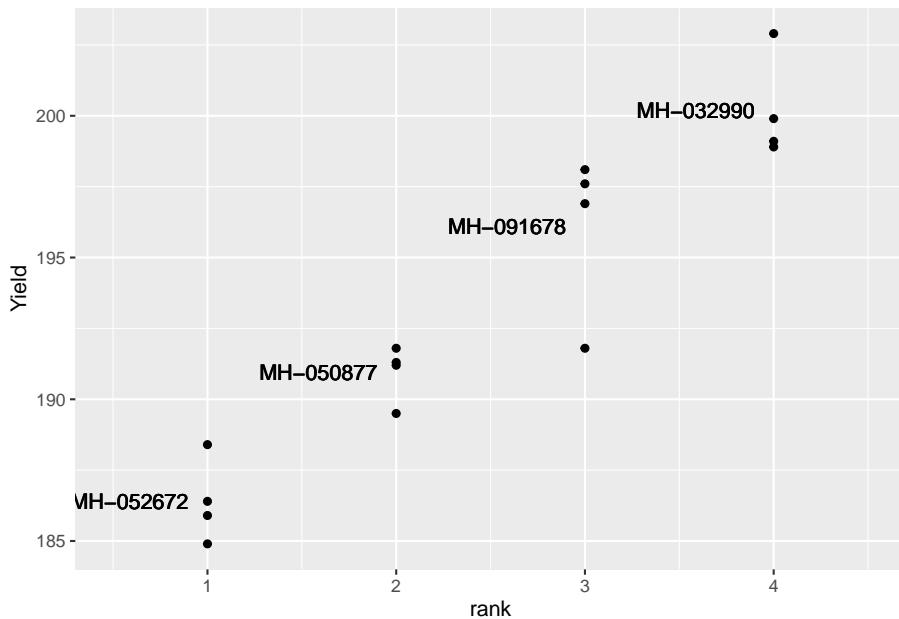
effects_data = hybrid_data %>%
  select(-col, -row) %>%
  mutate(mu = mean(Yield)) %>%
  arrange(Hybrid) %>%
  group_by(Hybrid) %>%
  mutate(T = mean(Yield) - mu) %>%
  ungroup() %>%
  mutate(E = Yield - T - mu) %>%
  left_join(hybrid_rank)

## Joining, by = "Hybrid"

hybrids = unique(effects_data$Hybrid)

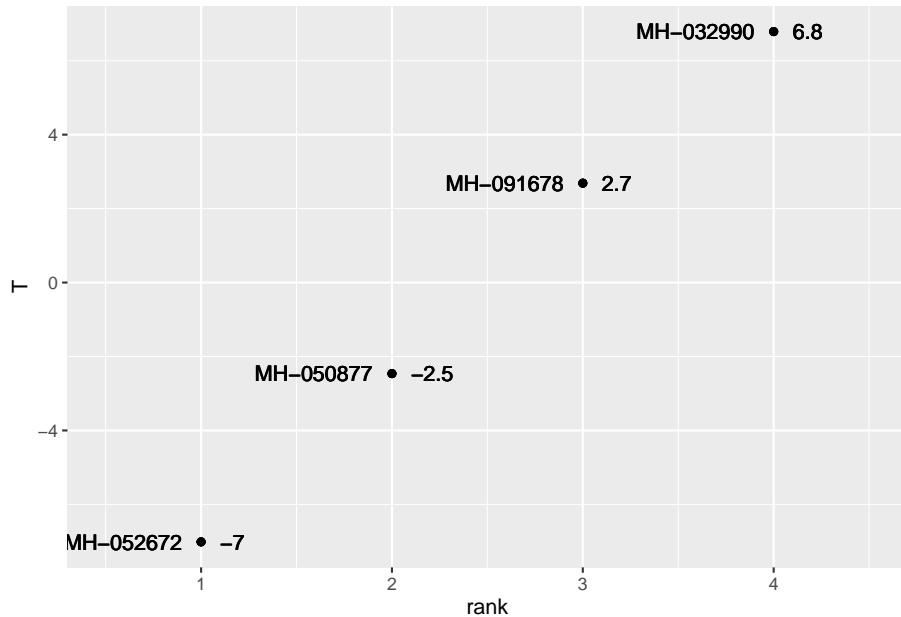
effects_data %>%
  ggplot(aes(x=rank, y=Yield)) +
  geom_point() +
  geom_text(aes(x=rank-0.1, y=mu+T, label=Hybrid, hjust=1)) +
  lims(x=c(0.5, 4.5))

```



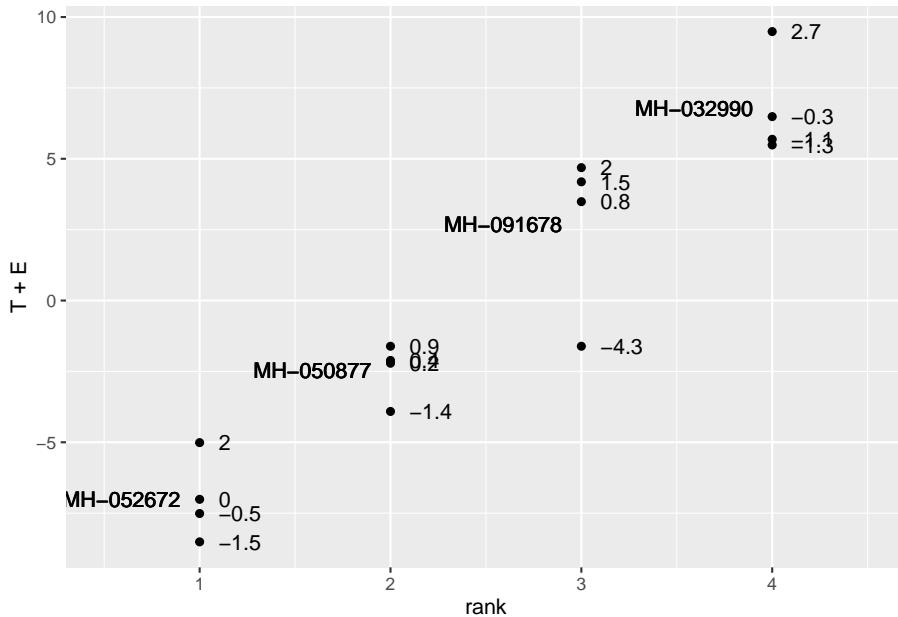
Our treatment means are shown below:

```
effects_data %>%
  ggplot(aes(x=rank, y=T)) +
  geom_point() +
  geom_text(aes(x=rank-0.1, y=T, label=Hybrid, hjust=1)) +
  geom_text(aes(x=rank+0.1, y=T, label=round(T, 1)), hjust=0) +
  lims(x=c(0.5, 4.5))
```



We can see the plot effects as well.

```
effects_data %>%
  ggplot(aes(x=rank, y=T+E)) +
  geom_point() +
  geom_text(aes(x=rank-0.1, y=T, label=Hybrid, hjust=1)) +
  geom_text(aes(x=rank+0.1, y=T+E, label=round(E, 1)), hjust=0, size=4) +
  lims(x=c(0.5, 4.5))
```



We can observe the linear model using this table:

```
effects_data %>%
  select(Hybrid, mu, T, E, Yield) %>%
  mutate(mu = round(mu, 1),
         T = round(T, 1),
         E = round(E, 1)) %>%
  rename(`Yield (mu + T + E)` = Yield)

## # A tibble: 16 x 5
##   Hybrid      mu      T      E `Yield (mu + T + E)`
##   <chr>     <dbl>   <dbl>   <dbl>      <dbl>
## 1 MH-032990 193.    6.8   -1.3     199.
## 2 MH-032990 193.    6.8   -1.1     199.
## 3 MH-032990 193.    6.8    2.7     203.
## 4 MH-032990 193.    6.8   -0.3     200.
## 5 MH-050877 193.   -2.5   -1.4     190.
## 6 MH-050877 193.   -2.5    0.2     191.
## 7 MH-050877 193.   -2.5    0.9     192.
## 8 MH-050877 193.   -2.5    0.4     191.
## 9 MH-052672 193.    -7     0       186.
## 10 MH-052672 193.    -7   -1.5     185.
## 11 MH-052672 193.    -7     2       188.
## 12 MH-052672 193.    -7   -0.5     186.
## 13 MH-091678 193.    2.7    0.8     197.
```

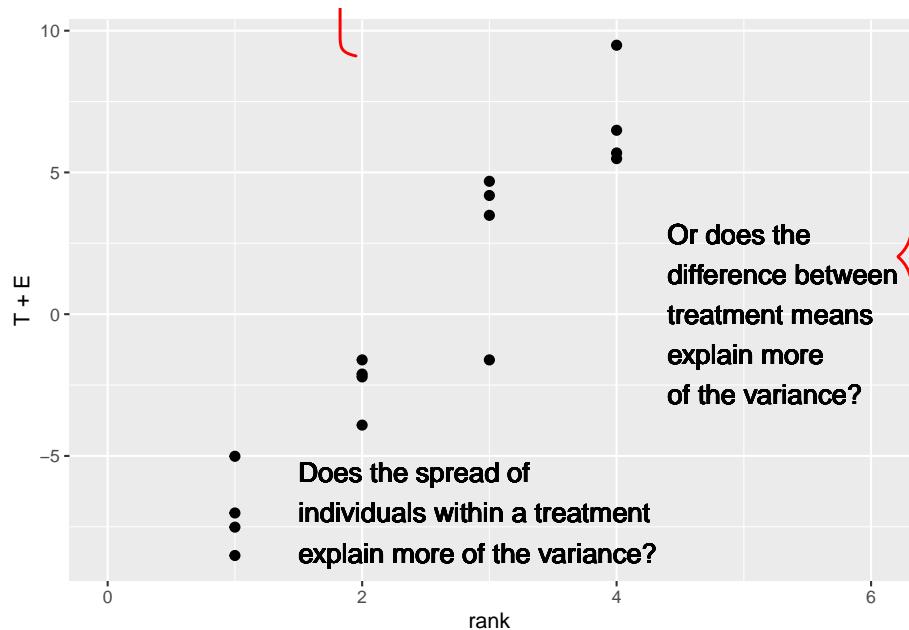
```
## 14 MH-091678 193.  2.7 -4.3          192.
## 15 MH-091678 193.  2.7   2          198.
## 16 MH-091678 193.  2.7   1.5         198.
```

Just as in the side-by-side trials, our statistical test is based on the ratio of the variation among treatment means to the variation among observations within each treatment.

```
library(grid)
library(pBrackets)

p = effects_data %>%
  ggplot(aes(x=rank, y=T+E)) +
  geom_point(size = 2) +
  lims(x=c(0, 6))
p + geom_text(aes(x=1.5, y=-7), label="Does the spread of\nindividuals within a treatment\nexplain more of the variance?",  
             hjust=0, vjust=0.5, size=5) +
  geom_text(aes(x=4.4, y=0), label="Or does the\ndifference between\ntreatment means\nexplain more of the variance?",  
             hjust=0, vjust=0.5, size=5)

grid.locator(unit="native")
#
grid.brackets(180, 300, 180, 375, h=0.05, lwd=2, col="red")
grid.brackets(473, 60, 473, 335, h=0.05, lwd=2, col="red")
```



### 6.3 Analysis of Variance

Chances are if you have spent time around agronomic research you have probably heard of *ANOVA*.



Figure 6.1: A Nova

No, not that fine example of Detroit muscle, but a statistical test, the *Analysis of Variance (ANOVA)*. The ANOVA test performs the comparison described above when there are more than two more populations that differ categorically in their management. I'll admit the nature of this test was a mystery to me for years, if not decades. As the name suggests, however, it is simply an analysis (a comparison, in fact) of the different sources of variation as outlined in our linear additive model. An Analysis of Variance tests two hypotheses:

- Ho: There is no difference among population means.
- Ha: There is a difference among population means.

In our hybrid trial, we can be more specific:

- Ho: There is no difference in yield among populations planted with four different hybrids.
- Ha: There is a difference in yield among populations planted with four different hybrids.

But the nature of the hypotheses stays the same.

## 6.4 The F statistic

The Analysis of Variance compares the variance from the treatment effect to the variance from the error effect. It does this by dividing the variance from treatments by the variance from Error:

$$F = \frac{\sigma^2_{treatment}}{\sigma^2_{error}}$$

The *F-value* quantifies the ratio of treatment variance to error variance. As the ratio of the variance from the treatment effect to the variance from the error effect increases, so does the F-statistic. In other words, a greater F-statistic suggests a greater treatment effect – or – a smaller error effect.

## 6.5 The ANOVA Table

At this point, it is easier to explain the

```
library(broom)

## Warning: package 'broom' was built under R version 4.1.3

hybrid_test = tidy(aov(Yield ~ Hybrid, hybrid_data))

hybrid_test

## # A tibble: 2 x 6
##   term        df  sumsq meansq statistic    p.value
##   <chr>     <dbl> <dbl>  <dbl>     <dbl>      <dbl>
## 1 Hybrid      3  434.   145.     38.4  0.00000197
## 2 Residuals   12  45.2    3.76     NA     NA

summary(hybrid_test)

##          term            df        sumsq       meansq
##  Length:2        Min.   : 3.00   Min.   :45.17   Min.   : 3.764
##  Class :character 1st Qu.: 5.25   1st Qu.:142.41   1st Qu.: 39.000
##  Mode  :character Median : 7.50   Median :239.65   Median : 74.237
## 
##               Mean   : 7.50   Mean   :239.65   Mean   : 74.237
##               3rd Qu.: 9.75   3rd Qu.:336.89   3rd Qu.:109.473
##               Max.  :12.00   Max.  :434.13   Max.  :144.709
```

```
##      statistic      p.value
##  Min.   :38.44   Min.   :2e-06
##  1st Qu.:38.44  1st Qu.:2e-06
##  Median :38.44  Median :2e-06
##  Mean   :38.44  Mean   :2e-06
##  3rd Qu.:38.44  3rd Qu.:2e-06
##  Max.   :38.44  Max.   :2e-06
##  NA's    :1       NA's    :1
```

As we did with the t-test a couple of units ago, lets go through the ANOVA output column by column, row by row.

### 6.5.1 Source of Variation

The furthest column to the left, *term* specifies the two effects in our linear additive model: the Hybrid and Residual effects. As mentioned in the last chapter, the term Residual is often used to describe the “leftover” variation among observations that a model cannot explain. In this case, it refers to the variation remaining when the Hybrid effect is accounted for. This column is also often referred to as the *Source of Variation* column.

### 6.5.2 Sum of Squares

Let’s skip the df column for a moment to expain the column titled *sumsq* in this output. This column lists the sums of squares associated with the Hybrid and Residual Effect. Remember, the sum of squares is the sum of the squared differences between the individuals and the population mean. Also, we need to calculate the sum of squares before calculating variance

The Hybrid sum of squares based on the the difference between the treatment mean and the population mean for each observation in the experiment. In the table below we have mu, the population mean, and T, the effect or difference between the treatment mean and mu. We square T for each of the 16 observations to create a new column, T-square

```
trt_ss = effects_data %>%
  select(Hybrid, mu, T) %>%
  mutate(`T-square` = T^2)

# %>%
#   mutate(mu = round(mu, 1),
#         T = round(T, 1))
trt_ss
```

```
## # A tibble: 16 x 4
##   Hybrid     mu     T `T-square`
##   <chr>    <dbl>  <dbl>      <dbl>
## 1 MH-032990 193.  6.79      46.1
## 2 MH-032990 193.  6.79      46.1
## 3 MH-032990 193.  6.79      46.1
## 4 MH-032990 193.  6.79      46.1
## 5 MH-050877 193. -2.46     6.06
## 6 MH-050877 193. -2.46     6.06
## 7 MH-050877 193. -2.46     6.06
## 8 MH-050877 193. -2.46     6.06
## 9 MH-052672 193. -7.01    49.2
## 10 MH-052672 193. -7.01    49.2
## 11 MH-052672 193. -7.01    49.2
## 12 MH-052672 193. -7.01    49.2
## 13 MH-091678 193.  2.69    7.22
## 14 MH-091678 193.  2.69    7.22
## 15 MH-091678 193.  2.69    7.22
## 16 MH-091678 193.  2.69    7.22
```

We then sum the squares of T to get the Hybrid sum of squares.

```
sum(trt_ss$`T-square`)
```

```
## [1] 434.1275
```

We use a similar approach to calculate the Error (or Residual) sum of squares. This time we square the error effect (the difference between the observed value and the treatment mean) for each observation in the trial.

```
err_ss = effects_data %>%
  select(Hybrid, mu, E) %>%
  mutate(`E-square` = E^2)

err_ss

## # A tibble: 16 x 4
##   Hybrid     mu     E `E-square`
##   <chr>    <dbl>  <dbl>      <dbl>
## 1 MH-032990 193. -1.30      1.69
## 2 MH-032990 193. -1.10      1.21
## 3 MH-032990 193.  2.70      7.29
## 4 MH-032990 193. -0.300    0.0900
## 5 MH-050877 193. -1.45      2.10
```

```
## 6 MH-050877 193. 0.25 0.0625
## 7 MH-050877 193. 0.850 0.723
## 8 MH-050877 193. 0.350 0.123
## 9 MH-052672 193. 0 0
## 10 MH-052672 193. -1.5 2.25
## 11 MH-052672 193. 2 4
## 12 MH-052672 193. -0.5 0.25
## 13 MH-091678 193. 0.800 0.640
## 14 MH-091678 193. -4.30 18.5
## 15 MH-091678 193. 2 4
## 16 MH-091678 193. 1.5 2.25
```

We again sum the squared error effects to get the Error or Residual sum of squares.

```
sum(err_ss$`E-square`)

## [1] 45.17
```

### 6.5.3 Degrees of Freedom

The *df* column above refers to the degrees of freedom. Remember, the variance is equal to the sum of squares, divided by its degrees of freedom. The hybrid sum of squares is simply the number of treatments minus 1. In this example, there were 4 hybrids, so there were three degrees of freedom for the Hybrid effect. The concept behind the hybrid degrees of freedom is that if we know the means for three hybrids, as well as the population mean, then we can calculate the fourth hybrid mean, as it is determined by the first three hybrids and the population mean. Degrees of freedom are a weird concept, so try not to overanalyze them.

The degrees of freedom for the error or residual effect are a little more confusing. The degrees of freedom are equal to the Hybrid degrees of freedom, times the number of replications. In this case, the error degrees of freedom are 12. The idea behind this is: if for a hybrid you know the values of three observations, plus the hybrid mean, you can calculate the value of the fourth observation.

### 6.5.4 Mean Square

In the Analysis of Variance, the Sum of Squares, divided by the degrees of freedom, is referred to as the “Mean Square”. As we now know, the mean squares are also the variances attributable to the Hybrid and Error terms of our linear model. Our hybrid mean square is about 144.7; the error mean square is about 3.8.

### 6.5.5 F-Value

The F-Value, as introduced earlier, is equal to the hybrid variance, divided by the error variance. In the ANOVA table, F is calculated as the hybrid mean square divided by the error mean square. When the F-value is 1, it means the treatment effect and error effect have equal variances, and equally describe the variance among observed values. In other words, knowing the treatment each plot received adds nothing to our understanding of observed differences.

### 6.5.6 P-value

The F-value is the summary calculation for the relative sizes of our Hybrid and Error variances. Its value is 38.4, which means the Hybrid variance is over 38 times the size of the Error variance. In other words, the Hybrid variance accounts for much, much more of the variation in our observations than the Error variance. But, given this measured F-value, what is the probability the true F-value is 1, meaning the Hybrid and Error variances are the same?

To calculate the probability our F-value could be the product of chance, we use the F-distribution. The shape of the F-distribution, like the t-distribution, changes with the number of replications (which changes the Error degrees of freedom). It also changes with the treatment degrees of freedom.

Please click the link below to access an app where you will be able to adjust the number of treatments and number of replications:

<https://marin-harbur.shinyapps.io/06-f-distribution/>

Adjust those two inputs and observe the change in the response curve. In addition, adjust the desired level of significance and observe how the shaded area changes. Please ignore the different color ranges under the curve when you see them: any shaded area under the curve indicates significance.

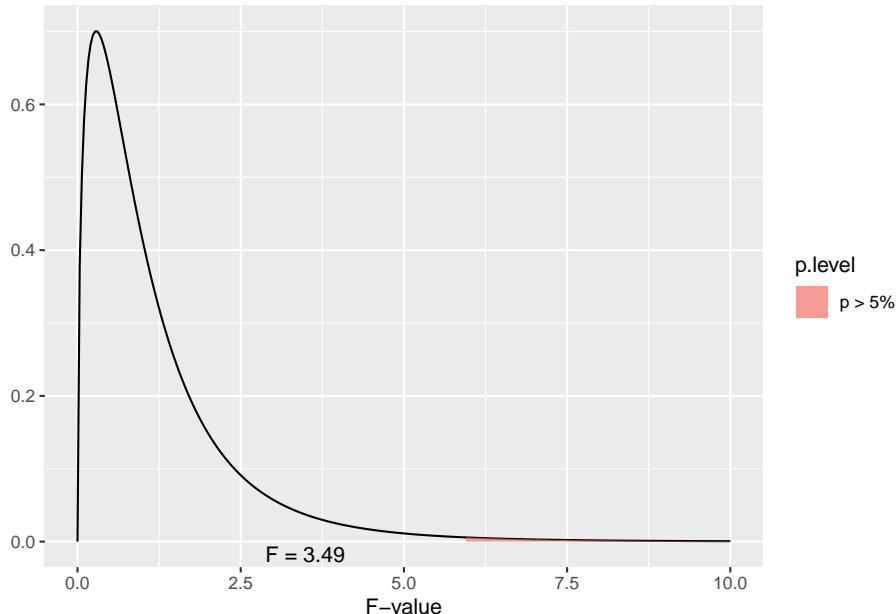
The F-distribution is one-tailed – we are only interested in the proportion remaining in the upper tail. If we were to visualize the boundary for the areas representing  $P \geq 0.05$  for our example above, we would test whether F was in the following portion of the tail.

As we can see, our observed F of 38.4 is much greater than what we would need for significance at  $P \geq 0.05$ . What about  $P \geq 0.01$ ?

```
library(sjPlot)
```

```
## Install package "strengejacke" from GitHub (`devtools::install_github("strengejacke")`)
```

```
dist_f(p = 0.01, deg.f1 = 3, deg.f2 = 12, xmax = 10)
```



Our value is also way beyond the F-value we would need for  $P \geq 0.05$ .

## 6.6 Visualizing How the Anova Table Relates to Variance

Please follow the following link to an app that will allow you to simulate a corn trial with three treatments:

<https://marin-harbur.shinyapps.io/06-anova-variances/>

Use your observations to address the following four questions in Discussion 6.1:

- 1) What do you observe when distance between the trt means increases?
- 2) what do you observe when the pooled standard deviation decreases?
- 3) Set the treatment means to treatment 1 = 180, treatment 2 = 188, and treatment 3 = 192. What do you observe about the shapes of the distribution curve for the treatment means (black curve) and treatment 2 (green curve)?
- 4) What does an F-value of 1 mean?

##Exercise: Completely Randomized Design Anova In this unit we were introduced to the Analysis of Variance (ANOVA) and the most basic experimental

design, the Completely Randomized Design. Running an analysis of variance for this trial is very easy, as we will now see.

### 6.6.1 Case Study: Barley

```
barley_data = read.csv("data-unit-6/exercise_data/barley_crd.csv")
head(barley_data)

##   rep gen yield
## 1  S1 G01  0.05
## 2  S1 G02  0.00
## 3  S1 G03  0.00
## 4  S1 G04  0.10
## 5  S1 G05  0.25
## 6  S1 G06  0.05
```

### 6.6.2 ANOVA

There are two steps to conducting a basic ANOVA analysis. First, define the model. We will use the `lm()` function to create a linear model. We need to provide two arguments to `lm()`. First, the model itself. We are going to model yield as a function of gen, that is, that yield differs among levels of gen. We express this as “yield~gen”. The second argument is simply the dataset in which these variables are located: “barley”.

```
barley_model = lm(yield~gen, data = barley_data)
```

The second step is to use the command `anova()` to generate an ANOVA table from the model.

```
anova(barley_model)
```

```
## Analysis of Variance Table
##
## Response: yield
##             Df Sum Sq Mean Sq F value    Pr(>F)
## gen          9 19604  2178.25   3.599 0.0008369 ***
## Residuals  80 48420   605.24
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

It's that simple.

### 6.6.3 Calculating the Coefficient of Variation

The Coefficient of Variation (CV) is a quality-control measurement for research. It essentially asks: were our experimental units (plots or pots) consistent enough for us to properly observe the difference among treatments? Or was our trial, in essence, sloppy?

Mathematically, the CV is calculated as:

$$CV = \frac{\sqrt{EMS}}{\mu} \times 100$$

Where MSE is the Error (or Residual) Mean Square and  $\mu$  is the population mean. In the barley example, the EMS is 605.2. The population mean for yield is 20.71

So the CV is equal to:

$$CV = \frac{\sqrt{605.2}}{20.71} = \frac{24.60}{20.71} = 118.75$$

We can quickly calculate this for any model we create using the `cv.model()` function in the `agricolae` package.

```
library(agricolae)
cv.model(barley_model)
```

```
## [1] 118.7551
```

This is a really, really large CV. In fact, I checked my math a few times before I accepted it was correct. In corn research, I like to see a CV below 10. For soybean I prefer a CV below 5. Any greater than that, and a trial will be closely examined and potentially excluded from analysis. Before using a CV to evaluate the quality of a trial, however, you should be familiar with CVs experienced by previous researchers in your discipline.

### 6.6.4 Practice: Beet Data

```
beet_data = read.csv("data-unit-6/exercise_data/beets_crd.csv")
head(beet_data)
```

```
##   plot fert yield
## 1     1  None  2.45
## 2     2      P  6.71
```

```
## 3     3    K  3.22
## 4     4    PK 6.34
## 5     5    PN 6.48
## 6     6    KN 3.70
```

Model the relationship of yield and fert by completing the analysis of variance below. Your  $\text{Pr}(>F)$  for fert should be  $1.299\text{e-}10$ . You will need to “uncomment” (delete the hashtags) to run this code.

```
# beet_model = lm( , data = )
# summary.aov(beet_model)
```

Calculate the Coefficient of Variance (CV) using the `cv.model()` function. Your CV should be 17.38.

```
# cv.model()
```

### 6.6.5 Practice: Potato Data

```
potato_data = read.csv("data-unit-6/exercise_data/potato_crd.csv")
head(potato_data)
```

```
##   inf trt row col
## 1   9  F3   4   1
## 2  12  O   4   2
## 3  18  S6   4   3
## 4  10 F12   4   4
## 5  24  S6   4   5
## 6  17 S12   4   6
```

Model the relationship between infection (inf) and treatment (trt) using the analysis of variance. You should get  $\text{Pr}(>F) = 0.0103$ .

Calculate the Coefficient of Variance (CV) using the `cv.model()` function.

## 6.7 Exercise: “Treatment Means”

The next step in a multiple treatment trial is to summarise the treatment means. Below you will learn how to calculate means and display them using a bar plot.

### 6.7.1 Case Study: Barley

This is the same dataset we used in the previous exercise.

```
library(tidyverse)
barley_data = read.csv("data-unit-6/exercise_data/barley_crd.csv")
head(barley_data)

##   rep gen yield
## 1 S1  G01  0.05
## 2 S1  G02  0.00
## 3 S1  G03  0.00
## 4 S1  G04  0.10
## 5 S1  G05  0.25
## 6 S1  G06  0.05
```

### 6.7.2 Calculating Treatment Means

We can create a new dataset, containing treatment means, using another couple of functions from the *tidyverse* package: *group\_by()* and *summarise()*. We will also use the “pipe”, `%>%`, to feed the results of one line to the next.

We start with “barley `%>%`”, which feeds the barley dataset to the next line. There, we use the *group\_by()* command to tell R we want to group the observations by *gen* (genetics). This means any summary calculations on the data will be done separately by level for the *gen* factor. Finally, we tell the data which variables to summarise. We use the *summarise* command to tell R to create a new variable, *yield\_mean*, which is equal to the mean of observed yield values for each level of *gen*.

```
barley_means = barley_data %>%
  group_by(gen) %>%
  summarise(yield_mean=mean(yield))

barley_means

## # A tibble: 10 x 2
##       gen   yield_mean
##   <chr>     <dbl>
## 1 G01      4.2
## 2 G02      4.77
## 3 G03      7.34
## 4 G04      9.57
## 5 G05      14
```

```
##   6 G06      21.8
##   7 G07      24.2
##   8 G08      34.8
##   9 G09      37.2
##  10 G10     49.3
```

We can see now that our values range from 4.20 to 49.33. Since the original data only had two decimal places, we should probably round these results. We can do that quickly using the *mutate()* and *round()* commands to our code. *mutate* tells R to create a new column. The *round()* function takes two arguments: the variable to round, and the number of decimal places to round to. In this case, we will create a new variable with the same name, *yield\_mean*, as the old variable. The value of the new *yield mean* will be equal to the old *yield mean*, rounded to two decimal places.

```
barley_means = barley_data %>%
  group_by(gen) %>%
  summarise(yield_mean=mean(yield)) %>%
  mutate(yield_mean = round(yield_mean, 2))

barley_means

## # A tibble: 10 x 2
##       gen   yield_mean
##   <chr>     <dbl>
## 1 G01      4.2
## 2 G02      4.77
## 3 G03      7.34
## 4 G04      9.57
## 5 G05      14
## 6 G06     21.8
## 7 G07     24.2
## 8 G08     34.8
## 9 G09     37.2
## 10 G10    49.3
```

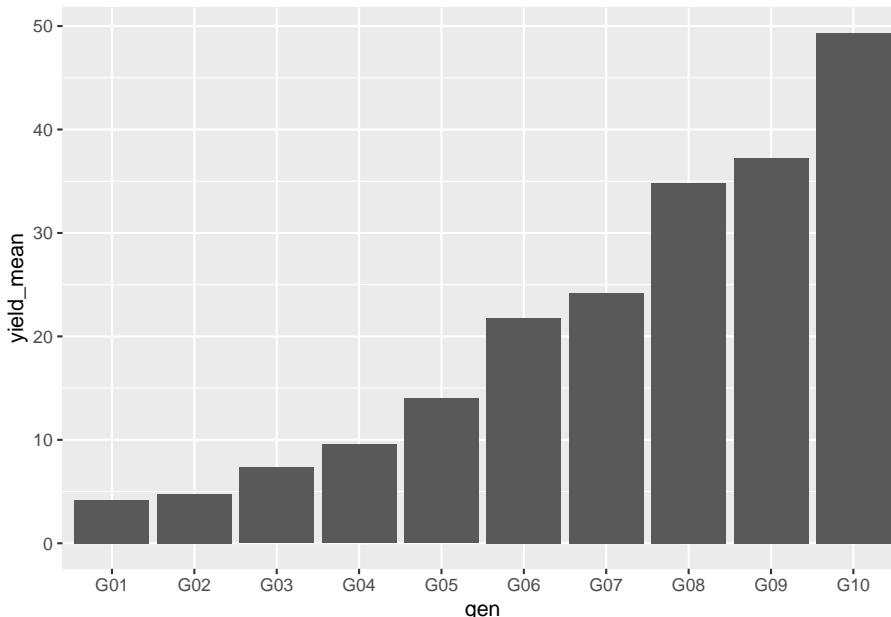
### 6.7.3 Plotting the Means

We can use the *ggplot()* function in R to plot our results. The first line, the *ggplot()* statement, uses “*data =*” to define the data frame, and the *aes()* argument to declare the aesthetics to be plotted. Aesthetics are any information in the chart whose position or appearance are dependent on variables from the dataset. In other words, anything in the chart that might change with the value

of an observation. In this case, the x-position of our data to be plotted will be defined by gen. The y-position will be defined by yield.

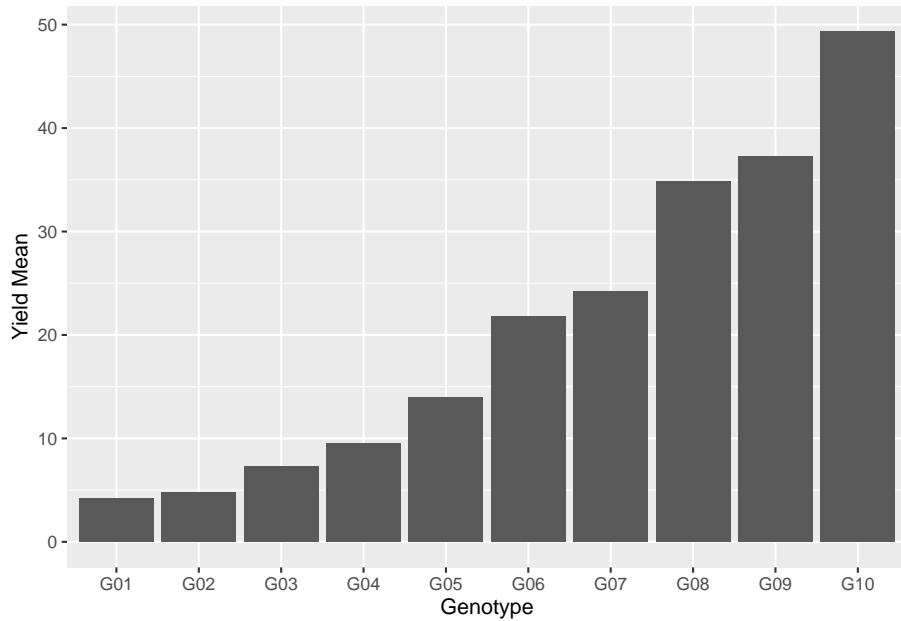
The second line defines a “geom”, the type of geometry that will be used to represent the data. In this case, we will use geom\_bar() to construct a bar plot. The final thing we need to do is to tell R whether the height of each bar should be determined by the number of observations, another statistical summary, or the value for y we declared in the initial ggplot statement. In this case, we want to use the value we declared above, yield\_mean, to determine the top of the bars. The stats=“identity” tells R to do just that.

```
ggplot(data=barley_means, aes(x=gen, y=yield_mean)) +
  geom_bar(stat="identity")
```



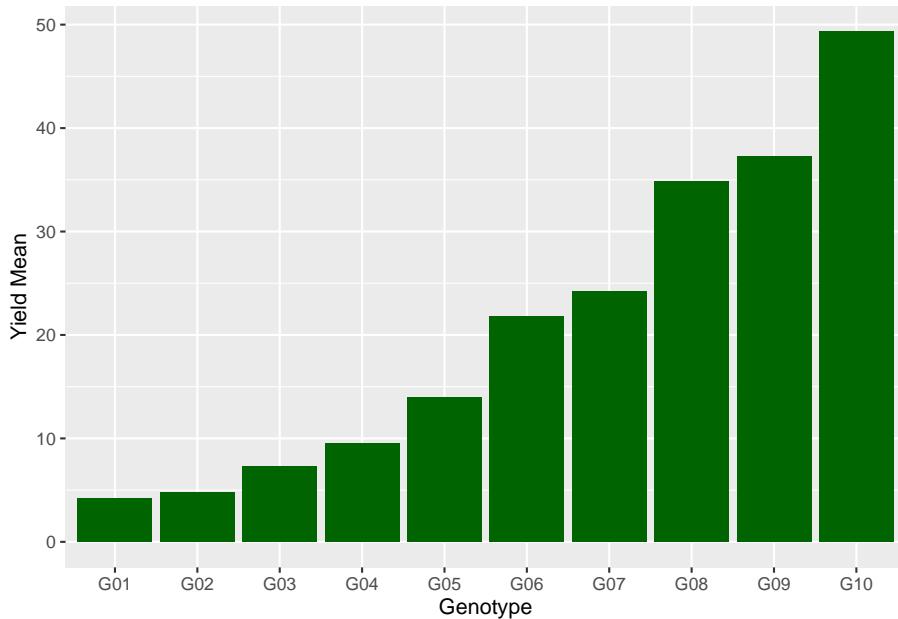
There are all sorts of things we can do in R with our plots. Say we wanted to change the labels on the x or y axes. We just add those using the “labs()” option:

```
ggplot(data=barley_means, aes(x=gen, y=yield_mean)) +
  geom_bar(stat="identity") +
  labs(x = "Genotype", y = "Yield Mean")
```



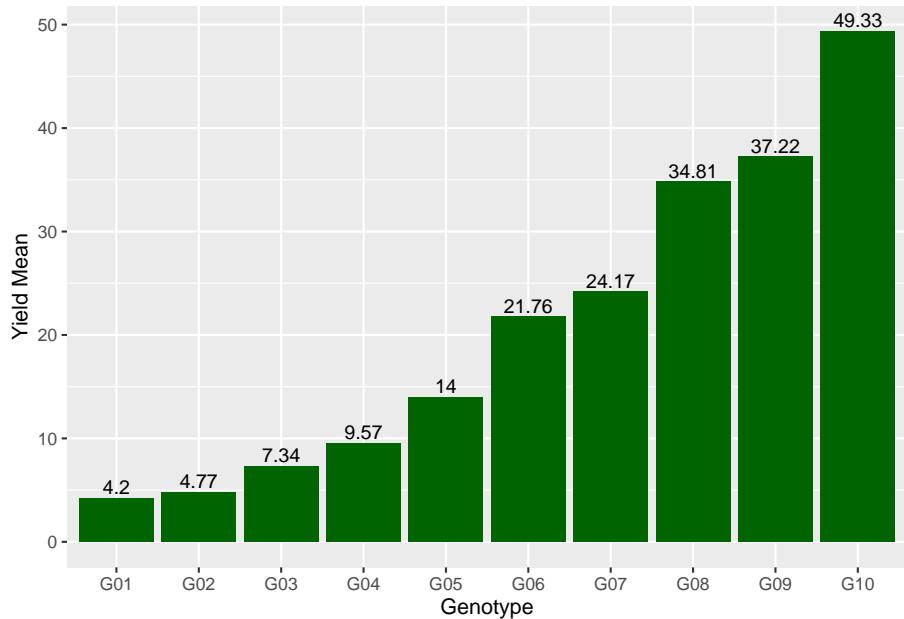
We can change the bar color by adding the `fill()` command to `geom_bar`.

```
ggplot(data=barley_means, aes(x=gen, y=yield_mean)) +  
  geom_bar(stat="identity", fill="darkgreen") +  
  labs(x = "Genotype", y = "Yield Mean")
```



We can include the actual value above each bar by adding a second geometry, `geom_text`. The `aes()` argument to `geom_text` tells it what variable to use as the label (it also tells it where to place the label). In addition to the `aes()` argument, “`vjust=-0.03`” tells R to adjust the label upward a few points so it doesn’t touch the bar. The font size is increased with “`size = 3.5`”.

```
ggplot(data=barley_means, aes(x=gen, y=yield_mean)) +
  geom_bar(stat="identity", fill="darkgreen") +
  labs(x = "Genotype", y = "Yield Mean") +
  geom_text(aes(label=yield_mean), vjust=-0.3, size=3.5)
```



#### 6.7.4 Practice: Beet Data

```
beet_data = read.csv("data-unit-6/exercise_data/beets_crd.csv")
head(beet_data)
```

```
##   plot fert yield
## 1    1  None  2.45
## 2    2     P  6.71
## 3    3     K  3.22
## 4    4    PK  6.34
## 5    5    PN  6.48
## 6    6    KN  3.70
```

Calculate the yield mean by fertilizer level by completing the code below. Remember, you will need to uncomment (remove hashtags from) the code before you can run it. (Hint: you can add or remove multiple lines of comments by highlighting those lines and typing CTRL-SHIFT-C.)

```
# beet_means = beet_data %>%
#   group_by() %>%
#   summarise()
```

Create a simple bar plot using ggplot() and geom\_bar().

```
# ggplot(data=beet_means, aes(x= , y= )) +  
#   geom_bar(stat = )
```

### 6.7.5 Practice: Potato Data

```
potato_data = read.csv("data-unit-6/exercise_data/potato_crd.csv")  
head(potato_data)
```

```
##   inf trt row col  
## 1  9  F3   4   1  
## 2 12   0   4   2  
## 3 18  S6   4   3  
## 4 10 F12   4   4  
## 5 24  S6   4   5  
## 6 17 S12   4   6
```

Calculate the mean infection (inf) by treatment (trt) and assign it to the data frame “potato\_means”.

Use the potato\_means data frame to create a simple bar plot.



## Chapter 7

# Multiple Treatment Designs

In the last unit, we were introduced to multiple-treatment experiments, using an example with which many of you are familiar: a hybrid research or demonstration trial. Recall how the analysis of variance worked: we compared two sources of variation to see how much of that variation each of them explained. There were two effects in our original trial: treatment and error

$$Y_{ij} = \mu + T_i + \epsilon_{i(j)}$$

The Analysis of Variance (ANOVA) was used to calculate and compare these variances. First, the sums of squares from the treatment means and the error (the summed distributions of observations around each treatment mean) were calculated. By dividing the sum of squares by their degrees of freedom, the we obtained the treatment and error mean squares, also known as their variances. The F-value was derived from the ratio of the treatment variance to the error variance. Finally, the probability that the difference among treatments was zero, given the F-value we observed, was calculated using the F-distribution.

This experimental design is known as a *Completely Randomized Design*. It is the simplest multiple-treatment design there is. In this unit, we will learn two other designs commonly used in trials:

- Randomized Complete Block Design
- Two-Way Factorial Design

These designs, we will see, use additional sources of variation to either expand the number of treatments we can evaluate, or reduce the error (unexplained variation) in our trials so that we can better identify treatment effects.

## 7.1 Randomized Complete Block Design

If you have participated in agronomic research, you have likely heard references to a Randomized Complete Block Design (RCBD). We first discussed blocking when we learned about side-by-side trials. When we block treatments, we force treatments to occur in closer proximity to each other than they likely would were they assigned at random. The best way to understand this is to look at a plot map.

```
library(tidyverse)

## -- Attaching packages ----- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr    1.0.8
## v tidyr   1.2.0     v stringr  1.4.0
## v readr   2.1.2     vforcats  0.5.1

## Warning: package 'ggplot2' was built under R version 4.1.3

## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()   masks stats::lag()

col = rep(1:4, each=4)
row = rep(1:4, 4)

block = rep(1:4, each=4)

set.seed(5)
crd = sample(col, 16, replace = FALSE)
rcbd_list = list()
for(i in c(1:4)){
  set.seed(i)
  z = sample(1:4, 4, replace = FALSE)
  rcbd_list[[i]] = z
}

rcbd = do.call(c, rcbd_list)

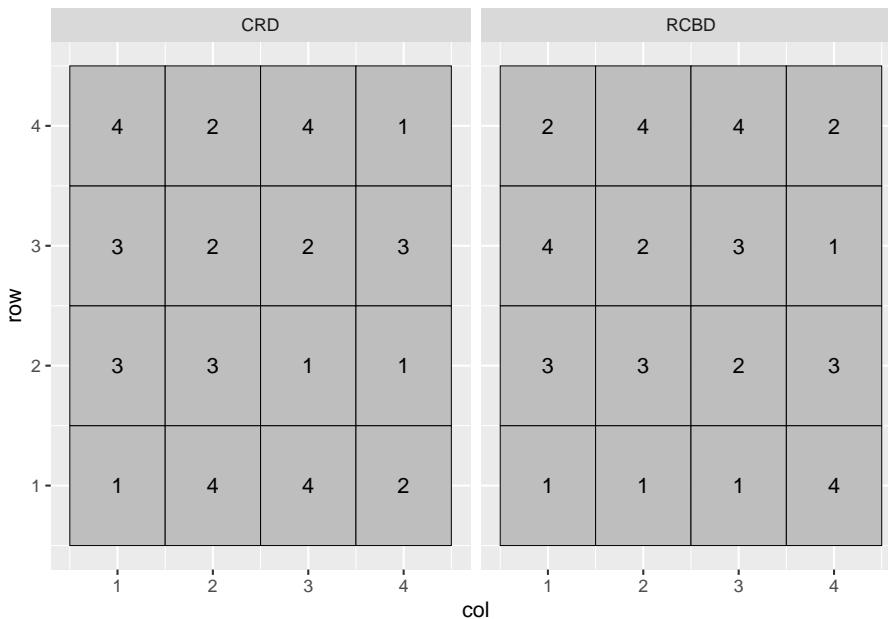
dataset = cbind(col, row, block, crd, rcbd) %>%
  as.data.frame() %>%
  gather(design, treatment, crd, rcbd) %>%
  mutate(design = toupper(design)) %>%
```

```

mutate(block_effect = case_when(col==1 ~ 3,
                                 col==2 ~ 1,
                                 col==3 ~ -1,
                                 col==4 ~ -3))

dataset %>%
  ggplot(aes(x=col, y=row, label=treatment)) +
  geom_tile(fill="grey", color="black") +
  geom_text(aes(label = treatment)) +
  facet_grid(. ~ design)

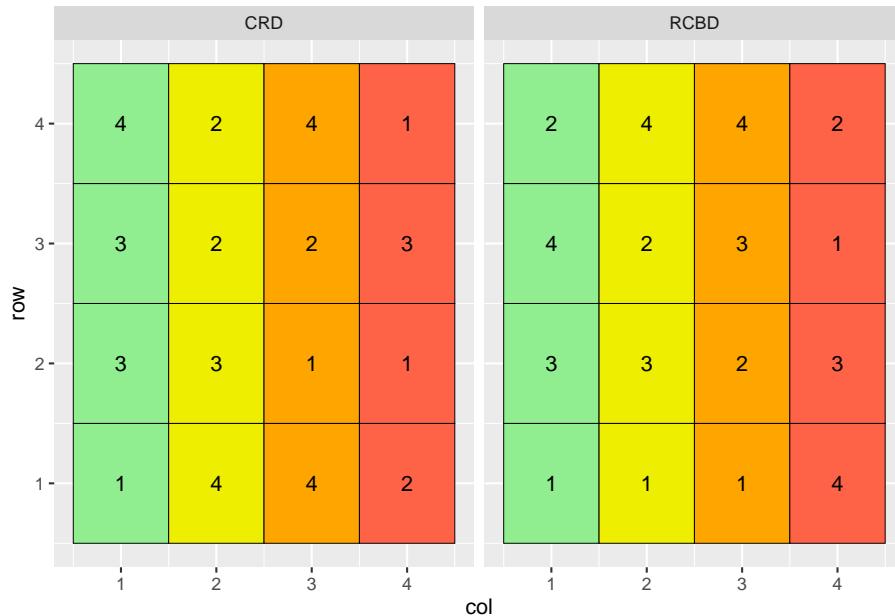
```



In the plot above, the Completely Randomized Design (CRD) is shown on the left, and the Randomized Complete Block Design (RCBD) on the right. In the Completely Randomized Design, any treatment can occur anywhere in the the plot. Note that in the left plot, treatment 3 occurs twice in the first column of plots while treatment 2 does not occur at all. Treatment 2 occurs twice in the second column, but there is no treatment 1. In the Randomized Complete Block Design, each treatment must occur once, and only once, per column. In this case, the treatments are blocked on column.

Why block? Let's suppose each column in the plot map above has a different soil type, with soils transitioning from more productive to less productive as columns increase from 1 to 4:

```
dataset %>%
  ggplot(aes(x=col, y=row, label=treatment)) +
  geom_tile(aes(fill=as.factor(col)), color="black") +
  geom_text(aes(label = treatment)) +
  scale_fill_manual(values = c("lightgreen", "yellow2", "orange1", "tomato")) +
  facet_grid(. ~ design) +
  theme(legend.position = "none")
```



Note that treatment 3 occurs three times in the more productive soils of columns 1 and 2, and only once in the less productive soils of columns 3 and 4. Conversely, treatment 1 occurs three times in the less productive soils of columns 3 and 4, but only once in the more productive soil of columns 1 or 2.

If the mean effect of treatment 3 is greater than the mean effect of treatment 1, how will we distinguish the effects of treatment and error? It is a moot question: we can't. Our linear model is *additive*:

$$Y_{ij} = \mu + T_i + \epsilon_{i(j)}$$

The term additive is important: it means we assume that treatment and error effects do not interact – they independently add or subtract from the population mean. If the measured effect of treatment is dependent on plot error, the model fails.

The plot on the right has blocked treatments according to column. Doing that allows us to remove the effect of soil type, which consistently varies from column

to column, from the effect of error, which is random. Our linear model changes as well:

$$Y_{ij} = \mu + B_i + T_j + BT_{ij}$$

Where  $Y_{ij}$  is the individual value,  $\mu$  is the population mean,  $B_i$  is the block effect,  $T_j$  is the treatment effect, and  $BT_{ij}$  is the interaction of block and treatment, also known as the error effect.

### 7.1.1 Case Study: Randomized Complete Block Design

A field trial outside Goshen, Indiana, evaluated the effect of seed treatments on soybean yield. Treatments were as follows: - A: untreated - B: metalaxy only - C: metalaxy + insecticide - D: metalaxy + insecticide + nematicide

Treatments were arranged in a Randomized Complete Block Design.

```
library(tidyverse)

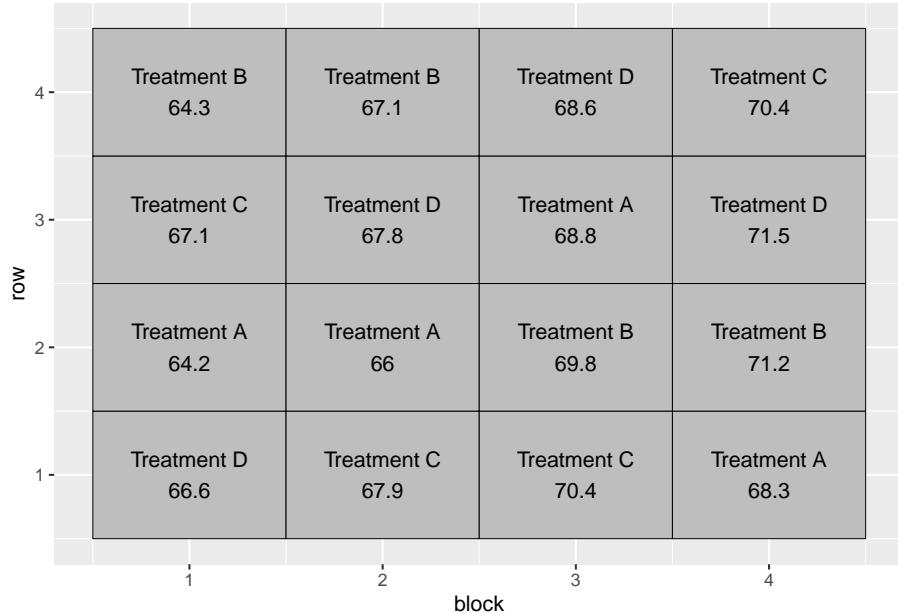
st = data.frame(obs=c(1:16))
set.seed(041383)
mu = rep(c(68.3))
Block = rep(1:4, each=4)
B = rep(c(-2.2, -0.8, 0.3, 1.2), each = 4)
Treatment = rep(c("A", "B", "C", "D"), 4)
T = rep(c(-1.1, -0.3, 0.8, 1.6), 4)
BT = rnorm(n=16, mean = 0, sd = 1)

st_data = st %>%
  cbind(Block, Treatment, mu, B, T, BT) %>%
  mutate(BT = round(BT, 1)) %>%
  mutate(Y = mu+B+T+BT) %>%
  group_by(Block) %>%
  sample_n(4) %>%
  ungroup() %>%
  select(-obs) %>%
  mutate(Block = as.factor(Block))

st_data$row = rep(1:4, 4)

st_data %>%
  mutate(plot_label = paste0("Treatment ", Treatment, "\n", Y)) %>%
  ggplot(aes(x=block, y=row, label=plot_label)) +
```

```
geom_tile(fill="grey", color="black") +
  geom_text(aes(label = plot_label))
```



### 7.1.1.1 Linear Additive Model

In this example, the linear additive model is:

$$Y_{ij} = \mu + B_i + T_j + BT_{ij}$$

Or, with regard to our particular trial:

$$Yield = Population\ Mean + Block\ Effect + Treatment\ Effect + Block \times Treatment\ Interaction$$

We can see how the additive model works in the following table:

```
st_effects = st_data %>%
  select(Block, row, Treatment, mu, B, T, BT, Y)

knitr::kable(st_effects)
```

Block	row	Treatment	mu	B	T	BT	Y
1	1	D	68.3	-2.2	1.6	-1.1	66.6
1	2	A	68.3	-2.2	-1.1	-0.8	64.2
1	3	C	68.3	-2.2	0.8	0.2	67.1
1	4	B	68.3	-2.2	-0.3	-1.5	64.3
2	1	C	68.3	-0.8	0.8	-0.4	67.9
2	2	A	68.3	-0.8	-1.1	-0.4	66.0
2	3	D	68.3	-0.8	1.6	-1.3	67.8
2	4	B	68.3	-0.8	-0.3	-0.1	67.1
3	1	C	68.3	0.3	0.8	1.0	70.4
3	2	B	68.3	0.3	-0.3	1.5	69.8
3	3	A	68.3	0.3	-1.1	1.3	68.8
3	4	D	68.3	0.3	1.6	-1.6	68.6
4	1	A	68.3	1.2	-1.1	-0.1	68.3
4	2	B	68.3	1.2	-0.3	2.0	71.2
4	3	D	68.3	1.2	1.6	0.4	71.5
4	4	C	68.3	1.2	0.8	0.1	70.4

In the first row of the table, we see that the observed yield, Y, is:

$$Y = 68.3 + (-2.2) + (1.3) + (-1.1) = 66.3$$

Similarly, in the fifth row:

$$Y = 68.3 + (-0.8) + (0.5) + (-0.4) = 67.6$$

### 7.1.1.2 Analysis of Variance

We can use the linear additive model above with R to create the proper linear model:

```
model = aov(Y ~ Block + Treatment, st_data)
anova(model)

## Analysis of Variance Table
##
## Response: Y
##             Df Sum Sq Mean Sq F value    Pr(>F)
## Block       3 56.250 18.7500 23.7175 0.0001313 ***
## Treatment   3 10.485  3.4950  4.4209 0.0359018 *
## Residuals  9  7.115  0.7906
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Note that in the above model, we only specify the Block and Treatment sources of variation. Any source of variation not included in the model and, if you will “leftover” from the model, is pooled into the Residuals, or error. In the model above, the interaction of Block and Treatment (BT) is not specified, for it is the source of any variation we observe from plot to plot.

Two sources of variation are tested above: Block and Treatment. The F-value for both is calculated by dividing their Mean Square by the Residual (or Error) Mean Square. The probability that the difference among blocks or treatment is zero, given their observed F-value, is reported in the  $Pr(> F)$  column.

The Block effect is usually of less interest in analyzing table results. If it is insignificant, that may indicate we don't need to block in this location in the future, but in the vast majority of trials there will be at least some benefit to blocking.

The most important effect of blocking is seen upon examining the Sum of Squares (“Sum Sq”) column. Here we can see just how much the Residual Sum of Squares was reduced by including blocks in the model. Had we not included the Block term in our model, our Residual Sum of Squares would have been about 63.4. Even given the greater residual degrees of freedom (which would have included the three degrees of freedom that were assigned to Block, the residual mean square would have been about  $64 \div 12 = 5.3$ . Without even calculating F, we can see the error mean square would have been larger than the treatment mean square, meaning there was more variance within treatments than between them. The Treatment effect would not have been significant.

## 7.2 Factorial Design

Agronomy is all about interactions. How do hybrids differ in their response to nitrogen? Response to fungicide? Does a fungicide increase yield more when it is sprayed on corn at V5 or VT? Does the effect of a starter fertilizer depend whether it is applied in-row or in a 2x2 band? How does a crop respond to population in 30-inch rows vs 15-inch rows?

To grow a successful crop requires not a single input, but dozens, some of which we can manage and others we can't. So the issue of how different treatments interact is critical. It is also often more efficient and informative for us to study these interactions together in a single trial, than to study them separately in two or more trials.

Before we go further, some nomenclature. In factorial design, treatments that differ in a single variable are called levels, and these levels together compose a factor. Here are some examples of levels and factors:

- multiple micronutrient products all applied at the V5 stage in corn – the product composition is the level, the timing the factor

- a single fungicide product, applied at V5, V10, or VT in corn – the timing is the level, the fungicide composition the factor
- a single adjuvant, tested with different nozzles – the nozzle is the level, the adjuvant composition the factor
- multiple hybrids, grown in rotation following soybean – the hybrid is the level, the crop rotation is the factor

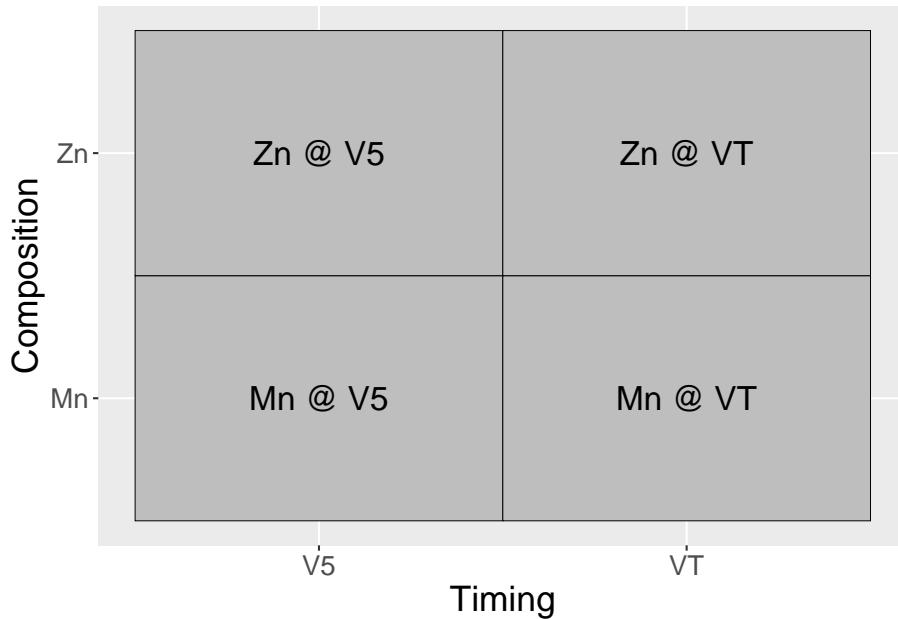
As you may have anticipated, a factorial design combines two or more factors, each containing two or more levels. For example:

- Factor “Composition” includes two levels of foliar micronutrient product: Mn and Zn
- Factor Timing” includes two levels of timing: V5 and VT

In a factorial design, every level of factor Composition will occur with every level of factor Timing. We can visualize these treatment combinations the same way we might visualize a Punnet square in Mendelian genetics. The main effects are given on the axes and the particular treatment combinations are in the cells.

```
factorial_trts_1 = data.frame(Timing = c("V5", "VT", "V5", "VT"),
                               Composition = c("Mn", "Mn", "Zn", "Zn"),
                               trt = c("Mn @ V5", "Mn @ VT", "Zn @ V5", "Zn @ VT"))

factorial_trts_1 %>%
  ggplot(aes(x=Timing, y=Composition)) +
  geom_tile(color="black", fill="grey") +
  geom_text(aes(label = trt), size=6) +
  theme(axis.title = element_text(size=18),
        axis.text = element_text(size=14))
```

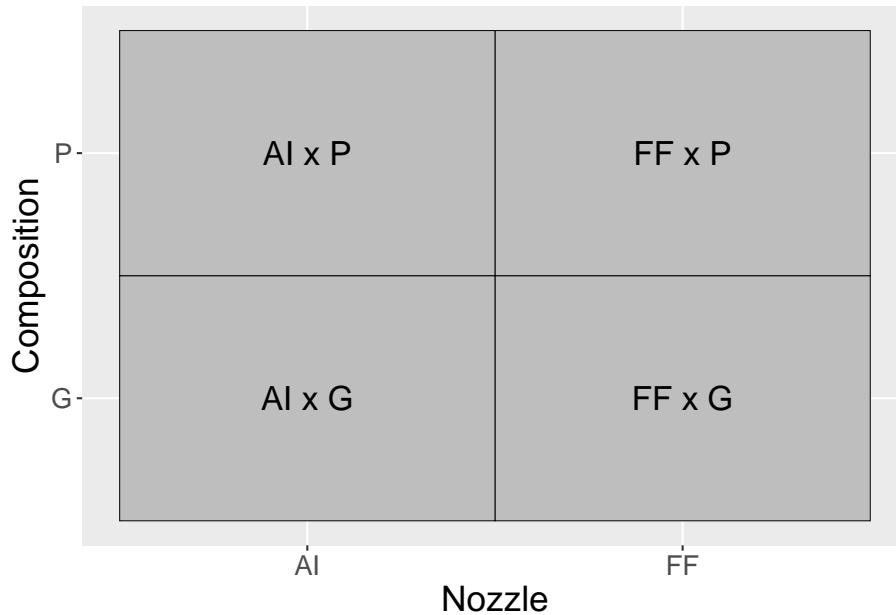


In another example: - Factor “Composition” consists of two adjuvant ingredients: guar (G) or an polyacrylamide (P) - Nozzles are Flat Fan (F) or AI nozzle (A)

Our treatments in the factorial design, then, are:

```
factorial_trts_2 = data.frame(Nozzle = c("FF", "AI", "FF", "AI"),
                               Composition = c("G", "G", "P", "P"),
                               trt = c("FF x G", "AI x G", "FF x P", "AI x P"))

factorial_trts_2 %>%
  ggplot(aes(x=Nozzle, y=Composition)) +
  geom_tile(color="black", fill="grey") +
  geom_text(aes(label = trt), size=6) +
  theme(axis.title = element_text(size=18),
        axis.text = element_text(size=14))
```



### 7.2.1 Case Study 1

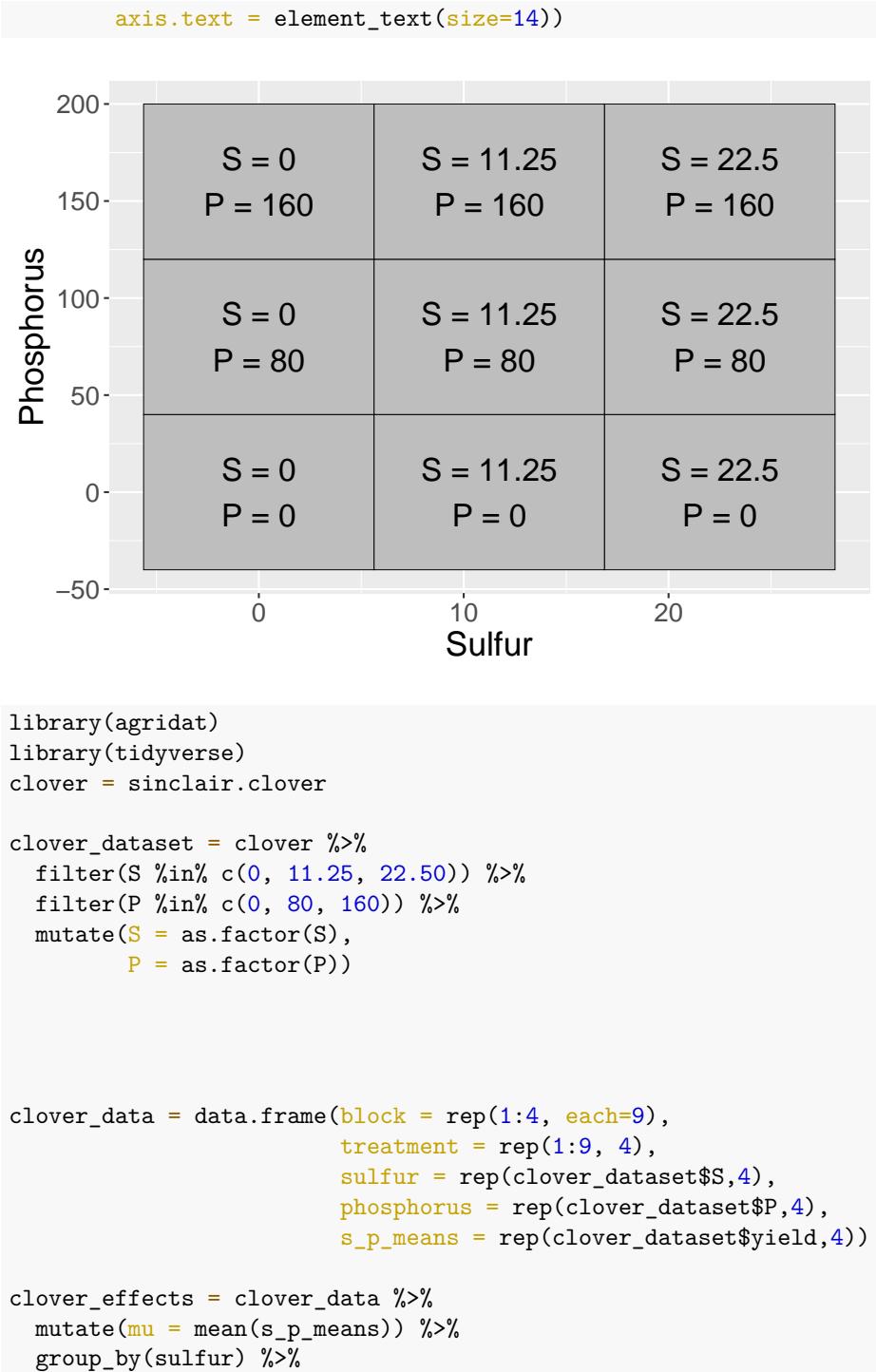
Our case study is a clover forage trial conducted in New Zealand from 1992 to 1994. This dataset is publically available as part of the *agridat* package in R. For this first case study, we will focus on a subset of the data. The two factors were sulfur (S) and phosphorus (P) fertilizer. Sulfur was applied at 0, 11.25, or 22.5 kg/ha, while phosphorus was applied at 0, 40, and 80 kg/ha. Yield is reported in tons/hectare.

Factorial trials are often nicknamed by the number of levels of each factor. In this case, we have a *three – by – three*, or  $3 \times 3$  trial. We can visualize the factorial combinations as producing the following nine treatments:

```
factorial_trts_3_main = data.frame(Sulfur = rep(c(0, 11.25, 22.5),3),
                                    Phosphorus = rep(c(0,80,160), each=3))

factorial_trts_3_combos = factorial_trts_3_main %>%
  mutate(trt = paste0("S = ", Sulfur, "\n", "P = ", Phosphorus))

factorial_trts_3_combos %>%
  ggplot(aes(x=Sulfur, y=Phosphorus)) +
  geom_tile(color="black", fill="grey") +
  geom_text(aes(label = trt), size=6) +
  theme(axis.title = element_text(size=18),
```



```

mutate(S = mean(s_p_means) - mu) %>%
ungroup() %>%
group_by(phosphorus) %>%
mutate(P = mean(s_p_means) - mu) %>%
ungroup() %>%
group_by(sulfur, phosphorus) %>%
mutate(SP = mean(s_p_means) - (mu + S + P)) %>%
ungroup() %>%
select(-treatment, -s_p_means) %>%
mutate(mu = round(mu,2),
       S = round(S,2),
       P = round(S,2),
       SP = round(SP,2))

set.seed(073020)
clover_final = clover_effects %>%
  mutate(Error = rnorm(36, 0, 0.4)) %>%
  mutate(Error = round(Error, 1)) %>%
  mutate(Yield = mu + S + P + SP + Error)

```

When we deal with factorial designs, it is important to visualize the data. We can observe the data patterns using a line plot. We see clover yield increased with sulfur and phosphorus. We notice, however, the difference in yield between P=80 and P=160 is greater when S=0 than when S=11.25 or S=22.5. We also notice the difference between sulfur=0 and sulfur=22.5 is greater for P=160 than P=0.

```

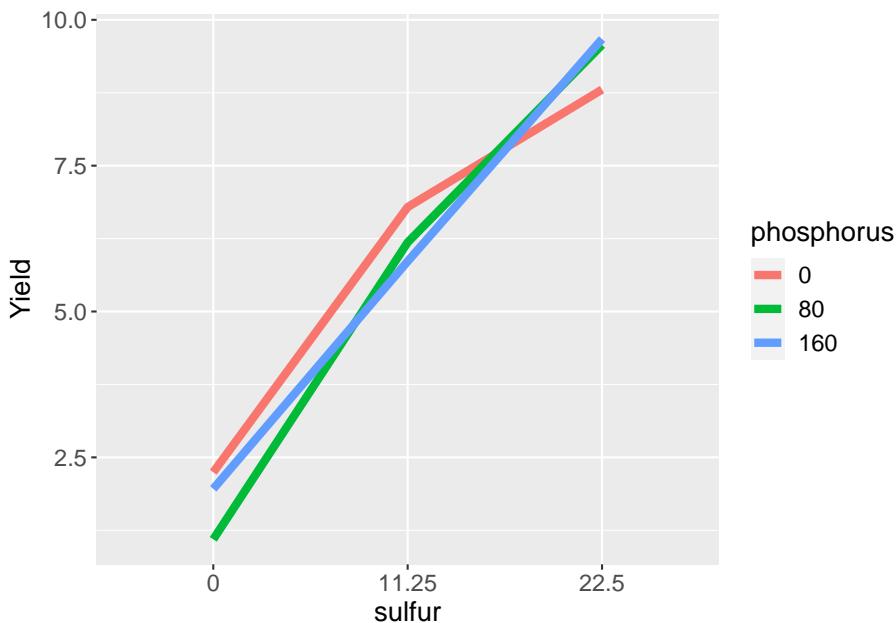
clover_final %>%
  group_by(sulfur, phosphorus) %>%
  summarise(Yield = mean(Yield)) %>%
  ungroup() %>%
  ggplot(aes(x=sulfur, y=Yield, group=phosphorus)) +
  geom_line(aes(color=phosphorus), size=2) +
  theme(axis.text = element_text(size=12),
        axis.title = element_text(size=14),
        legend.text = element_text(size=12),
        legend.title = element_text(size=14))

```

```

## `summarise()` has grouped output by 'sulfur'. You can override using the
## `.` argument.

```



These differences are called *interactions* and are the most interesting part of a factorial design. In this case, it is no surprise that crop yield increases with sulfur fertilization in a sulfur soil. It is interesting, however, that the response to sulfur appears to be dependent on phosphorus fertilization, and lends support to Liebig's *Law of the Minimum*, which states that addition of a particular nutrient will not increase yield if other nutrients are more deficient.

### 7.2.1.1 Linear Additive Model

For our trial with two factors, the linear model is:

$$Y_{ijk} = \mu + S_i + P_j + SP_{ij} + \epsilon_{ij(k)}$$

Each observed value, then, is the sum (additive value) of the population mean ( $\mu$ ), sulfur effect ( $S$ ), phosphorus effect ( $P$ ), the interaction of sulfur and phosphorus ( $SP$ ), and the error ( $\epsilon$ ). As in previous models, error is the random source of variation due to environmental and instrumental inconsistency.  $S_i$  and  $P_j$  are described as *main effects*.  $SP_{ij}$  is the interaction.

In the table below, the effects from this linear model are broken out:

```
knitr::kable(clover_final)
```

block	sulfur	phosphorus	mu	S	P	SP	Error	Yield
1	0	0	5.72	-2.06	-2.06	0.37	-0.1	1.87
1	0	80	5.72	-2.06	-2.06	-0.38	0.2	1.42
1	0	160	5.72	-2.06	-2.06	0.01	0.1	1.71
1	11.25	0	5.72	0.24	0.24	0.29	0.2	6.69
1	11.25	80	5.72	0.24	0.24	0.16	-0.4	5.96
1	11.25	160	5.72	0.24	0.24	-0.44	0.0	5.76
1	22.5	0	5.72	1.81	1.81	-0.66	0.6	9.28
1	22.5	80	5.72	1.81	1.81	0.23	0.2	9.77
1	22.5	160	5.72	1.81	1.81	0.43	-0.3	9.47
2	0	0	5.72	-2.06	-2.06	0.37	0.4	2.37
2	0	80	5.72	-2.06	-2.06	-0.38	0.0	1.22
2	0	160	5.72	-2.06	-2.06	0.01	0.2	1.81
2	11.25	0	5.72	0.24	0.24	0.29	1.0	7.49
2	11.25	80	5.72	0.24	0.24	0.16	0.1	6.46
2	11.25	160	5.72	0.24	0.24	-0.44	-0.4	5.36
2	22.5	0	5.72	1.81	1.81	-0.66	-0.2	8.48
2	22.5	80	5.72	1.81	1.81	0.23	-0.1	9.47
2	22.5	160	5.72	1.81	1.81	0.43	0.2	9.97
3	0	0	5.72	-2.06	-2.06	0.37	0.4	2.37
3	0	80	5.72	-2.06	-2.06	-0.38	-0.1	1.12
3	0	160	5.72	-2.06	-2.06	0.01	0.7	2.31
3	11.25	0	5.72	0.24	0.24	0.29	0.2	6.69
3	11.25	80	5.72	0.24	0.24	0.16	-0.6	5.76
3	11.25	160	5.72	0.24	0.24	-0.44	0.9	6.66
3	22.5	0	5.72	1.81	1.81	-0.66	0.4	9.08
3	22.5	80	5.72	1.81	1.81	0.23	-0.3	9.27
3	22.5	160	5.72	1.81	1.81	0.43	0.0	9.77
4	0	0	5.72	-2.06	-2.06	0.37	0.4	2.37
4	0	80	5.72	-2.06	-2.06	-0.38	-0.6	0.62
4	0	160	5.72	-2.06	-2.06	0.01	0.4	2.01
4	11.25	0	5.72	0.24	0.24	0.29	-0.2	6.29
4	11.25	80	5.72	0.24	0.24	0.16	0.2	6.56
4	11.25	160	5.72	0.24	0.24	-0.44	-0.1	5.66
4	22.5	0	5.72	1.81	1.81	-0.66	-0.3	8.38
4	22.5	80	5.72	1.81	1.81	0.23	0.2	9.77
4	22.5	160	5.72	1.81	1.81	0.43	-0.3	9.47

### 7.2.1.2 Analysis of Variance

In R, we use the same approach as previous weeks. All terms from the linear model, except the population mean and error, are included in the model statement.

```
library(broom)

## Warning: package 'broom' was built under R version 4.1.3

# define the model
model = aov(Yield ~ sulfur + phosphorus + sulfur:phosphorus, data = clover_final)
#run the anova
anova_tab = tidy(model)

knitr::kable(anova_tab)
```

term	df	sumsq	meansq	statistic	p.value
sulfur	2	349.0468222	174.5234111	1227.117734	0.0000000
phosphorus	2	0.6720889	0.3360444	2.362812	0.1133374
sulfur:phosphorus	4	5.7705111	1.4426278	10.143477	0.0000381
Residuals	27	3.8400000	0.1422222	NA	NA

In the table, we see that the main effect of sulfur is significant at the  $P \leq 0.05$  level. The phosphorus effect is not significant. The interaction (sulfur:phosphorus) effect is also significant at the  $P \leq 0.05$  level.

When an interaction is significant, we should examine the effect of one factor independently at each level of the other. We can group analyses of one factor by levels of another factor using the *group\_by* command in R. In this case we will tell R to run the analysis of variance for the sulfur effect separately for each level of P.

```
library(broom)
slice_1 = clover_final %>%
  group_by(phosphorus) %>%
  do(tidy(aov(.~sulfur))) %>%
  as.data.frame() %>%
  mutate(term = gsub("[.] [\$]", "", term))

knitr::kable(slice_1)
```

phosphorus	term	df	sumsq	meansq	statistic	p.value
0	sulfur	2	90.33447	45.1672333	264.8242	0
0	Residuals	9	1.53500	0.1705556	NA	NA
80	sulfur	2	145.58927	72.7946333	671.9505	0
80	Residuals	9	0.97500	0.1083333	NA	NA
160	sulfur	2	118.89360	59.4468000	402.2716	0
160	Residuals	9	1.33000	0.1477778	NA	NA

We see that the effect of sulfur is significant at each level of phosphorus. We can group analyses of phosphorus by each level of sulfur.

```

library(broom)
slice_2 = clover_final %>%
  group_by(sulfur) %>%
  do(tidy(aov(.~Yield ~ .phosphorus))) %>%
  as.data.frame() %>%
  mutate(term = gsub("[.] [\$]", "", term))

knitr::kable(slice_2)

```

sulfur	term	df	sumsq	meansq	statistic	p.value
0	phosphorus	2	2.869267	1.4346333	17.331141	0.0008196
0	Residuals	9	0.745000	0.0827778	NA	NA
11.25	phosphorus	2	1.782067	0.8910333	3.734249	0.0659399
11.25	Residuals	9	2.147500	0.2386111	NA	NA
22.5	phosphorus	2	1.791267	0.8956333	8.507335	0.0084258
22.5	Residuals	9	0.947500	0.1052778	NA	NA

Whoa, what is going on here! We can now see the phosphorus effect is significant at S=0 and S=22.5, and almost significant at S=11.25. If we look at the line plot above, we see that phosphorus increases yield when S=0 and S=11.25, but decreases yield when S=22.5. The positive and negative effects cancelled each out when we looked at the overall analysis of variance – the interaction *masked* the sulfur effect so that its significance was not reflected in the results.

This demonstrates why it is so important to investigate interactions before making inferences about treatments. If we concluded from the first analysis of variance that sulfur affected yield, we would have been accurate. But if we had not analyzed the phosphorus effect separately by sulfur level, we would have erroneously concluded it did not affect yield.

### 7.2.2 Case Study 2

For the second case study, we are going to look at an experiment where turnips were planted at different densities in different row spacings. There were 5 densities and 4 row widths. This trial also blocked treatments, so it combines the randomized complete block and factorial designs.

#### 7.2.2.1 Linear Additive Model

The linear additive model for this trial is:

$$Y_{ijk} = B_i + D_j + S_k + DS_{jk} + BDS_{ijk}$$

In this model,  $Y_{ijk}$  is yield,  $B_i$  is block,  $D_j$  is density,  $S_k$  is row spacing,  $DS_{jk}$  is the interaction of planting density and row spacing, and  $BDS_{ijk}$  is the interaction of block, density, and row width, which is used as the residual or error

term in this model. We can see the additive effects of the factor levels and their interactions in the table below.

```

library(agridat)
turnip = mead.turnip

turnip_data = turnip %>%
  filter(!spacing %in% c(32)) %>%
  filter(density %in% c(8,20,32)) %>%
  mutate(spacing = as.factor(spacing),
         density = as.factor(density))

turnip_effects = turnip_data %>%
  mutate(mu = mean(yield)) %>%
  group_by(block) %>%
  mutate(B = mean(yield) - mu) %>%
  ungroup() %>%
  group_by(spacing) %>%
  mutate(S = mean(yield) - mu) %>%
  ungroup() %>%
  group_by(density) %>%
  mutate(D = mean(yield) - mu) %>%
  ungroup() %>%
  group_by(spacing, density) %>%
  mutate(SD = mean(yield) - (mu + B + S + D)) %>%
  ungroup() %>%
  mutate(Error = yield - (mu + B + S + D + SD)) %>%
  mutate(mu = round(mu,2),
         B = round(B,2),
         S = round(S,2),
         D = round(D,2),
         SD = round(SD,2),
         Error = round(Error, 2))

turnip_final = turnip_effects %>%
  mutate(yield = if_else(spacing==16 & density==32, yield-0.4, yield)) %>%
  mutate(yield = if_else(spacing==8 & density==32, yield+0.5, yield)) %>%
  mutate(yield = if_else(spacing==4 & density==32, yield+0.3, yield))%>%
  mutate(yield = if_else(spacing==16 & density==8, yield+3.2, yield))

#turnip_data

knitr::kable(turnip_final)

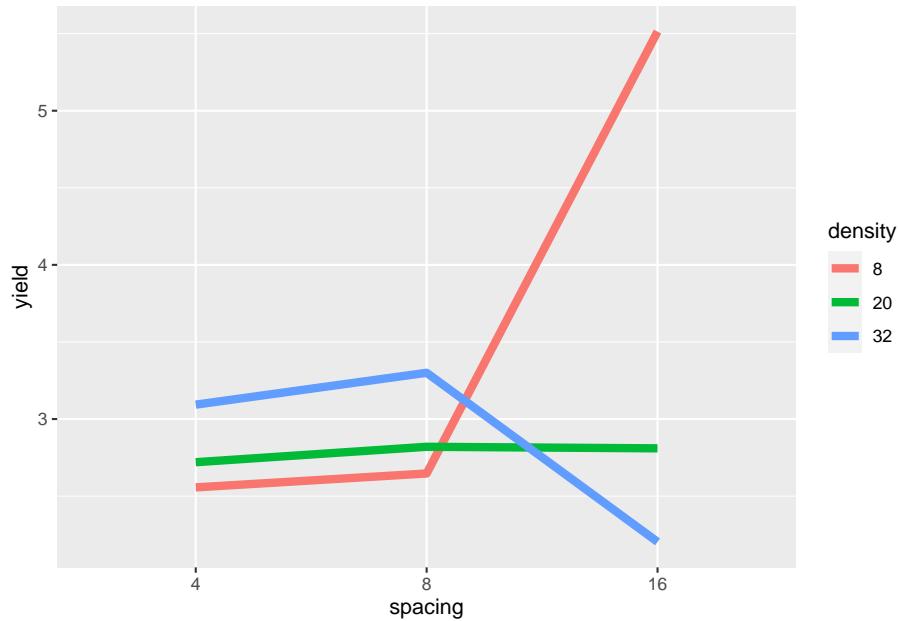
```

yield	block	spacing	density	mu	B	S	D	SD	Error
2.40	B1	4	8	2.67	-0.26	0.02	-0.17	0.29	-0.16
2.56	B1	4	20	2.67	-0.26	0.02	0.11	0.18	-0.16
2.78	B1	4	32	2.67	-0.26	0.02	0.06	0.30	-0.31
2.33	B1	8	8	2.67	-0.26	0.08	-0.17	0.32	-0.32
2.56	B1	8	20	2.67	-0.26	0.08	0.11	0.21	-0.26
3.03	B1	8	32	2.67	-0.26	0.08	0.06	0.25	-0.27
5.21	B1	16	8	2.67	-0.26	-0.10	-0.17	0.17	-0.30
2.56	B1	16	20	2.67	-0.26	-0.10	0.11	0.38	-0.25
1.90	B1	16	32	2.67	-0.26	-0.10	0.06	0.23	-0.30
2.60	B2	4	8	2.67	0.07	0.02	-0.17	-0.03	0.04
2.89	B2	4	20	2.67	0.07	0.02	0.11	-0.15	0.17
3.06	B2	4	32	2.67	0.07	0.02	0.06	-0.02	-0.03
2.63	B2	8	8	2.67	0.07	0.08	-0.17	-0.01	-0.02
2.69	B2	8	20	2.67	0.07	0.08	0.11	-0.11	-0.13
3.48	B2	8	32	2.67	0.07	0.08	0.06	-0.08	0.18
5.57	B2	16	8	2.67	0.07	-0.10	-0.17	-0.16	0.06
3.04	B2	16	20	2.67	0.07	-0.10	0.11	0.06	0.23
2.31	B2	16	32	2.67	0.07	-0.10	0.06	-0.10	0.11
2.67	B3	4	8	2.67	0.19	0.02	-0.17	-0.16	0.11
2.71	B3	4	20	2.67	0.19	0.02	0.11	-0.27	-0.01
3.44	B3	4	32	2.67	0.19	0.02	0.06	-0.15	0.35
2.98	B3	8	8	2.67	0.19	0.08	-0.17	-0.13	0.33
3.21	B3	8	20	2.67	0.19	0.08	0.11	-0.24	0.39
3.39	B3	8	32	2.67	0.19	0.08	0.06	-0.21	0.09
5.76	B3	16	8	2.67	0.19	-0.10	-0.17	-0.29	0.25
2.83	B3	16	20	2.67	0.19	-0.10	0.11	-0.07	0.02
2.40	B3	16	32	2.67	0.19	-0.10	0.06	-0.22	0.20

As we did before, we visually inspect the data for insights into how they may interact.

```
turnip_final %>%
  group_by(density, spacing) %>%
  summarise(yield = mean(yield)) %>%
  ungroup() %>%
  ggplot(aes(x=spacing, y=yield, group=density)) +
  geom_line(aes(color=density), size=2)
```

```
## `summarise()` has grouped output by 'density'. You can override using the
## `.groups` argument.
```



The plot gives critical insight into these data. Increasing spacing from 4 to 8 to 16 seems to cause a slight increase in yield where density=20. But where density=8, yield seems to increase rapidly between row spacing 8 and row spacing 16. Where density = 32, yield increases slightly with row spacing from 4 to 8, and then decreases markedly from 8 to 16.

The mean yield, averaged across row spacings, changes little across planting densities – even though the yields change dramatically within each of the individual row spacings. If we did not visually examine the data above, and instead relied on the ANOVA to alert us to an affect, we could miss this very important insight.

### 7.2.2.2 Analysis of Variance

Our analysis of variance is similar to that we ran for the first case study, except for it now includes the block term.

```
model = aov(yield ~ block + density + spacing + density:spacing, data = turnip_final)
anova_tab = anova(model)

knitr::kable(anova_tab)
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
block	2	0.9770963	0.4885481	25.06684	1.17e-05
density	2	3.3854519	1.6927259	86.85182	0.00e+00
spacing	2	2.6353852	1.3176926	67.60929	0.00e+00
density:spacing	4	16.3880593	4.0970148	210.21312	0.00e+00
Residuals	16	0.3118370	0.0194898	NA	NA

The planting density and plant spacing main effects were significant, as was their interaction.

Was the spacing effect significant at each level of density? We can slice the data to find this out.

```
library(broom)
slice_1 = turnip_final %>%
  group_by(density) %>%
  do(tidy(aov(.yield ~ .spacing))) %>%
  as.data.frame() %>%
  mutate(term = gsub("[.][$]", "", term))

knitr::kable(slice_1)
```

density	term	df	sumsq	meansq	statistic	p.value
8	spacing	2	16.9677556	8.4838778	125.0694513	0.0000129
8	Residuals	6	0.4070000	0.0678333	NA	NA
20	spacing	2	0.0182000	0.0091000	0.1341523	0.8770078
20	Residuals	6	0.4070000	0.0678333	NA	NA
32	spacing	2	2.0374889	1.0187444	12.8701572	0.0067549
32	Residuals	6	0.4749333	0.0791556	NA	NA

We can see above that the effect of spacing on yield is only significant at  $P \leq 0.05$  density=8 and density=32. If we examine the effect of density separately for each level of spacing:

```
library(broom)
slice_2 = turnip_final %>%
  group_by(spacing) %>%
  do(tidy(aov(.yield ~ .density))) %>%
  as.data.frame() %>%
  mutate(term = gsub("[.][$]", "", term))

knitr::kable(slice_2)
```

spacing	term	df	sumsq	meansq	statistic	p.value
4	density	2	0.4540667	0.2270333	4.347447	0.0680698
4	Residuals	6	0.3133333	0.0522222	NA	NA
8	density	2	0.6872889	0.3436444	3.670979	0.0909484
8	Residuals	6	0.5616667	0.0936111	NA	NA
16	density	2	18.6321556	9.3160778	135.037365	0.0000103
16	Residuals	6	0.4139333	0.0689889	NA	NA

We similarly see that density is only significant at  $P \leq 0.05$  where spacing=16.

In this trial, both main (density and spacing) effects are significant. But if don't explore and explain our analysis further, we might miss how the both the magnitude of and the rank rank of row spacing effects on yield changes with plant density.

### 7.2.3 Discussing Interactions

In factorial experiments, there are three kinds of interaction that may occur. We can have no interaction, a spreading interaction, or a crossover interaction.

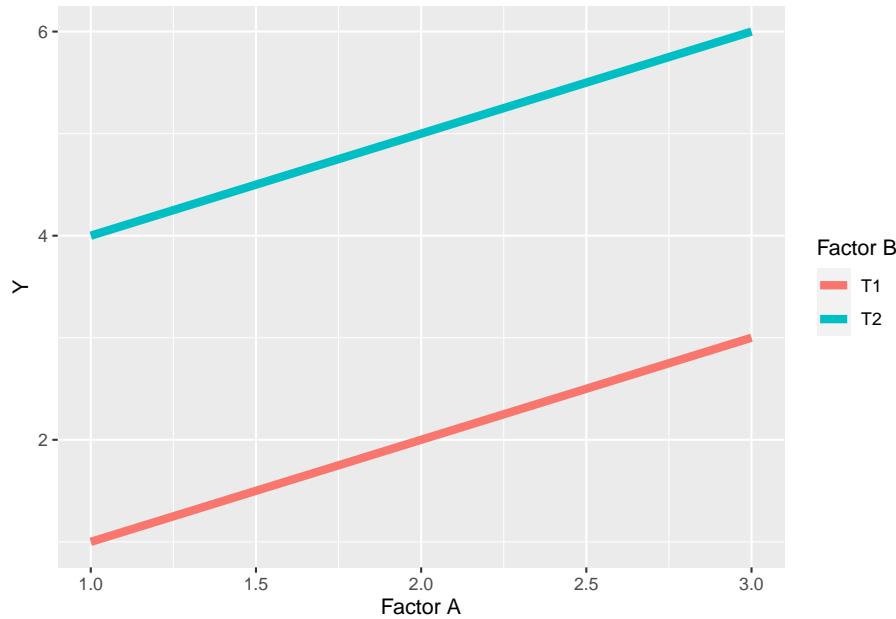
#### 7.2.3.1 No Interaction

Where there is no interaction, the treatments effects are simply additive – the observed value of each observation in the plot above is the sum of the effect of the level of Factor A plus the effect of the level of Factor B. The difference between levels of Factor A are consistent across levels of Factor B, and vice versa. If we plot the data using a line plot, it will look like this:

```
factA = rep(c(1,2,3), 2)
Y = c(1:6)
factB = rep(c("T1", "T2"), each=3)

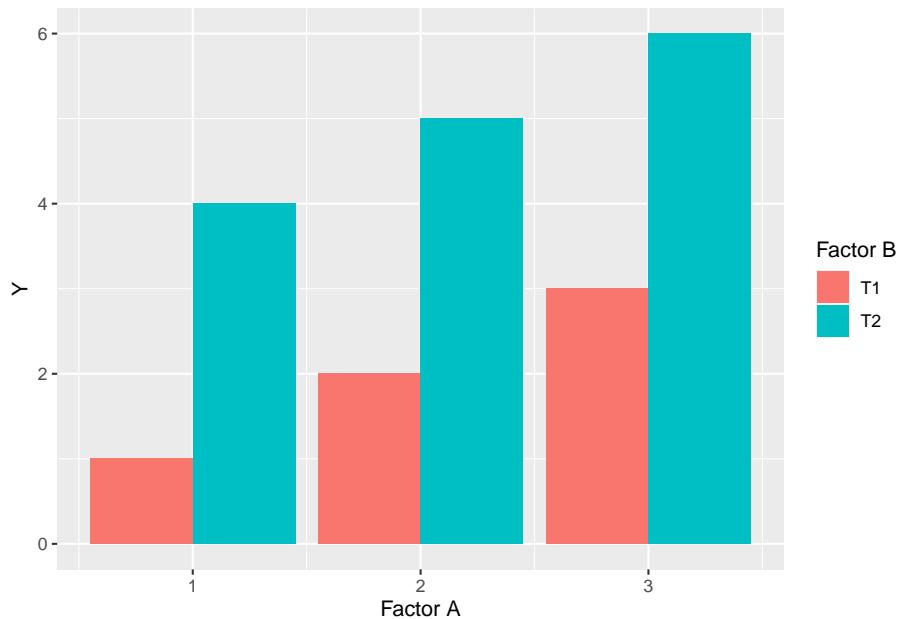
dataset = cbind(factA, Y, factB) %>%
  as.data.frame() %>%
  mutate(factA=as.numeric(factA),
        Y=as.numeric(Y))

dataset %>%
  ggplot(aes(x=factA, y=Y, group=factB)) +
  geom_line(aes(color=factB), size=2) +
  labs(x="Factor A", legend="Factor B", color="Factor B")
```



In a bar plot, it should look like this:

```
dataset %>%
  ggplot(aes(x=factA, y=Y, group=factB)) +
  geom_bar(stat="identity", aes(fill=factB), position = "dodge") +
  labs(x="Factor A", legend="Factor B", fill="Factor B")
```



Finally, there will be no change in the ranking of levels within factors. Rank is the order of levels according to their observed effects, from least to greatest. For factor A, the ranking of levels within Factor A is  $1 > 2 > 3$ , while within Factor B, level T2 always ranks higher than level T1.

### 7.2.3.2 Spreading Interaction

In a spreading interaction, the ranking of levels within factors does not change, but the difference between them does.

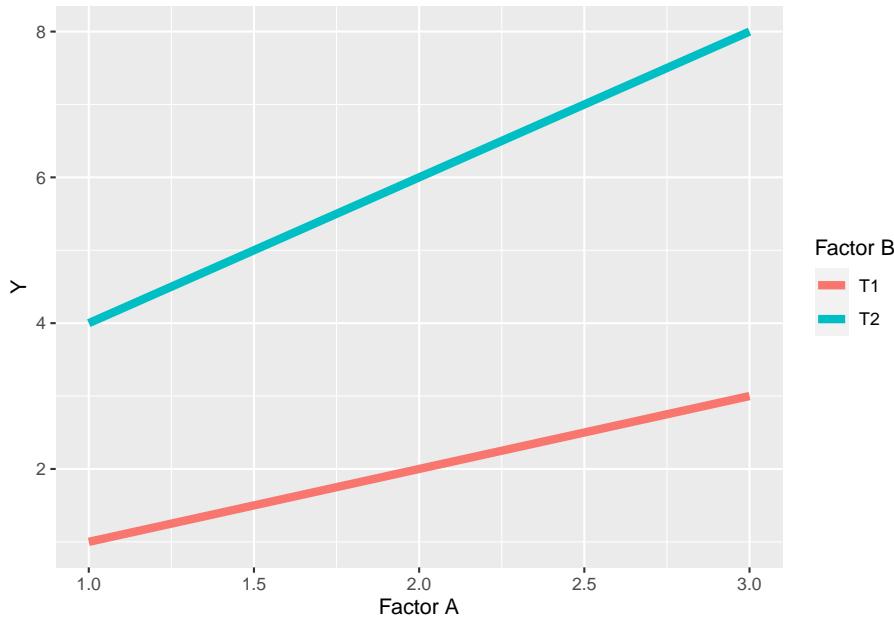
```

factA = rep(c(1,2,3), 2)
Y = c(1,2,3,4,6,8)
factB = rep(c("T1", "T2"), each=3)

dataset = cbind(factA, Y, factB) %>%
  as.data.frame() %>%
  mutate(factA=as.numeric(factA),
        Y=as.numeric(Y))

dataset %>%
  ggplot(aes(x=factA, y=Y, group=factB)) +
  geom_line(aes(color=factB), size=2) +
  labs(x="Factor A", legend="Factor B", color="Factor B")

```



In the above plot, we can see the levels of Factor A still rank  $1 < 2 < 3$  in their effect on the observed value  $Y$ , for both level T1 and level T2 of Factor B. We also note that the levels of Factor B rank  $T1 < T2$  at each level of Factor A. In this spreading interaction, however, the difference between T1 and T2 of factor B increases as the levels of Factor A increase. Similarly, the differences among levels of Factor A are greater for level T2 than level T1 of Factor B.

We saw a spreading interaction before in Case Study 1. The effect of sulfur increased with the level of phosphorus, and vice versa.

### 7.2.3.3 Crossover Interaction

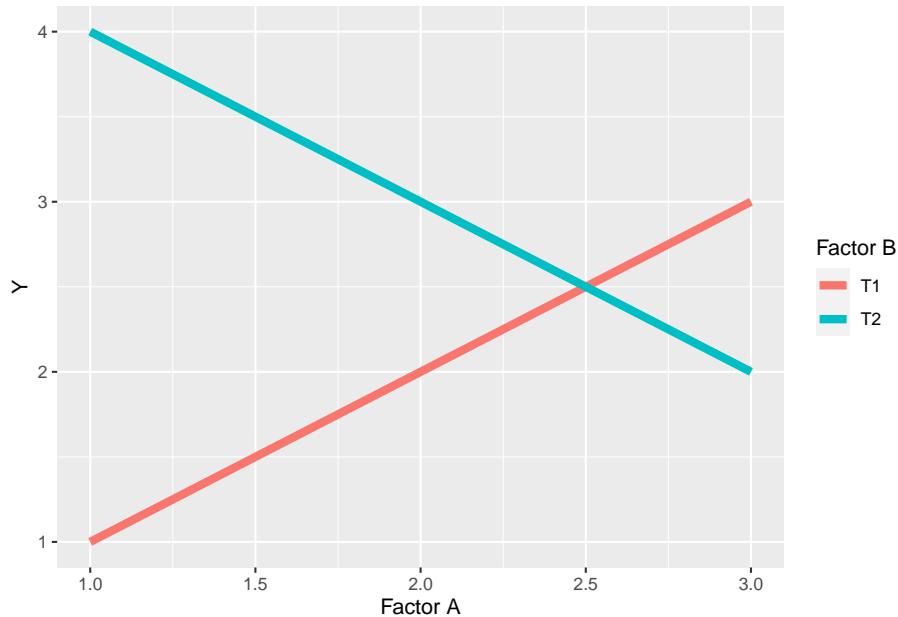
A crossover interaction is similar to a spreading interaction in that the differences between levels within one factor change with the levels of a second factor, but different in that the ranking of levels changes as well. In addition, as we saw above, crossover reactions can mask the effects of factor levels.

```

factA = rep(c(1,2,3), 2)
Y = c(1,2,3,4,3,2)
factB = rep(c("T1", "T2"), each=3)

dataset = cbind(factA, Y, factB) %>%
  as.data.frame() %>%
  mutate(factA=as.numeric(factA),
        Y=as.numeric(Y))
  
```

```
dataset %>%
  ggplot(aes(x=factA, y=Y, group=factB)) +
  geom_line(aes(color=factB), size=2) +
  labs(x="Factor A", legend="Factor B", color="Factor B")
```



In the plot above, the ranking of levels within Factor B is  $T2 > T1$  for levels 1 and 2 of Factor A, but  $T1 > T2$  for level 3 of Factor A. In other words, whether T2 is greater than T1 depends on the level of Factor A. In addition, the levels of Factor B behave differently in response to the levels of Factor A. Level T1 of Factor B increases with the level of Factor B, while level T2 decreases.

We observed a crossover reaction in Case Study 2 above, where the effect of the widest row spacing on yield was greater than the narrow row spacings at the lowest planting density, but was less than the narrow spacings at the greatest planting density.

#### 7.2.3.4 Discussing Interactions

Interactions are exciting. Spreading interactions show us how the proper combination of management practices or inputs can have a combined effect that is greater than the individual effect of inputs. Conversely, crossover interactions show us how the advantage of one practice can be lost with the mismanagement of a second practice. This is the essence of agronomy.

When we identify spreading interactions, we learn how to build more productive *cropping systems*, as opposed to one-off solutions. Don't get me wrong – there are some wonderful one-off solutions. But almost every input or practice can be made more effective by understanding how it interacts with other practices. In addition, trials can be designed to test how the effect of that input interacts with different environmental conditions, such as temperature and precipitation.

Interactions should always be highlighted when discussing the findings of an Analysis of Variance. The essence of an interaction is that the effect of one factor is *dependent* on the level of a second factor. If you don't discuss the interaction between factors, you don't completely describe the effect of either factor.

As we saw above, crossover interactions can mask the effects of factors. In the second case study, had we simply concluded from the main effect that yield did not differ with plant density, we would have failed to report what the data actually show: yield differs profoundly with plant density – but that row spacing affects the direction of that difference.

## 7.3 Split-Plot Design

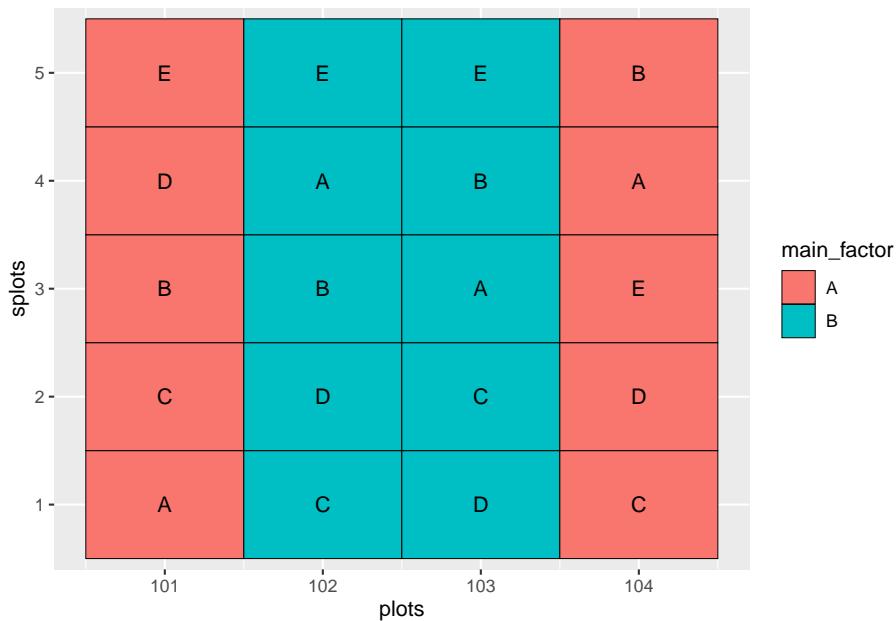
The *Split-Plot Design* is a special kind of factorial experiment, in which one factor is nested within another. Let's jump to an illustration.

```
library(agricolae)

main_factor = c("A", "B")
sub_factor = c("A", "B", "C", "D", "E")
trt_rand = design.split(main_factor,
                        sub_factor,
                        r=2,
                        design = "rcbd",
                        seed=2)

plot_map = trt_rand$book

plot_map %>%
  ggplot(aes(x=plots, y=splots), group=main_factor) +
  geom_tile(aes(fill=main_factor), color="black") +
  geom_text(aes(label=sub_factor))
```



In the experiment show above, there are two factors: the main factor and the sub-factor. The main factor has two levels, while the sub-factor has 5 levels. The design is a randomized complete block design. As with any randomized complete block design, all treatments must occur once within each block.

Main factor levels are indicated by color. Sub-factor levels are indicated by letter.

In the above plot map, the main factors occur once in Block 1 (plots 101 and 102) and Block 2 (plots 103 and 104). Within each occurrence of the main factor, all five levels of the sub-factor are nested. The levels of the main factor are randomized within each block, and the levels of the subfactor are randomized within each main plot.

So now that we know what a split-plot design looks like, we will address the question: why would we want to do this? The first answer has to do with practicality. Say the first factor is composed of fungicide treatments that we are going to apply with a Hagge applicator with a 100 ft boom. The second factor is composed of treatments (hybrid, in-furrow fertilizer, etc) that can be apply in 20-foot widths. We can apply our treatments much more easily if we use a split-plot design and nest the second factor (the sub-factor) within the first factor (the main factor).

The second answer is that a thoughtfully-designed split-plot experiment will be more sensitive to differences among levels of the sub-factor than a randomized complete block trial. This is because, no matter how hard we try, plots that are closer together are more likely to be similar than plots that are further apart.

By putting the sub-factor treatments close together, we can better estimate and test treatment effects.

The greater sensitivity to subfactor treatments, however, comes at a cost: we sacrifice some of our ability to detect differences among levels of our main factor. Sometimes, however, the levels of the main plot factor are so markedly different that we know we will detect differences among them, even if they are placed further apart. Plus, we may be more interested in the interaction between the main factor and sub-factor, and our ability to estimate and test this interaction, too, is enhanced by the split plot design.

####Case Study: Habanero Peppers If you are a chili pepper fan like me, you enjoy habanero chilies, also called “Scotch bonnets” in strict moderation. They are hotter than the jalapeno, but not devastating like the Carolina Reaper. In this study, habanero peppers were grown in pots of soils characterized by their color. They were harvested at two stages of ripening, green and orange, and their polyphenol concentrations measured. Soil was the main factor and harvest time the subfactor.

```
habanero = read.csv("data-unit-7/habanero.csv")
head(habanero)
```

```
##   block  soil harvest_stage total_polyphenols
## 1     R1    red        green      109.84958
## 2     R1    red       orange      186.71777
## 3     R1  brown        green      130.53991
## 4     R1  brown       orange      207.00041
## 5     R1  black        green       97.86705
## 6     R1  black       orange      212.19618
```

## 7.4 Linear Additive Model

$$Y_{ijk} = \mu + M_j + \text{Error}(B_i + BM_{ij}) + S_k + MS_{ik} + \text{Error}(BS_{ik} + BMS_{ijk})$$

Bear with me. We can break this model down in to two sections. Let's start with the first three terms. These focus on explaining the observed variance among the main plots.

$B_i$  is the effect of block  $i$

$M_j$  is the effect of level  $j$  of the main factor

$\text{Error}(BM_{ij})$  is the error (unexplained variance) associated with the block  $i$  and level  $j$  of the main factor. When we calculate the F-value for the main factor, we use this error term.

The second three terms focus on explaining the observed variance among sub-plots.

$S_k$  is the effect of level  $k$  of the sub-factor

$MS_{ik}$  is the interaction between level  $i$  of the main factor and level  $k$  of the subfactor

$Error(BS_{ik} + BMS_{ijk})$  is the unexplained variance associated with the given levels of the sub factor and the main factor - sub-factor interaction. It is used in testing the significance of both those effects

Ugh. What a dumpster fire. Let's look at the ANOVA table for our habanero trial to make more sense of this.

```
habanero_model = aov(total_polyphenols ~ soil + harvest_stage + soil:harvest_stage + Error(block:soil))

## Warning in aov(total_polyphenols ~ soil + harvest_stage + soil:harvest_stage + :
## Error() model is singular

summary(habanero_model)

##
## Error: block:soil
##           Df Sum Sq Mean Sq F value    Pr(>F)
## soil          2 1322.7   661.3   8.691 0.00791 ***
## Residuals    9  684.9    76.1
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Error: Within
##           Df Sum Sq Mean Sq F value    Pr(>F)
## harvest_stage     1 42120   42120  176.72 3.2e-07 ***
## soil:harvest_stage 2  5938    2969   12.46 0.00256 **
## Residuals        9  2145     238
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

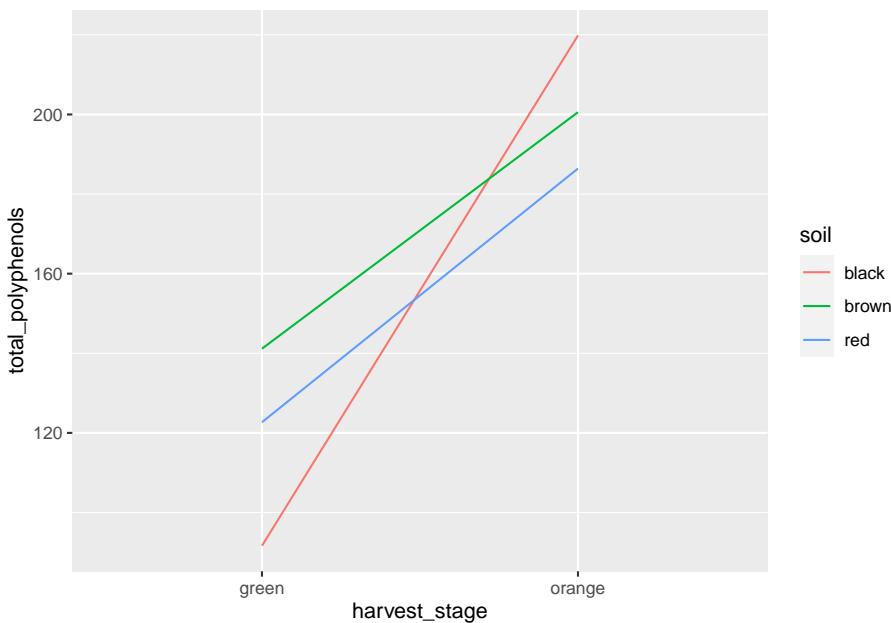
We have two ANOVA tables. The top table evaluates the effect of soil in our main plots. Looking at the degrees of freedom, we can see we have  $3 \text{ soils} - 1 = 2$  degrees of freedom for soil and  $(4 \text{ blocks} - 1) \times 3 = 9$  degrees of freedom for the error. The F-value is the ratio of the mean square for soil and the mean square for the main factor error (residuals).

The second table evaluates the effect of the harvest stage and the interaction between soil and harvest stage in the subplots. There are  $2 \text{ stages} - 1 = 1$  degree of freedom for harvest stage and  $*(3 \text{ soils} - 1) \times (2 \text{ stages} - 1) = 2$  degrees of freedom for the interaction. The F-values for the harvest stage and interaction effects are the ratios of their mean squares to the mean square for the subplot error (residuals).

We analyze the ANOVA results the same as we would for any factorial trial. Looking at our table, we see significant results for soil, harvest stage, and their interaction. So lets go straight to the interaction plot.

```
habanero %>%
  group_by(soil, harvest_stage) %>%
  summarise(total_polyphenols = mean(total_polyphenols)) %>%
  ungroup() %>%
  ggplot(aes(x=harvest_stage, y=total_polyphenols, group = soil)) +
  geom_line(aes(color=soil))

## `summarise()` has grouped output by 'soil'. You can override using the
## `.` argument.
```



We can see the the total polyphenol concentration was affected by harvest stage in each soil. Orange habaneros had greater total polyphenols than green ones. Furthermore, we can see that black soils produced the fewest total polyphenols when habaneros where harvested green, but the most when they were harvested orange.

Looking at our interaction tables, we see soil has a significant effect on total polyphenols at both harvest stages.

```
library(broom)
slice_1 = habanero %>%
  group_by(harvest_stage) %>%
  do(tidy(aov(.total_polyphenols ~ .block + .soil))) %>%
  as.data.frame() %>%
  mutate(term = gsub("[.] [\$]", "", term))

knitr::kable(slice_1)
```

harvest_stage	term	df	sumsq	meansq	statistic	p.value
green	block	3	312.1119	104.03731	0.4324894	0.7375426
green	soil	2	5006.9642	2503.48208	10.4071271	0.0112036
green	Residuals	6	1443.3275	240.55458	NA	NA
orange	block	3	79.2278	26.40927	0.1592069	0.9199513
orange	soil	2	2253.3551	1126.67757	6.7921174	0.0287564
orange	Residuals	6	995.2810	165.88017	NA	NA

And that harvest stage had a significant effect at each level of soil.

```
library(broom)
slice_2 = habanero %>%
  group_by(soil) %>%
  do(tidy(aov(.total_polyphenols ~ .block + .harvest_stage))) %>%
  as.data.frame() %>%
  mutate(term = gsub("[.] [\$]", "", term))

knitr::kable(slice_2)
```

soil	term	df	sumsq	meansq	statistic	p.value
black	block	3	79.03859	26.34620	0.0555253	0.9798439
black	harvest_stage	1	32872.72483	32872.72483	69.2801410	0.0036344
black	Residuals	3	1423.46960	474.48987	NA	NA
brown	block	3	214.87947	71.62649	0.5937286	0.6605111
brown	harvest_stage	1	7062.17490	7062.17490	58.5400106	0.0046367
brown	Residuals	3	361.91529	120.63843	NA	NA
red	block	3	390.95481	130.31827	1.0869202	0.4734925
red	harvest_stage	1	8122.82365	8122.82365	67.7484520	0.0037542
red	Residuals	3	359.69045	119.89682	NA	NA

## 7.5 Conclusion

Experimental design of multiple treatment experiments plays a critical role in the quality of our data and the inferences that can be made. In most cases, you

will probably run randomized complete block design experiments with a single factor, where blocking will reduce the error among plots within each block.

The factorial design allows greater efficiency if you are interested in studying two factors. It is also the only way to study the interaction between two factors.

Finally, where one factor is known to be more subtle (and therefore, harder to test significance for) than a second factor, or where differences in equipment size make it more practical to nest one factor inside the other, the split-plot design can be very useful.

## 7.6 Exercise: Randomized Complete Block Design

While the lecture is long this week, fortunately running the analyses in R is very fast. We start the exercises with the Randomized Complete Block Design. To run a Randomized Complete Block Design, we will again use same analysis of variance approach we did last week. We only need two lines of code: the first to define the model we wish to test, and the second to tell R to summarise the results.

### 7.6.1 Case Study

Barley varieties were compared using a Randomized Complete Block Design. Yield is in grams per plot. Five varieties were grown in three replicates.

```
library(tidyverse)
barley = read.csv("data-unit-7/exercise_data/barley_besag.csv")
head(barley)

##   block gen yield
## 1      1 G37 10.92
## 2      1 G55 12.07
## 3      1 G58  9.51
## 4      1 G63  9.23
## 5      1 G09  8.03
## 6      2 G37 10.24
```

### 7.6.2 ANOVA

For this analysis of variance, we need two lines of code. The first is the model statement:

```
barley_model = aov(yield ~ block + gen, data=barley)
```

In the above code:

- barley\_model defines the object to which the ANOVA results will be saved
- “aov” tells R to run an analysis of variance
- “yield ~ block + gen” tells R our linear additive model. When using R, we leave out mu and the error term – R knows to calculate these.
- “data=barley” tells R to fit our ANOVA model to the barley dataset

Reviewing our results is simple:

```
summary(barley_model)
```

```
##          Df Sum Sq Mean Sq F value    Pr(>F)
## block      1  2.611   2.611   7.476 0.023066 *
## gen        4 19.758   4.940  14.142 0.000642 ***
## Residuals  9  3.144   0.349
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In our result above, we see the effect of barley variety (gen) is highly significant.

### 7.6.3 Plotting the Results

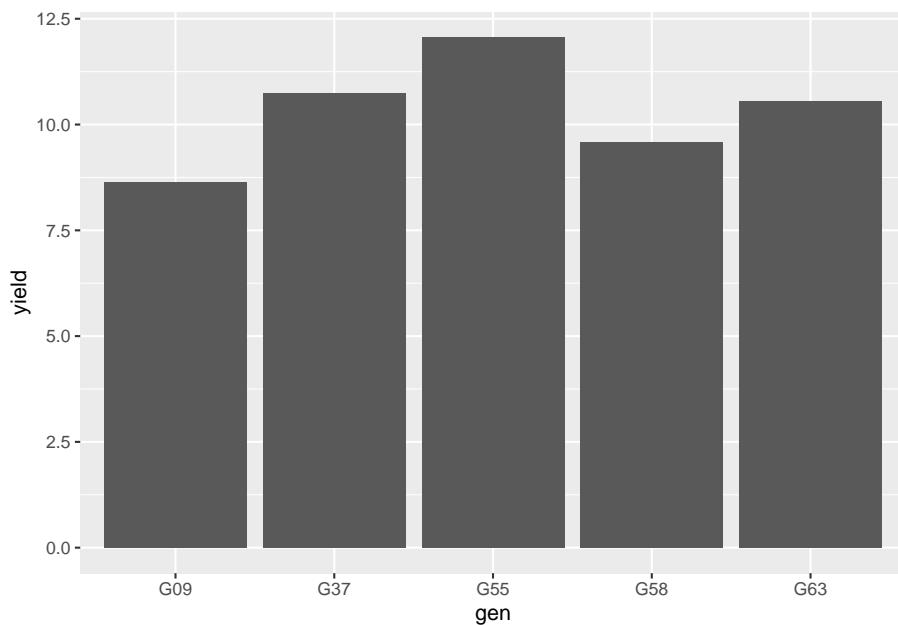
Although we get into the presentation of treatment means in more detail later in the course, let's take a quick look at our treatment means. We will use ggplot.

```
barley_plot = ggplot(barley, aes(x=gen, y=yield)) +
  geom_bar(stat = "summary", fun="mean")
```

In the plot above: \* barley is the object to which we are saving the plot \* ggplot() starts the ggplot function \* “ggplot(barley, aes(x=gen, y=yield))” tells R that our plot will be based on the barley dataset. Anything in ggplot that occurs in aes() brackets is an “aesthetic”. An aesthetic is any value that is based on a variable from our plot and changes in value from treatment to treatment. x=gen tells R the horizontal position of whatever we draw (points, bars, etc.) is determined by the gen variable. y=yield tells R the vertical position (height) of our object is based on the yield variable. \* “geom\_bar” tells R we want to draw bars with our data. \* With geom\_bar, we can also calculate summary statistics on the fly. stat=“summary” tells R we are going to calculate a summary statistic. fun=“mean” tells R to use the mean as that statistic.

Now we can display the plot at any time by running barley\_plot:

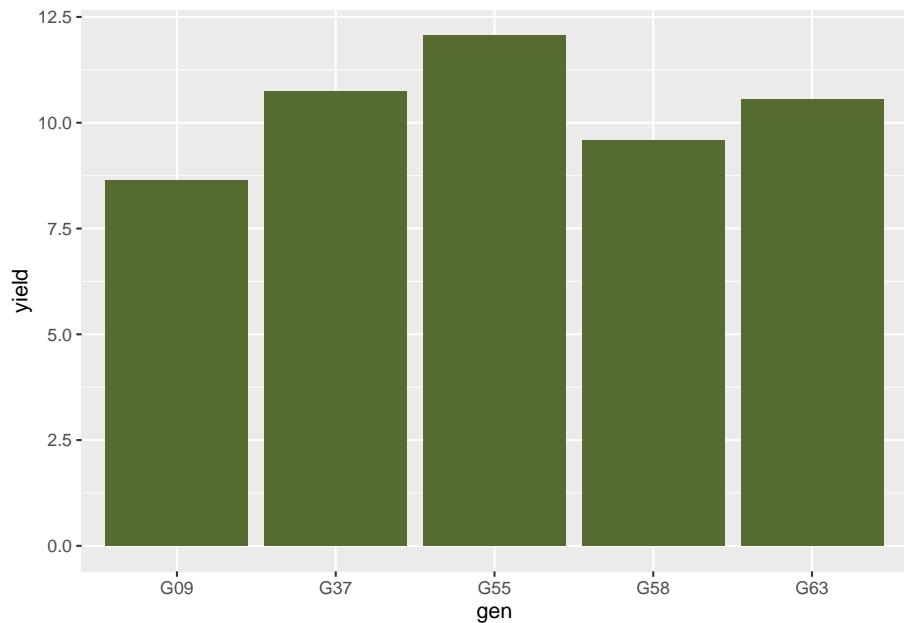
```
barley_plot
```



This is so drab! While I don't want you to get bogged down in formatting plots, let's tweak a few details just so you can see the power of ggplot to make publication-quality plots.

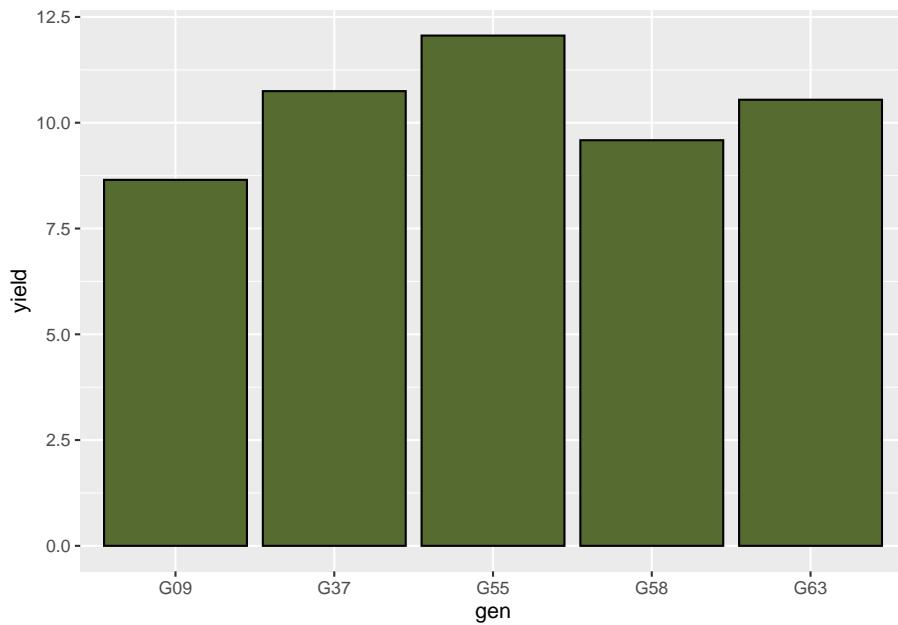
Let's change the bar color by adding the "fill" command to our code:

```
barley_plot = ggplot(barley, aes(x=gen, y=yield)) +  
  geom_bar(stat = "summary", fun="mean", fill="darkolivegreen")  
  
barley_plot
```



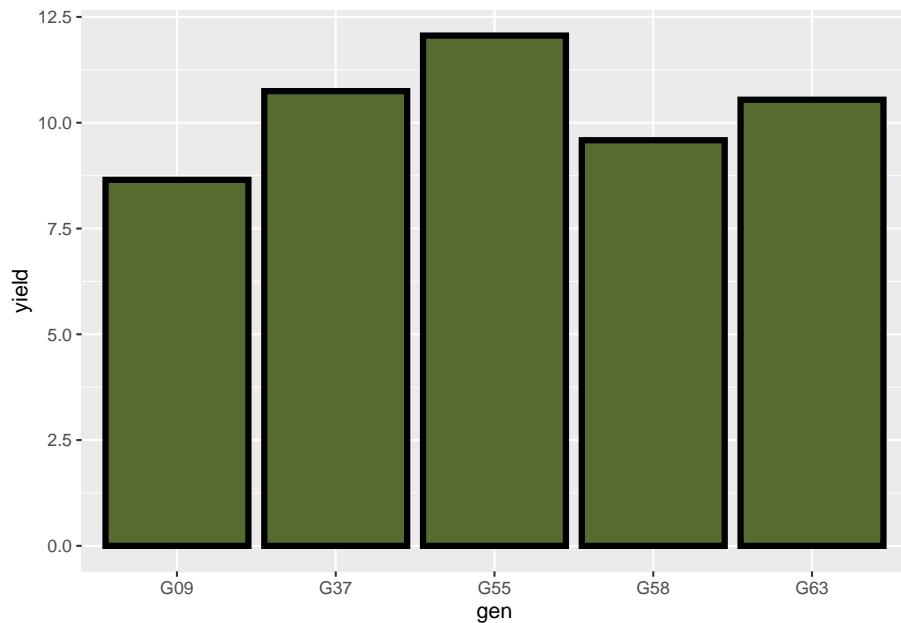
Let's draw a black line around the outside of the bars by adding "color" to our `geom_bar` statement.

```
barley_plot = ggplot(barley, aes(x=gen, y=yield)) +  
  geom_bar(stat = "summary", fun="mean", fill="darkolivegreen", color="black")  
barley_plot
```



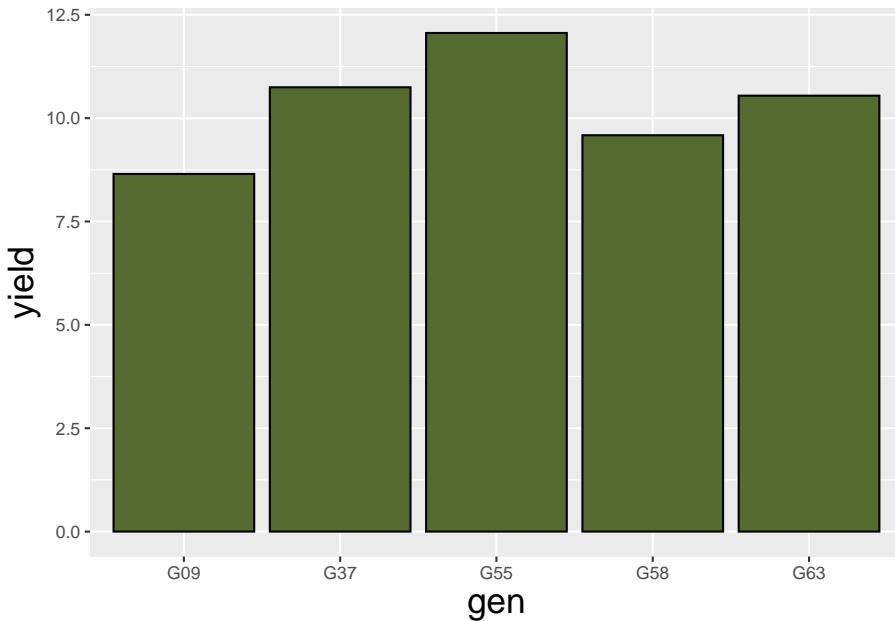
Let's make the outline a little thicker using the size command:

```
barley_plot = ggplot(barley, aes(x=gen, y=yield)) +  
  geom_bar(stat = "summary", fun="mean", fill="darkolivegreen", color="black", size=1.5)  
barley_plot
```



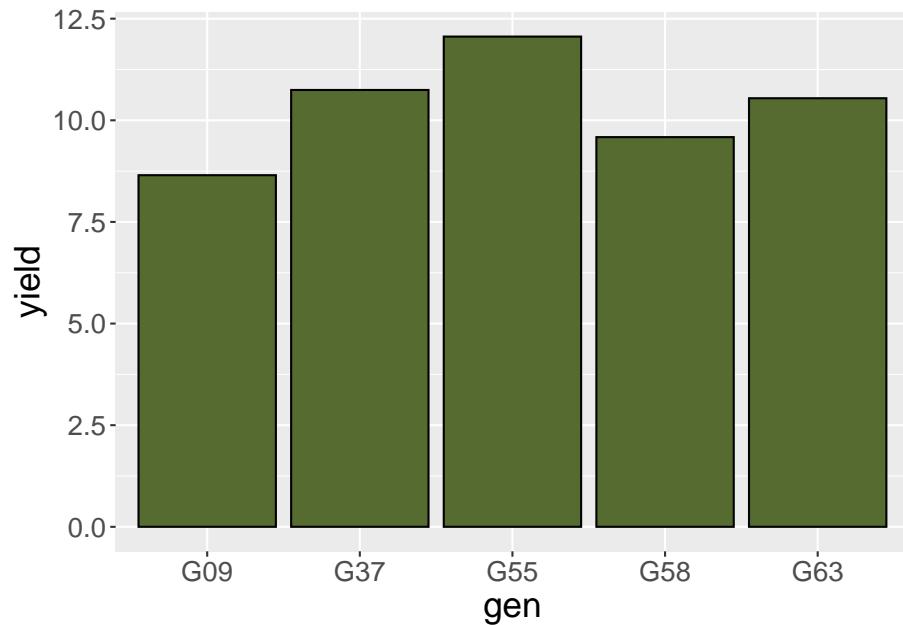
The axis titles are little hard to read. We can change text formatting, legend formatting, and other objects using the “theme” function. In the theme function below, let’s change the “axis.title” size using the “element\_text” command and setting the size to 18.

```
barley_plot = ggplot(barley, aes(x=gen, y=yield)) +  
  geom_bar(stat = "summary", fun="mean", fill="darkolivegreen", color="black") +  
  theme(axis.title = element_text(size=18))  
barley_plot
```



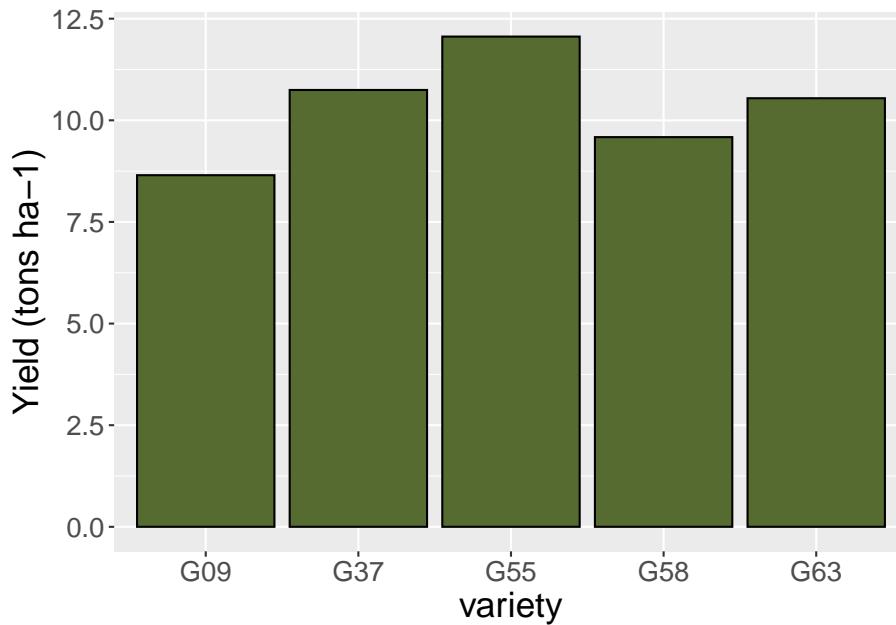
Let's similarly increase the size of the axis.text (tick labels) to 14pt.

```
barley_plot = ggplot(barley, aes(x=gen, y=yield)) +
  geom_bar(stat = "summary", fun="mean", fill="darkolivegreen", color="black") +
  theme(axis.title = element_text(size=18),
        axis.text = element_text(size=14))
barley_plot
```



Finally, let's rename our axis titles using the `lab()` command.

```
barley_plot = ggplot(barley, aes(x=gen, y=yield)) +  
  geom_bar(stat = "summary", fun="mean", fill="darkolivegreen", color="black") +  
  theme(axis.title = element_text(size=18),  
        axis.text = element_text(size=14)) +  
  labs(x="variety", y="Yield (tons ha-1)")  
barley_plot
```



Like I said, I don't want you to get too bogged down in formatting your plots. And, yes, a simple plot like this could be made just as quickly in Excel. But building the plot in R Studio has multiple advantages. First, you don't have to cut and paste your data to Excel. Second, you don't have to copy your completed chart back to R or your report document. You can include it right in your R Notebook with your data. That way, if you are working on your creative component, you can have one document, instead of multiple documents all over the place. Third, if you are working on the multiple plots, you can coordinate them by reusing your theme code from plot to plot, rather than having to click and change each field in each plot. Finally, and most importantly, you have way more options for creating your plot in R than just about any software I know!

#### 7.6.4 Practice

Navy bean varieties were evaluated in a randomized complete block trial .

```
beans = read.csv("data-unit-7/exercise_data/besag_beans.csv")
head(beans)

##   block     gen yield
## 1    R1   Maris   350
## 2    R1   Dwarf   230
## 3    R1  Minica   355
## 4    R1   Stella   370
```

```
## 5    R1 Topless    280
## 6    R1 Metissa   185
```

Run the analysis in R. Your output should match the following:

```
Df Sum Sq Mean Sq F value Pr(>F)
block 23 294256 12794 2.324 0.00182 ** gen 5 719695 143939 26.147 < 2e-16
*** Residuals 115 633063 5505
Signif. codes: 0 ‘‘ 0.001 ‘‘ 0.01 ‘‘ 0.05 ‘‘ 0.1 ‘‘ 1
```

What do you conclude from the analysis of variance?

## 7.7 Exercise: Factorial ANOVA

For the second exercise this week, we will work with factorial experimental designs. Remember, a factorial design has two or more factors; within each factor are two or more levels, which we commonly refer to as treatments. For a factorial design, our analysis becomes a little more complex, but not terribly so. Factorial experiments may have Completely Randomized Designs or Randomized Complete Block Designs. The latter is more common.

### 7.7.1 Case Study: Biochar

For our case study, we will use a dataset inspired from a recent paper:

Ahmad, M.; Wang, X.; Hilger, T.H.; Luqman, M.; Nazli, F.; Hussain, A.; Zahir, Z.A.; Latif, M.; Saeed, Q.; Malik, H.A.; Mustafa, A. Evaluating Biochar-Microbe Synergies for Improved Growth, Yield of Maize, and Post-Harvest Soil Characteristics in a Semi-Arid Climate. *Agronomy* 2020, 10, 1055.

This trial, in Pakistan, looked at the factorial effects of biochar soil amendments and seed inoculum on maize growth. I am not an expert on biochar – it is basically an amendment made from burned crop residues – a charcoal of sorts. More information can be found in the paper above, which is open-source.

```
library(tidyverse)
biochar = read.csv("data-unit-7/exercise_data/biochar_inoculum.csv")
head(biochar)
```

##	Block	Innoculation	Amendment	Yield
## 1	B1	Inoculated	Control	7.426576
## 2	B1	Inoculated Egyptian.acacia.biochar..0.1..		6.848225

```
## 3   B1   Inoculated Egyptian.acacia.biochar..0.2... 8.130492
## 4   B1   Inoculated                      FYM.biochar..0.1... 6.349240
## 5   B1   Inoculated                      FYM.biochar..0.2... 8.463009
## 6   B1   Inoculated      Wheat.straw.biochar..0.1... 7.647091
```

I tend to skip over the treatments, so let's discuss them in greater detail. There are 2 levels of the Innoculation factor: inoculated, or not Innoculated. There are 7 levels of the factor biochar: one control plus 6 biochar treatments of different sources and rates. There are 6 blocks. Yield is reported in tons per hectare.

### 7.7.2 ANOVA

We use the same two lines of code we have used for other analyses of variance. First, our model statement. Before we enter that, let's review the linear additive model for this factorial experiment:

$$\text{Yield} = \mu + \text{Block} + \text{Innocation} + \text{Amendment} + \text{Innocation} * \text{Amendment} + \text{Error}$$

This includes the effect of Block (which is assumed, by nature of the design, to not interact with the Innocation or Amendment factors), Innocation, Amendment, and their interaction, Innocation\*Amendment.

Having identified the linear additive model, we can write out model statement in R.

```
biochar_model = aov(Yield ~ Block + Innocation + Amendment + Innocation:Amendment, data = biochar)
```

Just the same as for other trials, the statement above defines an object, "biochar" model, and assigns it the output of an analysis of variance (aov), based on our linear model and the dataset "biochar".

We can look at the ANOVA table by using the *summary()* function.

```
summary(biochar_model)
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)						
## Block	5	26.98	5.396	8.782	2.09e-06 ***						
## Innocation	1	4.24	4.244	6.907	0.010703 *						
## Amendment	6	19.10	3.183	5.181	0.000208 ***						
## Innocation:Amendment	6	4.44	0.740	1.205	0.315335						
## Residuals	65	39.94	0.614								
## ---											
## Signif. codes:	0	'***'	0.001	'**'	0.01	'*'	0.05	'.'	0.1	' '	1

Looking first at our main effects, we see the effects of both factors, Innoculation and Amendment, are significant at the P=0.05 level. The interaction is not significant.

### 7.7.3 Interaction Plots

As we learned in the lecture, it is important when analyzing factorial design experiments to plot the interactions between the data. Although line plots are normally used in the final presentation of qualitative factors like products and practices, they are useful to visualize the nature of interactions.

The first step in creating our plot is to create a dataset of means. To do this, we need to group our data using the `group_by()` function, calculate the means for each combination of Innoculation and Amendment, and `ungroup()` the data. We can link these functions with “%>%”

```
grouped_data = biochar %>%      # this feeds the biochar dataset to the next line
  group_by(Innoculation, Amendment) %>%      # group by the interactions of Innoculation
  summarise(Yield = mean(Yield)) %>%      # calculate the means for the interactions
  ungroup()      # ungroup the data for further processing

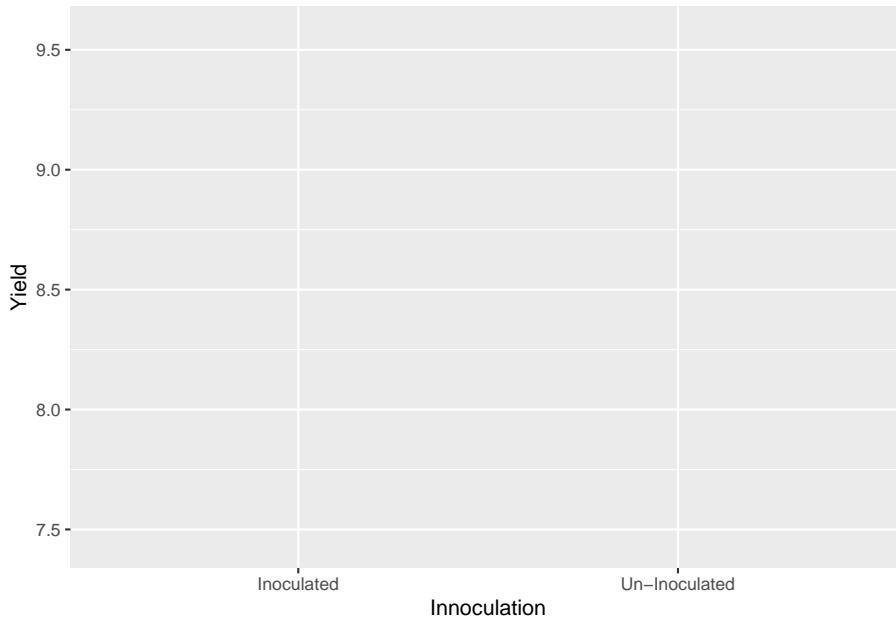
## `summarise()` has grouped output by 'Innoculation'. You can override using the
## `.groups` argument.

grouped_data

## # A tibble: 14 x 3
##   Innoculation Amendment     Yield
##   <chr>        <chr>     <dbl>
## 1 Inoculated   Control       8.38
## 2 Inoculated   Egyptian.acacia.biochar..0.1.. 7.84
## 3 Inoculated   Egyptian.acacia.biochar..0.2..  8.05
## 4 Inoculated   FYM.biochar..0.1..    8.33
## 5 Inoculated   FYM.biochar..0.2..    9.19
## 6 Inoculated   Wheat.straw.biochar..0.1..  8.75
## 7 Inoculated   Wheat.straw.biochar..0.2..  9.57
## 8 Un-Inoculated Control       7.45
## 9 Un-Inoculated Egyptian.acacia.biochar..0.1.. 7.54
## 10 Un-Inoculated Egyptian.acacia.biochar..0.2.. 8.13
## 11 Un-Inoculated FYM.biochar..0.1..    8.16
## 12 Un-Inoculated FYM.biochar..0.2..    8.20
## 13 Un-Inoculated Wheat.straw.biochar..0.1..  8.86
## 14 Un-Inoculated Wheat.straw.biochar..0.2..  8.62
```

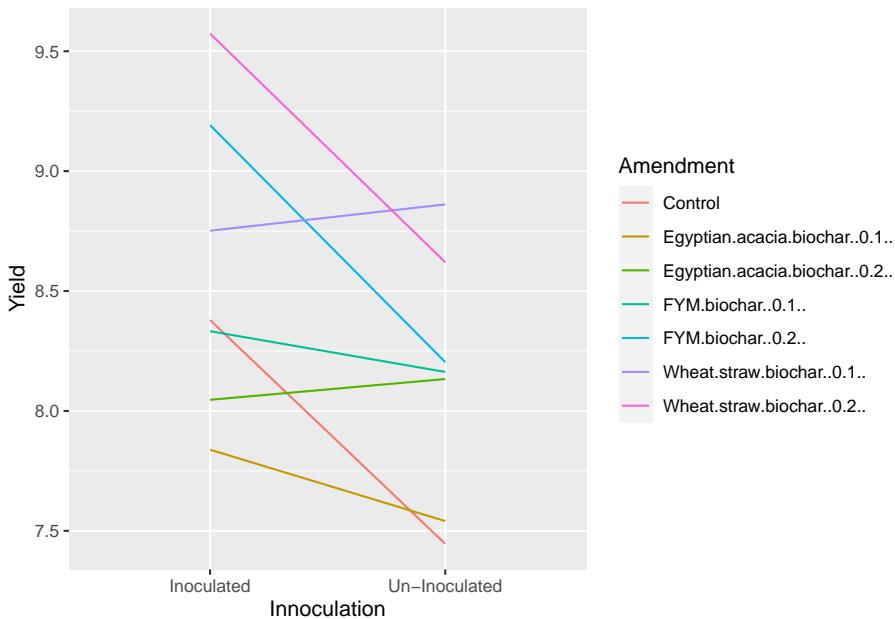
We can now feed our grouped data into ggplot to create our line plot. The `aes()` argument tells R that the position of the points connected by our lines will be determined by their level of the Innoculation factor and their Yield. It further tells R that lines should be drawn between points with the same level of the Amendment factor.

```
grouped_data %>%
  ggplot(aes(x=Innoculation, y=Yield, group=Amendment))
```



We need to add one more line telling R what shapes (geometries) to draw. We add the `geom_line` argument. In it we include an `aes()` argument to associate line color to with level of Innoculation.

```
grouped_data %>%
  ggplot(aes(x=Innoculation, y=Yield, group=Amendment)) +
  geom_line(aes(color=Amendment))
```



Our plot above reflects our ANOVA results. Maize yield with most amendments decreased when the crop was un-innoculated. The maize response was flat or even slightly increased for a few levels of Amendment, but the interaction only had a P-value of about 0.32, so the interaction was not significant. Similarly, the ranking of Amendments changed somewhat with level of Innoculation, but not enough to be significant.

#### 7.7.4 Testing Factors Individually

Sometimes we may want to know whether one factor has a significant effect at each level of the other factor. To do this requires three lines of code. The last is a little tricky, but we will get there.

The first couple of lines should look familiar. We take the biochar data.frame, and we use group\_by to group it by Innoculation level.

The third line requires further explation. What we are doing is telling R to run a separate analysis of variance for Amendment at each level of Innoculation.

```
*aov(.Yield .Block + .Amendment)) is our anova model, but it looks different than the ones we have specified.
*aov(Yield Block+Amendment, data = biochar)*. We can't do that there, though, because the model is using
tells R that each term in the model is a column from the data that is being fed
to it.
```

*tidy* tells R to return the anova as a data.frame() This allows us to stack the ANOVAs for different levels of Innoculant on top of each other. *tidy* is part

of the *broom* package, so we use *library(broom)* to load that package before we begin.

Finally, *do()* is the magical command that tells R to, well, do something – anything – with the groups of data that are fed to it.

So when we put it all together, the third line tells R to *do* something with the two levels of Innoculant, specifically to create *tidy* data.frames of the analyses of variance for both of them.

```
library(broom)

biochar %>%
  group_by(Innoculation) %>%
  do(tidy(aov(.Yield ~ .Block + .Amendment)))

## # A tibble: 6 x 7
## # Groups:   Innoculation [2]
##   Innoculation term      df    sumsq   meansq statistic   p.value
##   <chr>        <chr>     <dbl>  <dbl>    <dbl>     <dbl>    <dbl>
## 1 Inoculated   .Block     5 22.1    4.42     11.6    0.00000272
## 2 Inoculated   .Amendment 6 14.0    2.33     6.10    0.000291
## 3 Inoculated   Residuals 30 11.5    0.382    NA      NA
## 4 Un-Inoculated .Block     5 13.3    2.67     4.00    0.00669
## 5 Un-Inoculated .Amendment 6 9.58    1.60     2.39    0.0521
## 6 Un-Inoculated Residuals 30 20.0    0.667    NA      NA
```

We can see from the output that the effect of Amendment is significant at both levels of Innoculant.

What about the other way around? We simply take the code above and reverse the positions of Amendment and Innoculation.

```
biochar %>%
  group_by(Amendment) %>%
  do(tidy(aov(.Yield ~ .Block + .Innoculation)))

## # A tibble: 21 x 7
## # Groups:   Amendment [7]
##   Amendment term      df    sumsq   meansq statistic p.value
##   <chr>        <chr>     <dbl>  <dbl>    <dbl>     <dbl>    <dbl>
## 1 Control     .Block     5  2.26    0.452     1.27    0.401
## 2 Control     .Inno~     1  2.61    2.61     7.32    0.0425
## 3 Control     Residu~    5  1.78    0.357    NA      NA
## 4 Egyptian.acacia.biochar..0.1.. .Block     5  4.01    0.801     1.68    0.292
## 5 Egyptian.acacia.biochar..0.1.. .Inno~     1  0.266   0.266     0.556    0.489
```

```

## 6 Egyptian.acacia.biochar..0.1.. Residu~      5  2.39   0.478    NA     NA
## 7 Egyptian.acacia.biochar..0.2.. .$Block      5  4.67   0.934   0.779   0.604
## 8 Egyptian.acacia.biochar..0.2.. .$Inno~      1  0.0225  0.0225   0.0188  0.896
## 9 Egyptian.acacia.biochar..0.2.. Residu~      5  5.99   1.20    NA     NA
## 10 FYM.biochar..0.1..             .$Block      5 12.3    2.47    3.30   0.108
## # ... with 11 more rows

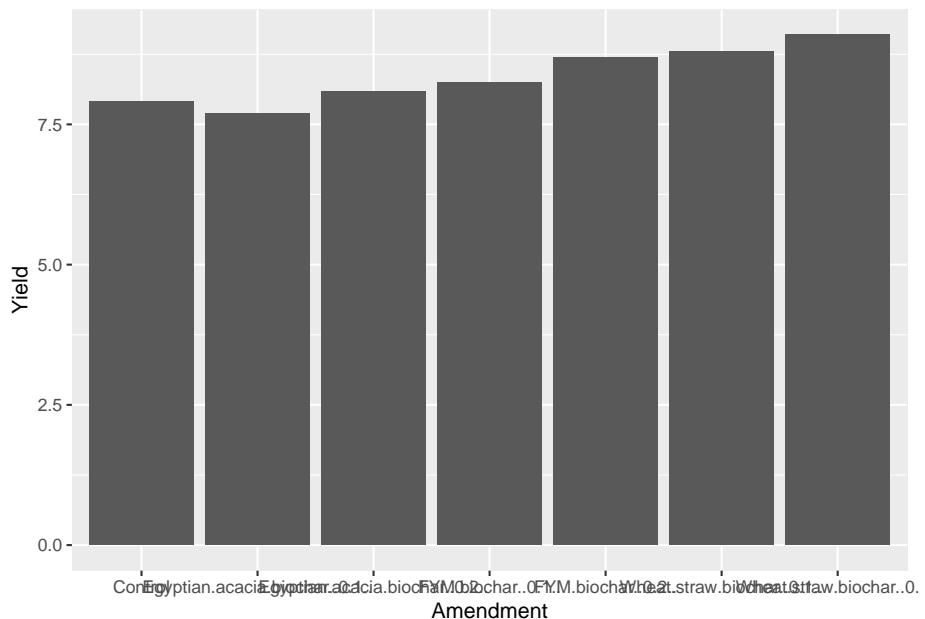
```

We see the effect of Inoculant on yield is only significant for the Control and Wheat Straw Biochar 2X rate. It is insignificant with all other levels of Amendment.

### 7.7.5 Bar Plots

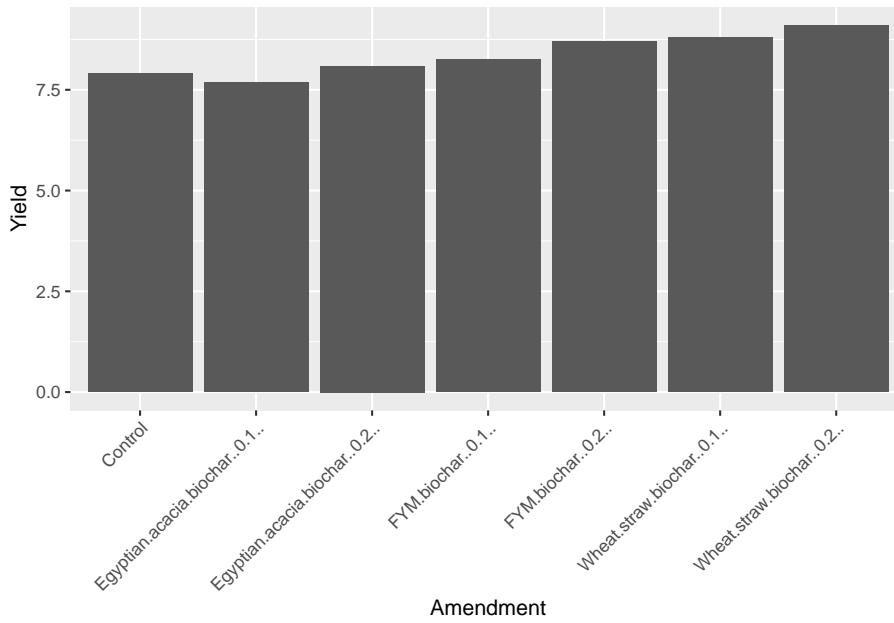
Similar to the other exercise, we can also create a bar plot of the treatment means. We have two factors, however, so we will need to tweak our code. The first thing to do is to pick one of the factors to be plotted along the X axis. Lets go with the Amendment factor. We can create a simple bar graph for our Amendment treatments

```
ggplot(data=biochar, aes(x=Amendment, y=Yield)) +
  geom_bar(stat="summary", fun="mean")
```



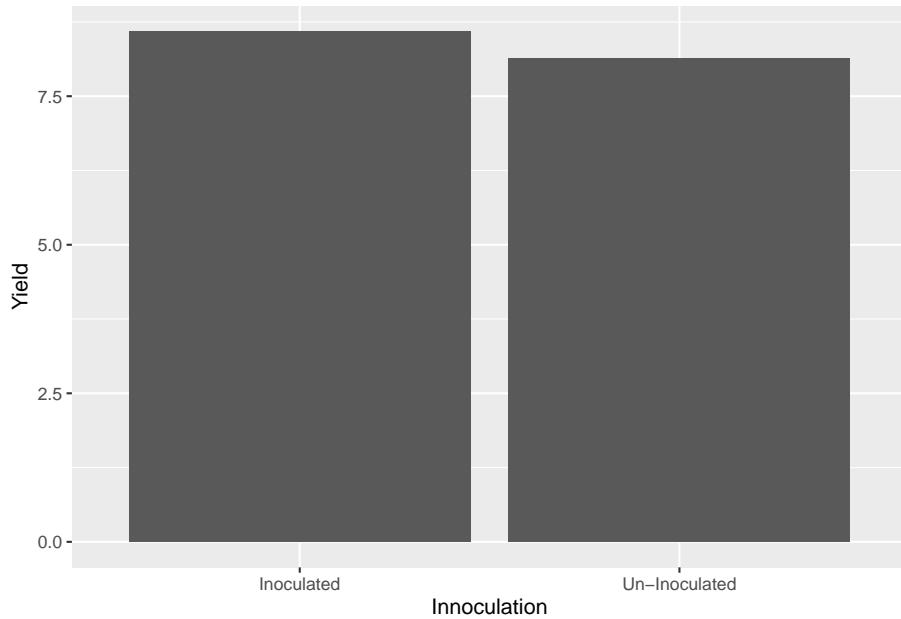
The treatment names are running together a little, so lets add a theme argument with “angle =” to pitch those at a 45 degree angle to the axis and “hjust=1) to right-justify them from the axis.

```
ggplot(data=biochar, aes(x=Amendment, y=Yield)) +  
  geom_bar(stat="summary", fun="mean") +  
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



We can plot the inoculum effect the same way, just substituting “Innoculation” for “Amendment” in our code above.

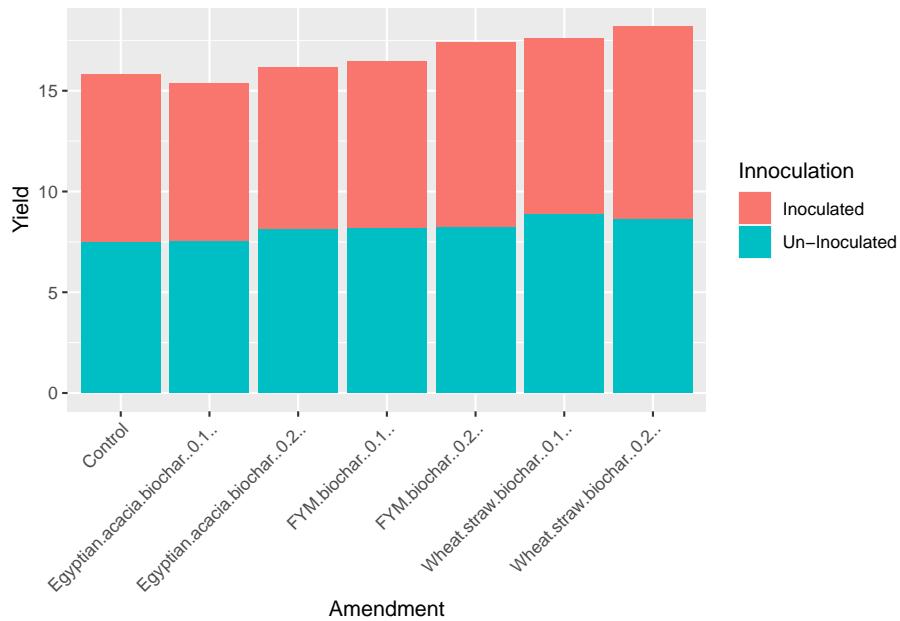
```
ggplot(data=biochar, aes(x=Innocation, y=Yield)) +  
  geom_bar(stat="summary", fun="mean")
```



Of course, in a factorial experiment, we may want to see both treatments together. In the plot below, we will plot Amendment along the X axis, Yield on the Y axis, and group the bars by level of Innoculation. Our plot starts out similar to the Amendment plot above. But in the first line we need to add the argument “group=Innoculation”. This tells R we will group our bars by the level of inoculation.

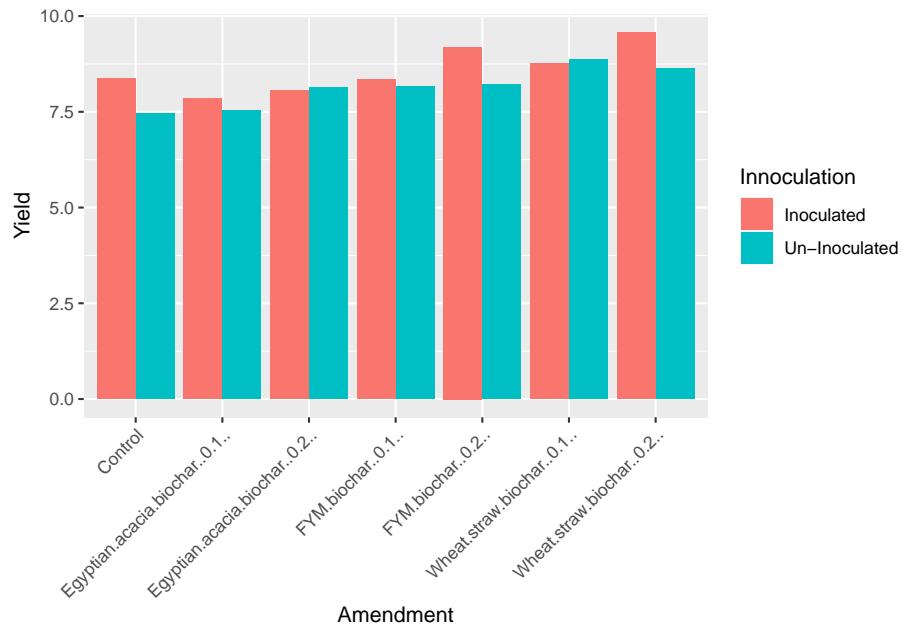
To our second line, we need to add an aesthetics argument (aes). Why? Because we want the color of the bar to change with the valuable of a variable, in this case the level of inoculation. Our complete argument reads “aes(fill = Innoculation)”

```
ggplot(data=biochar, aes(x=Amendment, y=Yield, group=Innoculation)) + # add group statement
  geom_bar(stat="summary", fun="mean", aes(fill=Innoculation)) + # add fill statement
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



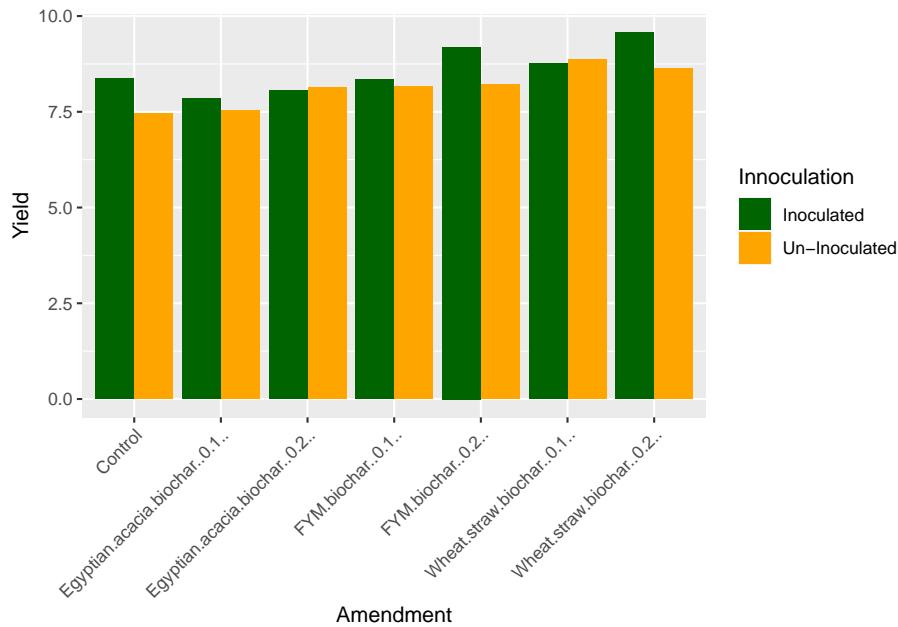
Ok, we've got different colored bars – but they are stacked! We want them side-by-side. So we need to add one more argument to our second line: `position="dodge"`. This tells R to position the bars so they “dodge” each other. It's a weird choice of words, but technically it works.

```
ggplot(data=biochar, aes(x=Amendment, y=Yield, group=Innoculation)) + # add group statement to t
  geom_bar(stat="summary", fun="mean", aes(fill=Innoculation), position="dodge") + # add fill s
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



And Voila, we have our plot. Whew, those color bars are high-contrast, which is great for accessibility. If we want to change the colors to something closer to our preferred palate, however,

```
ggplot(data=biochar, aes(x=Amendment, y=Yield, group=Innocation)) + # add group stat
  geom_bar(stat="summary", fun="mean", aes(fill=Innocation), position="dodge") + # dodges
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  scale_fill_manual(values = c("darkgreen", "orange"))
```



### 7.7.6 Practice

For practice we have a dataset inspired by “Plant population and row spacing effects on corn: Plant growth, phenology, and grain yield” (2020), published by Brad J. Bernhard Frederick E. Below in Agronomy Journal. The article is open source.

In this study, corn was grown with two row spacings and four plant populations. The trial was a complete random block design.

```
corn = read.csv("data-unit-7/exercise_data/corn_pop_spacing.csv")
head(corn)
```

```
##   block    pop spacing     yield
## 1     1 94000      76 16.90595
## 2     1 94000      51 16.72022
## 3     1 109000     76 15.69209
## 4     1 109000     51 18.51509
## 5     1 124000     76 16.26873
## 6     1 124000     51 17.26152
```

We need to do a couple of things to our dataset before we can run an analysis of variance. We need to reformat the data for block, pop, and spacing. Although I did not intend to make this part of the exercise, I'm glad it is in here.

ANOVA is based on named effects. But if we look under the words block, pop, and spacing, we see the expression “”. The problem is that we are using numbers to represent our treatments. If we run the analysis of variance on these data as-is, our table will look like this:

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
block	1	17.808	17.808	24.850	3.17e-05 *
spacing	1	<b>5.550</b>	<b>5.550</b>	<b>7.745</b>	<b>0.00971</b>
pop:spacing	1	1.318	1.318	1.839	0.18633
Residuals	27	19.349	0.717		

If you see nothing wrong, be comforted this has happened to me *many* times. But look again, particularly at the degrees of freedom. Do you see it now? We have four blocks in this trial, so we should have three degrees of freedom. We have four populations, so again we should have three degrees of freedom. Our degrees of freedom for population should be equal to the population degrees of freedom ( $4-1 = 3$ ) times the spacing degrees of freedom ( $2-1 = 1$ ), or three degrees.

What is going on is R thinks we want to create a regression model, and it will just have to be patient because we don't get to that until Unit 9! To tell R to run this analysis correctly, let's tell it to treat block, pop, and spacing as the factors they are, rather than as integers.

We are going to use the “as.factor” command to tell R these variables are factors. We use the ominous sounding command “mutate” to change our variables.

```
corn_fixed = corn %>% # the "%>%" tells R to take the corn dataset and use it in the next line
  mutate(block = as.factor(block), # mutate tells R to change the variables according to the new command
         pop = as.factor(pop),
         spacing = as.factor(spacing))
```

### 7.7.6.1 ANOVA

Now, run your ANOVA on the new dataset, corn\_fixed. The degrees of freedom will be correct. Your ANOVA output should look like:

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
block	3	18.791	6.264	12.841	5.53e-05 <i>pop 3 6.860 2.287 4.688 0.01169</i>
spacing	1	<b>5.550</b>	<b>5.550</b>	<b>11.379</b>	<b>0.00287</b>
pop:spacing	3	4.609	1.536	3.150	0.04645 *
Residuals	21	10.243	0.488		

Signif. codes: 0 ‘‘ **0.001** ‘‘ 0.01 ‘‘ 0.05 ‘‘ 0.1 ‘ ’ 1

### 7.7.6.2 Test Factors Individually

Test the effect of spacing at each level of pop. Your results should look like:

```
pop term df sumsq meansq statistic p.value 1 94000 .block32.200.7335.930.0889294000.spacing
1 0.336 0.336 2.72 0.198 3 94000 Residuals 3 0.371 0.124 NA NA
4 109000 .block36.002.001.740.3305109000.spacing 1 4.65 4.65 4.05 0.138 6
109000 Residuals 3 3.45 1.15 NA NA
7 124000 .block35.881.963.010.1958124000.spacing 1 3.72 3.72 5.71 0.0968 9
124000 Residuals 3 1.95 0.651 NA NA
10 139000 .block38.702.9018.00.020111139000.spacing 1 1.46 1.46 9.07 0.0571
12 139000 Residuals 3 0.482 0.161 NA NA
```

Test the effects of pop at each level of spacing. Your results should look like:

```
spacing term df sumsq meansq statistic p.value 1 51 .block38.242.755.250.0228251.pop
3 8.06 2.69 5.13 0.0243 3 51 Residuals 9 4.71 0.523 NA NA
4 76 .block311.33.767.030.00984576.pop 3 3.41 1.14 2.13 0.167
6 76 Residuals 9 4.81 0.534 NA NA
```

## 7.8 Exercise: Split-Plot Design

The split-plot design is similar to the factorial design but, as we learned in the lecture, has two error terms: one for the main factor, and the second for the sub-factor. This requires we use different R code than for the factorial design.

### 7.8.1 Case Study: Corn-Soybean Systems Trial

A cropping systems trial was conducted in Wisconsin to investigate the combined effects of tillage and in-furrow fungicide on soybean yield. This dataset was inspired by the following article, which is open-source:

Potratz, DJ, Mourtzinis, S, Gaska, J, Lauer, J, Arriaga, FJ, Conley, SP. Strip-till, other management strategies, and their interactive effects on corn grain and soybean seed yield. *Agronomy Journal*. 2020; 112: 72– 80. <https://doi.org/10.1002/agj2.20067>

The trial was a split-plot design with two levels of tillage (no-till, NT, and strip-till, ST) as the main factor and two levels of fungicide (Fungicide and No Fungicide) as the sub-factor. There were four replications. Main factor levels were blocked.

```
library(tidyverse)
soybean = read.csv("data-unit-7/exercise_data/tillage_fungicide_soybean.csv")
head(soybean)
```

```
##   block tillage   fungicide   yield
## 1     B1      NT    Fungicide 3.698314
## 2     B1      NT No Fungicide 3.452027
## 3     B1      ST    Fungicide 4.059679
## 4     B1      ST No Fungicide 3.867225
## 5     B2      NT    Fungicide 4.045291
## 6     B2      NT No Fungicide 3.902678
```

### 7.8.2 ANOVA

Lets first construct our linear model:

$$Y = \mu + T + \text{Error}(B + BT) + F + TF + \text{Error}(BF + BTF)$$

So our anova output should include include the effects of tillage (T), fungicide (F), and their interaction (TF). We also need to tell R to use the interaction of block and tillage to test the main factor tillage effect. We code this in R as follows.

```
soybean_model = aov(yield ~ tillage + fungicide + tillage:fungicide + Error(block:tillag

## Warning in aov(yield ~ tillage + fungicide + tillage:fungicide + 
## Error(block:tillage), : Error() model is singular

soybean_model = with(soybean, sp.plot(block, tillage, fungicide, yield))

## 
## ANALYSIS SPLIT PLOT: yield
## Class level information
##
## tillage : NT ST
## fungicide : Fungicide No Fungicide
## block : B1 B2 B3 B4
##
## Number of observations: 16
##
## Analysis of Variance Table
##
## Response: yield
##              Df  Sum Sq Mean Sq F value    Pr(>F)
## block          3 2.17303 0.72434    NaN      NaN
## tillage        1 0.38181 0.38181 47.9344 0.006177 ***
## Ea             3 0.02390 0.00797    NaN      NaN
## fungicide      1 0.26756 0.26756 23.4352 0.002878 **
```

```

## tillage:fungicide 1 0.00017 0.00017 0.0147 0.907315
## Eb                 6 0.06850 0.01142     NaN      NaN
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## cv(a) = 2.1 %, cv(b) = 2.5 %, Mean = 4.289329

```

Of particular note is the new “Error” term we have added. This tells R which error to use for testing the tillage effect.

The ANOVA output is created using the *summary()* function.

```
summary(soybean_model)
```

```

##      Length Class  Mode
## ANOVA 5     anova  list
## gl.a   1     -none- numeric
## gl.b   1     -none- numeric
## Ea    1     -none- numeric
## Eb    1     -none- numeric

```

As we saw in lecture, the ANOVA output for a split-plot experiment includes two tables. The top table tests the main factor effect. At the top of that table, it specifies we are using the block:tillage interaction as our error term. The bottom table tests the sub-factor effect and the interaction between the main factor and the sub-factor.

We can see tillage did not have an effect. Fungicide did affect yield.

### 7.8.3 Interaction Plot

Although the interaction is not significant, we can still draw a line plot to visualize the relationship between tillage and fungicide effects. First, we need to create a summary dataset with mean yields for the interactions of tillage and fungicide.

```

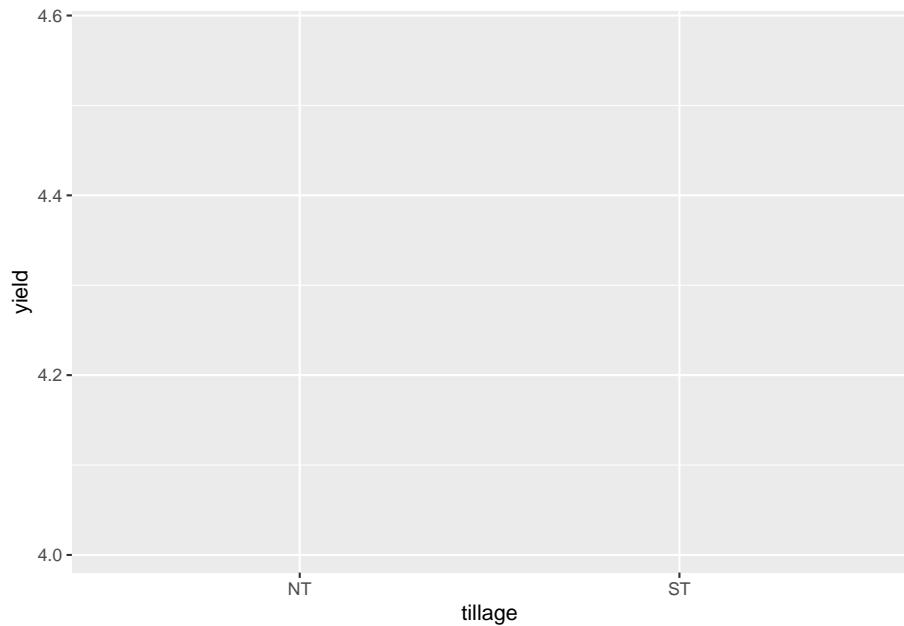
soybean_group = soybean %>%      # build new dataframe, soybean_group, on summary of soybean datafr
  group_by(tillage, fungicide) %>%  # summarise based on interaction of tillage and fungicide
  summarise(yield = mean(yield)) %>%  # summarise by means across replications
  ungroup()    # ungroup for further processing

## `summarise()` has grouped output by 'tillage'. You can override using the
## `.groups` argument.

```

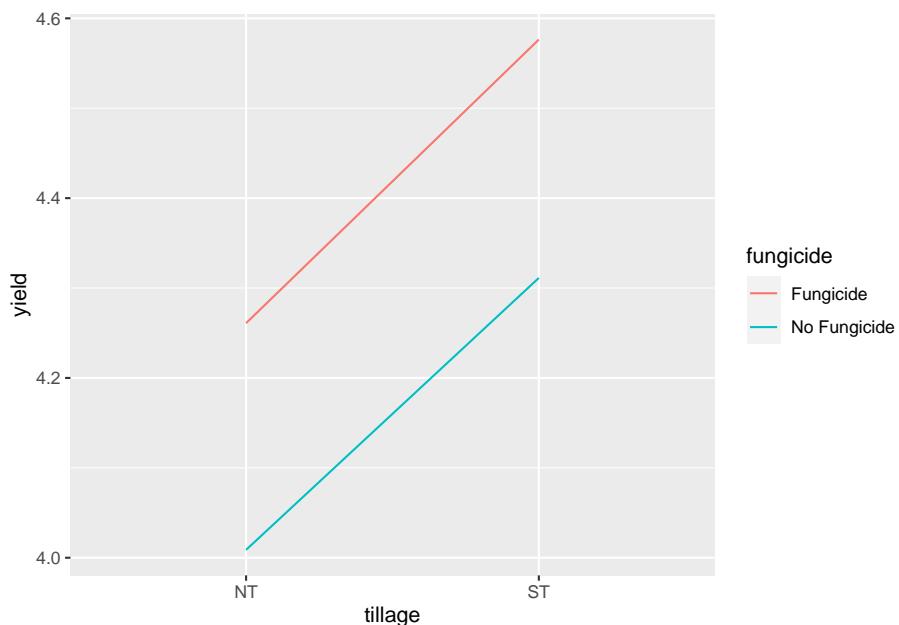
Then we create our line plot. We can start the plot with the `ggplot()` function. The `aes()` argument tells R to position the data points according to their tillage level and yield. It also tells R to group the points by fungicide level.

```
soybean_group %>%
  ggplot(aes(x=tillage, y=yield, group=fungicide))
```



Next we add line geometries to our plot using `geom_line()`. We use another `aes()` argument to tell R to differentiate line color by level of fungicide.

```
soybean_group %>%
  ggplot(aes(x=tillage, y=yield, group=fungicide)) +
  geom_line(aes(color=fungicide))
```

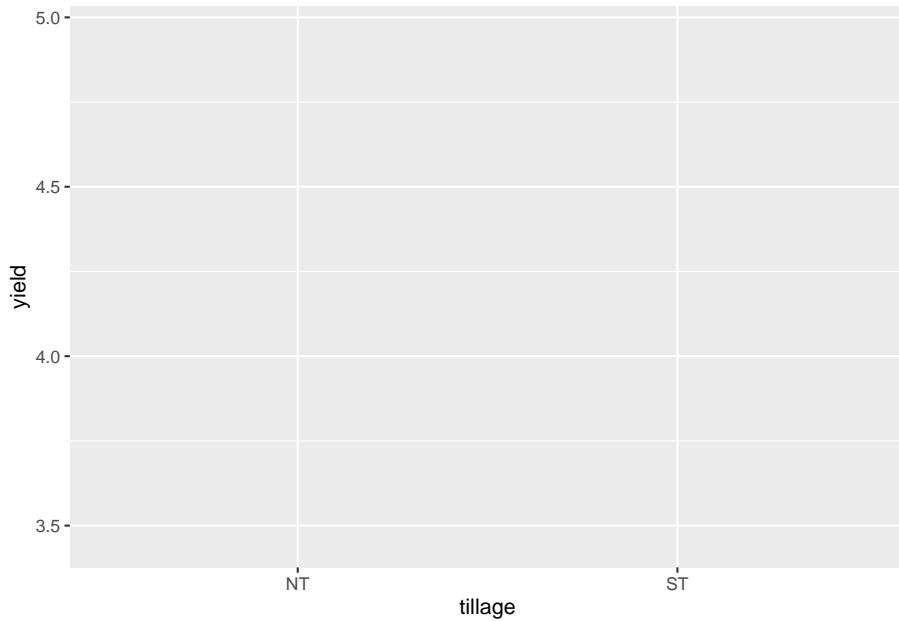


#### 7.8.4 Bar plot

We can create a bar plot just as we did for the factorial trial.

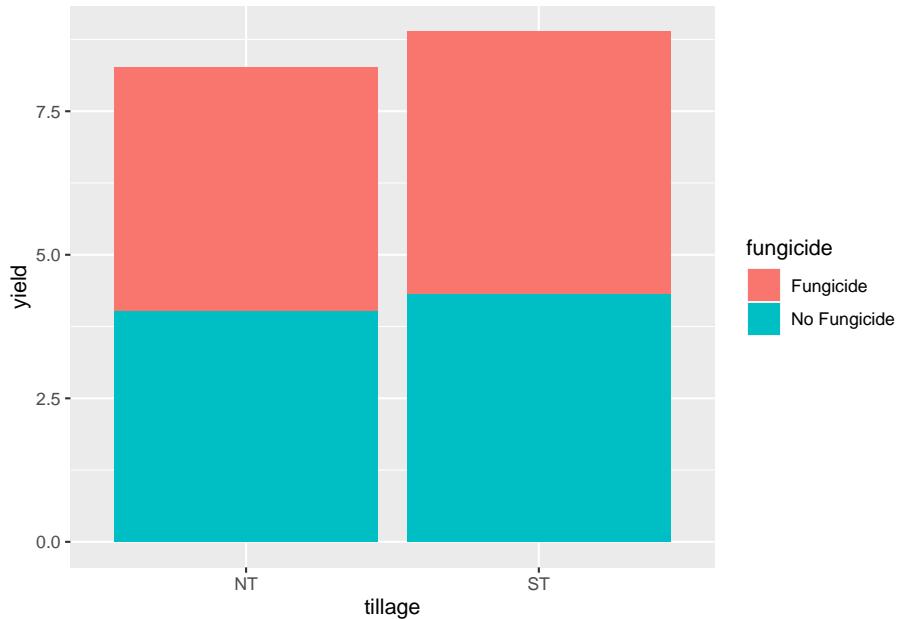
First, we tell ggplot with the aes() argument to position the bars according to their tillage level and their yield, and to group means with the same fungicide level together so they are the same color.

```
soybean %>%
  ggplot(aes(x=tillage, y=yield, group=fungicide))
```



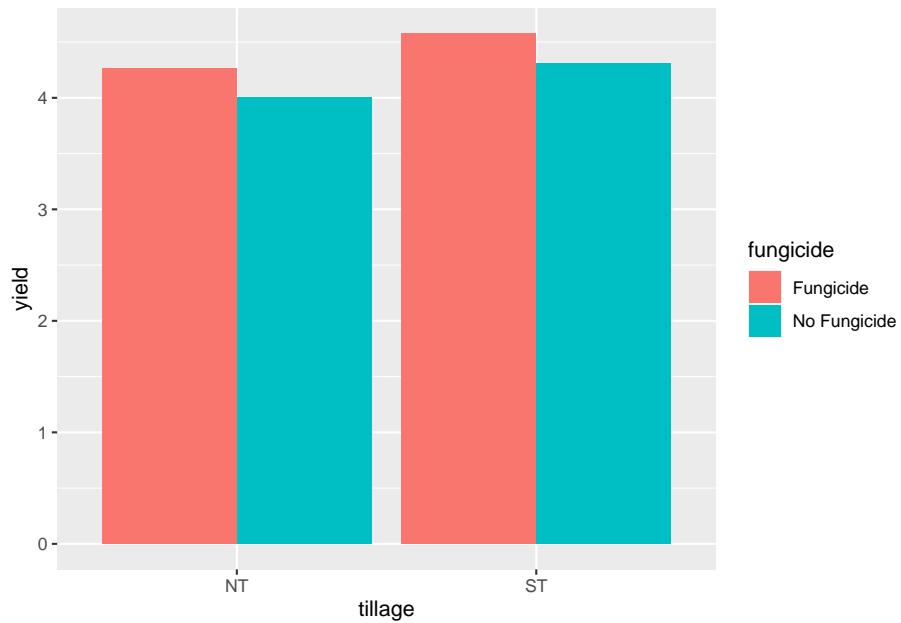
Then we add our geometry with the `geom_bar()` argument. We will add an `aes()` argument to tell R that bar color should be matched to fungicide level. We also need to use the `stat="summary"` argument that the statistic to be plotted is a summary statistic to be calculated. The we need to tell it with `fun="mean"` that the statistic to be calculated is the mean.

```
soybean %>%
  ggplot(aes(x=tillage, y=yield, group=fungicide)) +
  geom_bar(aes(fill=fungicide), stat = "summary", fun="mean")
```



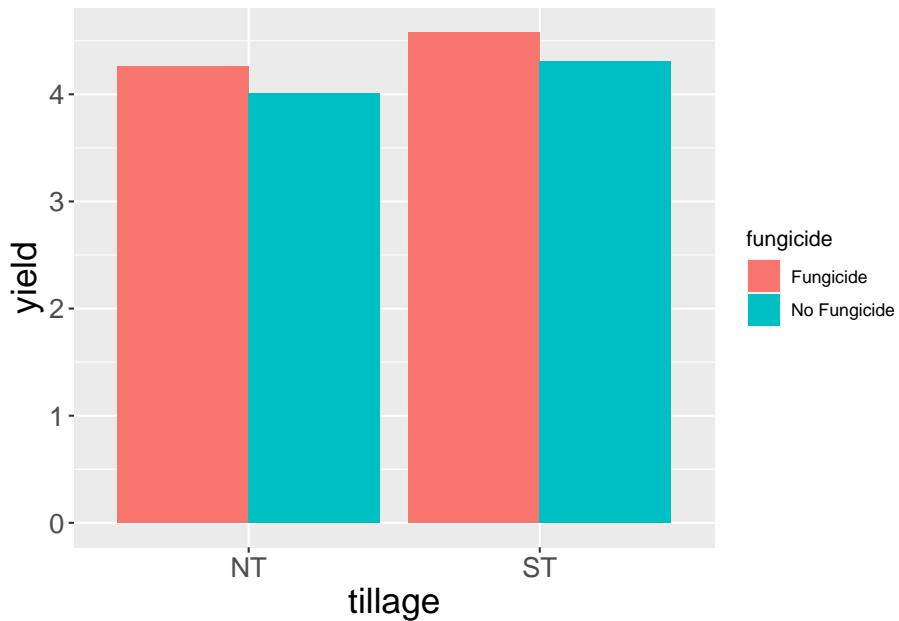
Whoops! We need to add a final argument to `geom_bar()`, to tell it to plot the two levels of fungicide next to each other, instead of stacking them. We add `position="dodge"` accordingly.

```
soybean %>%
  ggplot(aes(x=tillage, y=yield, group=fungicide)) +
  geom_bar(aes(fill=fungicide), stat = "summary", fun="mean", position="dodge")
```



Finally, if we want, we can add a couple of theme() arguments to increase the font size of the axis titles and labels.

```
soybean %>%
  ggplot(aes(x=tillage, y=yield, group=fungicide)) +
  geom_bar(aes(fill=fungicide), stat = "summary", fun="mean", position="dodge") +
  theme(axis.title = element_text(size=18),
        axis.text = element_text(size=14)) # element_text() tells R we are modifying
```



### 7.8.5 Practice

The same study above produced a corn dataset. The main factor was rotation and the sub-factor was fungicide.

```
corn = read.csv("data-unit-7/exercise_data/rotation_fungicide_corn.csv")
head(corn)
```

```
##   block rotation   fungicide     yield
## 1     B1        CS   Fungicide 12.95823
## 2     B1        CS  No Fungicide 12.40752
## 3     B1        CC   Fungicide 11.49778
## 4     B1        CC  No Fungicide 11.60836
## 5     B2        CS   Fungicide 14.19937
## 6     B2        CS  No Fungicide 13.69898
```

#### 7.8.5.1 ANOVA

Create an analysis of variance for the data above. Your results should look like:

```
Error: block:rotation Df Sum Sq Mean Sq F value Pr(>F)
rotation 1 4.99 4.987
0.896 0.38 Residuals 6 33.39 5.565
```

```
Error: Within Df Sum Sq Mean Sq F value Pr(>F)
fungicide 1 0.1309 0.1309 64.19 0.000202 rotation:fungicide 1 0.3492
0.3492 171.19 1.23e-05 Residuals 6 0.0122 0.0020
Signif. codes: 0 ‘‘ 0.001 ‘‘ 0.01 ‘‘ 0.05 ‘‘ 0.1 ‘ ’ 1
```

### 7.8.5.2 Test Factors Individually

First, test the effect of rotation at each level of fungicide. Your results should look like:

```
fungicide term df sumsq meansq statistic p.value 1 Fungicide .block316.65.526385.0.000003332Fungicide.r
1 3.99 3.99 4612. 0.00000703 3 Fungicide Residuals 3 0.00259 0.000865 NA NA
4 No Fungicide .block316.85.616068.0.000003595NoFungicide.rotation 1 1.35
1.35 1458. 0.0000395 6 No Fungicide Residuals 3 0.00277 0.000925 NA NA
```

Second, test the effect of fungicide at each level of rotation. Your results should look like:

```
rotation term df sumsq meansq statistic p.value 1 CC .block316.75.584075.0.000006522CC.fungicide
1 0.0262 0.0262 19.2 0.0220
3 CC Residuals 3 0.00410 0.00137 NA NA
4 CS .block316.75.562049.0.00001835CS.fungicide 1 0.454 0.454 167. 0.000997
6 CS Residuals 3 0.00814 0.00271 NA NA
```

### 7.8.5.3 Create a line plot to view the interaction between rotation and fungicide.

## 7.9 Exercise: Experimental Design

Now that you have been introduced to four powerful experimental designs – completely randomized, randomized complete block, factorial, and split-plot, how can you use R to create your plot layout and randomize your treatments?

Of course, R has a package for that. We will use the *agricolae()* packages and its multiple tools for experimental design.

### 7.9.1 Completely Randomized Design

We have a hemp trial with five varieties: “China Cat”, “Eyes”, “Scarlet”, “Fire”, and “Tower”. How do we randomize it? We use the *design.crd()* function.

```
library(agricolae)

# First, we need to define the varieties as a vector:
```

```

varieties = c("China Cat", "Eyes", "Scarlet", "Fire", "Tower")

# Then we define the number of replications we want:
reps = 6

# One more thing we can do is provide a "seed". A seed tells R where to start in randomizing the
seed_no = 910

# then we just feed these to the design.crd argument
crd = design.crd(trt=varieties, r=reps, seed = seed_no)

```

design.crd outputs a list (a collection of R objects). To view the treatment assignments in our new plot plan, “crd”, we add “\$book” to the end of it.

```
crd$book
```

```

##   plots r varieties
## 1    101 1     Fire
## 2    102 1     Eyes
## 3    103 2     Eyes
## 4    104 1   Scarlet
## 5    105 2   Scarlet
## 6    106 2     Fire
## 7    107 3     Eyes
## 8    108 3     Fire
## 9    109 4     Eyes
## 10   110 1     Tower
## 11   111 4     Fire
## 12   112 2     Tower
## 13   113 5     Fire
## 14   114 1 China Cat
## 15   115 5     Eyes
## 16   116 2 China Cat
## 17   117 3 China Cat
## 18   118 3   Scarlet
## 19   119 3     Tower
## 20   120 4   Scarlet
## 21   121 4     Tower
## 22   122 6     Eyes
## 23   123 6     Fire
## 24   124 5     Tower
## 25   125 5   Scarlet
## 26   126 4 China Cat

```

```
## 27    127 6    Tower
## 28    128 5 China Cat
## 29    129 6 China Cat
## 30    130 6    Scarlet
```

### 7.9.2 Randomized Complete Block Design

What if we want to block our treatments? For this, we use the `design.rcbd()` function. The values we feed to it will be identical to the completely randomized design above.

```
# First, we need to define the varieties as a vector:
varieties = c("China Cat", "Eyes", "Scarlet", "Fire", "Tower")

# Then we define the number of replications we want:
reps = 6

# One more thing we can do is provide a "seed". A seed tells R where to start in random
seed_no = 910

rcbd = design.rcbd(trt = varieties, r=reps, seed = seed_no)
```

We can review our treatment assignments using “`rcbd$book`”.

```
rcbd$book
```

```
##   plots block varieties
## 1    101    1 China Cat
## 2    102    1     Fire
## 3    103    1    Scarlet
## 4    104    1     Eyes
## 5    105    1    Tower
## 6    201    2     Fire
## 7    202    2     Eyes
## 8    203    2    Tower
## 9    204    2    Scarlet
## 10   205    2 China Cat
## 11   301    3     Fire
## 12   302    3 China Cat
## 13   303    3    Scarlet
## 14   304    3    Tower
## 15   305    3     Eyes
## 16   401    4     Fire
## 17   402    4    Scarlet
```

```
## 18    403    4 China Cat
## 19    404    4     Eyes
## 20    405    4    Tower
## 21    501    5 China Cat
## 22    502    5   Scarlet
## 23    503    5    Tower
## 24    504    5     Eyes
## 25    505    5     Fire
## 26    601    6    Tower
## 27    602    6   Scarlet
## 28    603    6     Fire
## 29    604    6     Eyes
## 30    605    6 China Cat
```

We can save this data.frame to our local directory – for use in our spreadsheet – using the `write.csv()` function.

```
# the write.csv arguments are (name of data.frame, filename, row.names=FALSE)
write.csv(rcbd$book, "field_book.csv", row.names = FALSE)
```

### 7.9.3 Factorial Design

We use the `design.ab()` function to create our plot layout.

The factorial experimental design is slightly different. We cannot give R named treatments. We can only tell it how many treatments are in each factor. Let's say we wanted to run a factorial experiment with three levels of hemp variety and two levels of soil amendments. We would call this a “3 x 2” factorial.

One additional change is we can tell R whether to arrange our treatments in a completely randomized design or a randomized complete block design. We will use the `design = “rcbd”` argument to tell R to create a randomized complete block design. (If we wanted to create a completely randomized design, we would use `design=“crd”` instead)

```
# the line below tells R that we have three levels of the first factor and two levels of the second
factors = c(3,2)
reps = 4
seed_no=910

fact = design.ab(trt = factors, r = reps, seed = seed_no, design = "rcbd")
```

We can again review our treatment assignments by using `$book`.

```
fact$book
```

```
##   plots block A B
## 1    101    1 1 1
## 2    102    1 2 1
## 3    103    1 1 2
## 4    104    1 2 2
## 5    105    1 3 1
## 6    106    1 3 2
## 7    107    2 2 2
## 8    108    2 1 1
## 9    109    2 2 1
## 10   110    2 3 1
## 11   111    2 3 2
## 12   112    2 1 2
## 13   113    3 2 2
## 14   114    3 1 1
## 15   115    3 1 2
## 16   116    3 3 1
## 17   117    3 2 1
## 18   118    3 3 2
## 19   119    4 2 2
## 20   120    4 3 1
## 21   121    4 2 1
## 22   122    4 3 2
## 23   123    4 1 2
## 24   124    4 1 1
```

#### 7.9.4 Split-Plot Design

Lets say we are going to use equipment to spread our compost. The applicator creates a 6-meter swath, while our three varieties are being grown in plots 2 meters wide. We deside to conduct a split plot trial with soil amendment as our main factor and variety as our sub-factor.

For the split plot design, will define the levels of our two factors, varieties and amendments, by name

```
# First, we need to define the varieties as a vector:
varieties = c("Eyes", "Scarlet", "Fire")

# Second, we define our amendment treatments
amendments = c("control", "compost")
#
```

```
# Then we define the number of replications we want:
reps = 4

# One more thing we can do is provide a "seed". A seed tells R where to start in randomizing the
seed_no = 910

# The design.split() function is similar to the factorial design.ab function above. The main difference
# is that it allows for multiple treatments to be assigned to each plot.
spdesign = design.split(trt1 = amendments, trt2 = varieties, r=4, seed = seed_no, design = "rcbd")
```

Our treatment assignments are below:

```
spdesign$book
```

```
##   plots splots block amendments varieties
## 1    101      1     1    control   Scarlet
## 2    101      2     1    control     Eyes
## 3    101      3     1    control     Fire
## 4    102      1     1   compost     Eyes
## 5    102      2     1   compost     Fire
## 6    102      3     1   compost   Scarlet
## 7    103      1     2    control     Fire
## 8    103      2     2    control     Eyes
## 9    103      3     2    control   Scarlet
## 10   104      1     2   compost     Fire
## 11   104      2     2   compost     Eyes
## 12   104      3     2   compost   Scarlet
## 13   105      1     3   compost     Eyes
## 14   105      2     3   compost     Fire
## 15   105      3     3   compost   Scarlet
## 16   106      1     3    control     Eyes
## 17   106      2     3    control     Fire
## 18   106      3     3    control   Scarlet
## 19   107      1     4   compost   Scarlet
## 20   107      2     4   compost     Fire
## 21   107      3     4   compost     Eyes
## 22   108      1     4    control     Fire
## 23   108      2     4    control   Scarlet
## 24   108      3     4    control     Eyes
```

### 7.9.5 Practice

We now have a new hemp trial comparing varieties “Chicago”, “New York”, and “Detroit”. The critics say they’re all on the same “street” (whatever), but we

would like to compare them.

#### 7.9.5.1 Completely Randomized Design

Create a completely randomized design with 4 replicates. Please use seed\_no=123 so our results will match.

```
varieties = c("Chicago", "New York", "Detroit")
reps=4
seed_no=123
```

Your plan should look like:

plots r varieties 1 101 1 New York 2 102 1 Detroit 3 103 1 Chicago 4 104 2 Chicago 5 105 2 Detroit 6 106 3 Chicago 7 107 3 Detroit 8 108 2 New York 9 109 4 Detroit 10 110 3 New York 11 111 4 Chicago 12 112 4 New York

#### 7.9.5.2 Randomized Complete Block Design

Create a randomized complete block design using the same three varieties as above. Please again use seed\_no=123 so our results will match.

```
varieties = c("Chicago", "New York", "Detroit")
reps=4
seed_no=123
```

Your plan should look like:

plots block varieties 1 101 1 Detroit 2 102 1 Chicago 3 103 1 New York 4 201 2 Detroit 5 202 2 Chicago 6 203 2 New York 7 301 3 New York 8 302 3 Chicago 9 303 3 Detroit 10 401 4 New York 11 402 4 Detroit 12 403 4 Chicago

#### 7.9.5.3 Factorial Design

Now lets take our 3 levels of hemp variety and treat each with two levels of foliar spray (control and kelp). We will arrange our treatments in blocked (design="rcbd") factorial design. We will use 2 reps (only to keep our practice data set small – more reps would be better!). Please again use seed\_no=123

```
factors = c(3,2)
reps=2
seed_no=123
```

Your results should look like: plots block A B 1 101 1 1 1 2 102 1 3 2 3 103 1 2 2 4 104 1 1 2 5 105 1 3 1 6 106 1 2 1 7 107 2 2 2 8 108 2 1 2 9 109 2 3 2 10 110 2 1 1 111 2 3 1 12 112 2 2 1

#### 7.9.5.4 Split Plot Design

Now lets suppose we had a 9-meter spray boom and variety plots that were 3-meters wide. Lets design a split plot trial with foliar spray as the main factor and hemp variety as the subfactor. We will block the main plots (design="rcbd"). Please again use seed\_no=123 and fill in the remaining code.

```
varieties = c("Chicago", "New York", "Detroit")
foliar = c("control", "kelp")
reps=2
seed_no=123
```

Your results should look like:

```
plots splots block foliar varieties 1 101 1 1 control Chicago 2 101 2 1 control
New York 3 101 3 1 control Detroit 4 102 1 1 kelp New York 5 102 2 1 kelp
Chicago 6 102 3 1 kelp Detroit 7 103 1 2 control New York 8 103 2 2 control
Detroit 9 103 3 2 control Chicago 10 104 1 2 kelp New York 11 104 2 2 kelp
Chicago 12 104 3 2 kelp Detroit
```



## Chapter 8

# Means Separation and Data Presentation

The previous two units focused on the design and analysis of effects in multiple treatment files. Our focus was to determine whether treatment effects explained more of the variation among individuals in a population than error (or residual) effects, which are based on unexplained differences among individuals.

In the first half of this unit, we will learn three common tools used for testing the differences *between* treatments. This is often the key purpose of a research trial. We know going in that some treatments will be different. But we don't know how they will rank, and whether the difference between them will be great enough to infer one is better than another.

In the second half, we will learn how to present treatment means in tables and plots. Proper data allows the reader to not only grasp results, but even incorporate some of your findings into their own work.

### 8.1 Case Study

Our sample dataset is inspired by Salas, M.C.; Montero, J.L.; Diaz, J.G.; Berti, F.; Quintero, M.F.; Guzmán, M.; Orsini, F. Defining Optimal Strength of the Nutrient Solution for Soilless Cultivation of Saffron in the Mediterranean. *Agronomy* 2020, 10, 1311.

Saffron is a spice made from the anthers of the saffron flower. It has a nuanced, sweet, complex flavor, and is used in dishes from Cornish saffron rolls to Spanish paella to Iranian tahdig. It comes from the anthers of the field-grown saffron flower and must be hand picked, making it very expensive.

In this study, saffron was grown hydroponically in 15-L pots filled with perlite, with four nutrient concentrations as defined by electroconductivity (EC): low (EC 2.0), medium (EC 2.5), high (EC 3.0), and very high (EC 4.0). The effect of the solutions on corm production (needed to propagate the crop) was measured.

```
library(tidyverse)

## -- Attaching packages ----- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr    1.0.8
## v tidyverse 1.2.0    v stringr  1.4.0
## v readr   2.1.2     vforcats 0.5.1

## Warning: package 'ggplot2' was built under R version 4.1.3

## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()   masks stats::lag()

saffron = read.csv("data-unit-8/saffron.csv")
head(saffron)

##   plot block rate corm_number
## 1   1     1   1X      246
## 2   2     2   4X      240
## 3   3     3   3X      254
## 4   4     1   2X      285
## 5   5     2   4X      233
## 6   6     2   3X      251
```

## 8.2 Least Significant Difference

Perhaps the most straightforward method of means separation is the infamous Least Significant Difference test. Not to be confused with the psychadelic parties in Marin County, California, the LSD test is as simple as this:

- calculate the least significant difference
- any treatment differences that are equal to or greater than the least significant difference are – you guessed it – significant

The least significant difference is calculated as follows:

$$LSD_{df,\alpha} = t_{df,\alpha} \cdot SED$$

Where  $LSD_{df,\alpha}$  is the least significant difference, given the degrees of freedom associated with the error effect and the desired significance.

Does this formula look vaguely familiar? Does it remind you of what you were doing back in Unit 4? Great, because this is the same formula we used to calculate the distance between confidence limits and the sample mean back when we learned t-tests. Back then, we saw how the confidence interval was used to test the probability our observed difference between treatments was different from zero. Recall if zero fell outside our confidence interval, we inferred the two treatments were different. Similarly, if the difference between two treatments is greater than the least significant difference, we infer the treatments are significantly different.

In R, we use a function, `LSD.test()`, which is part of the *agricolae* package, to calculate the LSD. First, however, lets run an analysis of variance on our t data. The experiment was a randomized complete block design, so our linear additive model is:

$$Y_{ij} = \mu + B_i + T_j + BT_{ij}$$

Where  $Y_{ij}$  is the number of corms,  $\mu$  is the overall population mean for the trial,  $B_i$  is the block effect,  $T_j$  is the treatment effect, and  $BT_{ij}$  is the block effect.

Our analysis of variance result is below. The effect of fertilization rate is highly significant. And this brings us to an important rule for using the LSD test. We only use the LSD test to separate means *if* the treatment effect is significant in our analysis of variance. Doing otherwise can lead to errors, as we will discuss below.

```
saffron$block = as.factor(saffron$block)
saffron_model = aov(corm_number ~ block + rate, saffron)
summary(saffron_model)

##          Df Sum Sq Mean Sq F value    Pr(>F)
## block      3     85   28.4   2.465    0.129
## rate       3   4473  1490.9 129.333 1.02e-07 ***
## Residuals  9    104    11.5
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

## 8.3 LSD Output in R

Now that we know the effect of fertilization rate is highly significant, we want to know how the individual treatments rank, and whether they are significantly different from one another. The results of our LSD test are below.

```
library(agricolae)
lsd = LSD.test(saffron_model, "rate")
lsd

## $statistics
##   MSerror Df      Mean       CV t.value     LSD
##   11.52778  9 257.125 1.32047 2.262157 5.43101
##
## $parameters
##           test p.adjusted name.t ntr alpha
## Fisher-LSD      none    rate    4  0.05
##
## $means
##   corm_number      std r      LCL      UCL Min Max   Q25   Q50   Q75
## 1X      249.25 4.272002 4 245.4097 253.0903 246 255 246.00 248.0 251.25
## 2X      284.75 2.629956 4 280.9097 288.5903 281 287 284.00 285.5 286.25
## 3X      254.25 2.872281 4 250.4097 258.0903 251 258 253.25 254.0 255.00
## 4X      240.25 5.439056 4 236.4097 244.0903 233 246 238.25 241.0 243.00
##
## $comparison
## NULL
##
## $groups
##   corm_number groups
## 2X      284.75    a
## 3X      254.25    b
## 1X      249.25    b
## 4X      240.25    c
##
## attr(,"class")
## [1] "group"
```

Lets unpack this piece by peace. The output from the LSD test is in a list of tables.

### 8.3.1 Statistics Table

Let's start with the `$statistics` table. This explains how our LSD was calculated:

- MSerror is the error or residual mean square. It should match that value from the ANOVA table above. Recall that MSerror is an estimate the variance within treatments – that is, the variation among plots unexplained by our model. Therefore, its square root is the standard deviation of the observations within each treatment.
- DF is the degrees of freedom, which is used to calculate our t.value
- Mean is just that – the overall mean of the trial.
- CV is the coefficient of variance. By dividing the standard deviation by the mean, and multiplying by 100, we arrive at this value. Recall from Unit 6 that the CV is a measure of the quality control of the trial: how consistent were our experimental units?
- The t-value is based on the degrees of freedom and  $\alpha$ , the desired p-value (often 0.05) to be used to test significance.
- LSD is the product of the t-value and the standard error of the difference, which can be derived from MSerror and the number of replicates.

This is a lot to absorb, I realize. The most important two statistics for you to understand from this table are the CV and LSD. The other numbers are intermediate values, although if you list the LSD in a report you should also report the degrees of freedom used to calculate the t-value.

### 8.3.2 Means Table

The `$means` table explains the distribution of observations within a treatment level around their sample mean. Most of these concepts we discussed in Unit 4. The statistics are:

- corm\_number: These are our sample means for each level of treatment
- std: the standard error of the sample mean. This is unique to the sample mean for each treatment.
- r: the number of replicates per treatment level
- LCL and UCL: the lower confidence limit and upper confidence limit for each mean. These are calculated just as we did in Unit 4.

The remainder of the statistics show the minimum and maximum values, and the quartiles.

### 8.3.3 Groups Table

Often, the `$groups` table is the most interesting, for it tests the differences among levels of a treatment. The treatment means among levels are ranked from greatest to least.

- `corm_number`: again, the sample mean for each level of treatment
- `group`: this groups treatments that are statistically equivalent. Any means followed by the same letter are considered equal. In the table above, the mean corm numbers associated with the 1X and 3X rates of fertilizer are considered equal. Any means not followed by the same letter are considered statistically different at the p-level chosen to test those differences. The 2X rate produced significantly greater corm numbers than the other fertilizer rates. The 4X rate produced statistically lesser corm numbers than the other fertilizer rates.

## 8.4 Comparisonwise versus Experimentwise Error

It is important LSD tests not be used indiscriminately to separate means. The problem is that each time we test the difference between two means, we have a 5% chance of committing a type I error. This is the probability of committing a Type I error in *that* comparison. This is known as the *comparisonwise error rate*.

The probability of committing a type I error across all the comparisons is known as the *experimentwise error rate*. If we only conduct one comparison, our experimentwise error rate is equal to the comparisonwise error rate. But if we conduct two comparisons, our experimentwise error rate is equal to the probability that comparisonwise errors will not occur in both comparison.

We can demonstrate this with some simple calculations. Before we start our comparisons, there is a 100% chance we have not committed an experimentwise Type I error. We cold express this as:

$$\text{Experimentwise Error} = 1$$

If we once comparison, there is 5% probability of a comparisonwise Type I error – and a 95% chance the error will not occur. We can express this as

$$\text{Experimentwise Error} = 1 - 0.95 = 0.05$$

If we have two comparisons, there is a 95% probability a comparisonwise Type I error won't occur in the first comparison – and a 95% probability it won't

occur in the second comparison. But the probability it doesn't occur in both comparisons is  $0.95 * 0.95$ :

$$\text{Experimentwise Error} = 1 - 0.95 \times 0.95 = 1 - 0.9025 = 0.0975$$

Now the Experimentwise error rate is 0.0975, or 9.75%.

What about three comparisons?

$$\text{Experimentwise Error} = 1 - 0.95 \times 0.95 \times 0.95 = 1 - 0.8573 = 0.1427$$

The Experimentwise error rate is now 0.1427, or 14.27%.

Finally, what if we had 10 comparisons?

$$\text{Experimentwise Error} = 1 - 0.95^{10} = 1 - 0.5987 = 0.4013$$

Our experimentwise error rate is now about 0.40, or 40%.

As the number of our comparisons increases, so does the probability of an experimentwise error. How can we avoid this? The first method, mentioned above, is to not use the LSD test unless the ANOVA shows a significant treatment effect. We call this approach the *F-protected LSD test*.

The second approach is to use a multiple range test that increases its minimum significant difference for comparing treatments as the number of treatments increases.

## 8.5 Tukey's Honest Significant Difference

If we are going to be comparing many treatments, it is better to use a minimum significant difference, like Tukey's Honest Significant Different (HSD) Test. Tukey's HSD is calculated very similarly to the least significant difference:

$$\text{Tukey's HSD} = Q_{\alpha, df, k} \cdot SED$$

The difference in Tukey's HSD is that we use  $Q$ , the “studentized range distribution” (you can just call it  $Q$  in this course) in place of the t-distribution.  $Q$  differs from  $t$  in that its value is determined not only by  $\alpha$ , the desired probability of a Type I error, and  $df$ , the degrees of freedom associated with the error mean square, but also  $k$ , the number of treatments.

The plot below includes four “ $Q$ ” distribution curves, associated with 3, 5, 6, and 10 treatments. Observe how the distributions shift to the right as the number of treatments increases. This means that the minimum difference for significance also increases with the number of treatments.

```

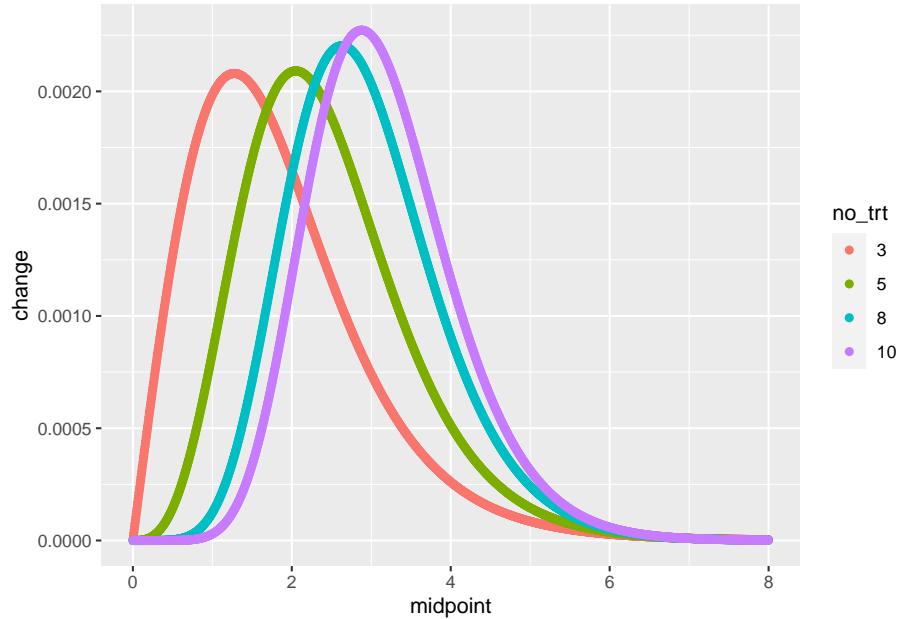
tukey_list = list()
for(k in c(3:10)){
  x_ptukey <- seq(0, 8, by = 0.005)          # Specify x-values for ptukey
  y_ptukey <- ptukey(x_ptukey, nmeans = k, df = 3*k)    # Apply ptukey function
  tukey_df = data.frame(x_ptukey) %>%
    cbind(y_ptukey) %>%
    mutate(change = y_ptukey - lag(y_ptukey)) %>%
    mutate(midpoint = (x_ptukey + lag(x_ptukey))/2) %>%
    mutate(no_trt = k) %>%
    as.data.frame()
  i=k-2
  tukey_list[[i]] = tukey_df
}

tukey_df_all = bind_rows(tukey_list)

tukey_df_all %>%
  filter(no_trt %in% c(3,5,8,10)) %>%
  mutate(no_trt = as.factor(no_trt)) %>%
  ggplot(aes(x=midpoint, y=change, group=no_trt)) +
  geom_point(aes(color=no_trt))

## Warning: Removed 4 rows containing missing values (geom_point).

```



The Tukey test output below is very similar to the LSD test output. The “\$statistics” section is identical to that of the LSD output, except it now reports the minimum significant difference instead of the least significant difference. In the “\$parameters” section, the “Studentized Range” value (Q) is given in place of the t-value. The “\$groups” section can be interpreted the same for the minimum significant difference as for the least significant difference.

```

tukey = HSD.test(saffron_model, "rate", group=TRUE)
tukey

## $statistics
##   MSerror Df      Mean       CV       MSD
##   11.52778  9 257.125 1.32047 7.494846
##
## $parameters
##   test name.t ntr StudentizedRange alpha
##   Tukey    rate    4        4.41489  0.05
##
## $means
##   corm_number     std r Min Max   Q25   Q50   Q75
##   1X      249.25 4.272002 4 246 255 246.00 248.0 251.25
##   2X      284.75 2.629956 4 281 287 284.00 285.5 286.25
##   3X      254.25 2.872281 4 251 258 253.25 254.0 255.00
##   4X      240.25 5.439056 4 233 246 238.25 241.0 243.00
##
## $comparison
## NULL
##
## $groups
##   corm_number groups
##   2X      284.75    a
##   3X      254.25    b
##   1X      249.25    b
##   4X      240.25    c
##
## attr(,"class")
## [1] "group"

```

Unlike the LSD test, the Tukey test does not need to be “protected” by first examining whether the Analysis of Variance treatment effect is significant. That said, the Tukey test is unlikely to indicate a significant difference between treatments without the ANOVA treatment effect being significant as well.

## 8.6 Linear Contrast

The last type of means comparison we will learn in this lesson is the linear contrast. Unlike the LSD and Tukey tests, linear contrasts may be used to separate two groups of treatments. While a lot of math may be introduced to the curious, in a linear contrast the statistician defines two groups of treatments through the use of *coefficients*; R then calculates their means and standard errors, and compares them using a t-test. There are multiple ways we can use a linear contrast, and our saffron dataset is a great way to introduce them.

### 8.6.1 Coefficients

Recall how a t-test between two treatments works. We calculate  $t$  as:

$$t = \frac{(\bar{x}_1 - \bar{x}_2) - (\mu_1 - \mu_2)}{SED}$$

Where  $\bar{x}_1 - \bar{x}_2$  is the difference between sample means,  $(\mu_1 - \mu_2)$  is the hypothesized difference between treatments (usually zero), and SED is the standard error of the difference.

For most situations, we could simplify the above equation to:

$$t = \frac{\bar{x}_1 - \bar{x}_2}{SED}$$

What about if we are comparing more than two treatments? For our saffron example, what if we wanted to calculate the difference between the mean of the two lower rates (1X and 2X) and the mean of the two higher rates (3X and 4X)? Lets call this difference  $L$ . We would calculate this difference as:

$$L = \frac{\bar{x}_{1X} + \bar{x}_{2X}}{2} - \frac{\bar{x}_{3X} + \bar{x}_{4X}}{2}$$

All we are doing above is 1) calculating the means for the two groups and 2) subtracting the mean of the 3X and 4X rates from the mean of the 1X and 2X rates. Now let's express this same formula a little differently. What we are doing in the above equation is multiplying each number by 1/2:

$$L = \frac{1}{2}(\bar{x}_{1X} + \bar{x}_{2X}) - \frac{1}{2}(\bar{x}_{3X} + \bar{x}_{4X})$$

In addition (bear with me!), when we subtract the mean of treatments 3X and 4X, it is the equivalent of adding the negative value of their mean to the mean of treatments 1X and 2X:

$$L = \frac{1}{2}(\bar{x}_{1X} + \bar{x}_{2X}) + (-\frac{1}{2})(\bar{x}_{3X} + \bar{x}_{4X})$$

Finally, we can arrange the equation above as:

$$L = \frac{1}{2}\bar{x}_{1X} + \frac{1}{2}\bar{x}_{2X} - \frac{1}{2}\bar{x}_{3X} - \frac{1}{2}\bar{x}_{4X} +$$

The reason for this tortuous algebra flashback is to show you where the contrast coefficients come from. Each of the  $\frac{1}{2}$ s in the equation above is a contrast coefficient.

Let's demonstrate this with our saffron data. Our saffron treatment means are:

```
saffron_means = saffron %>%
  group_by(rate) %>%
  summarise(corm_number = mean(corm_number)) %>%
  ungroup()

saffron_means
```

```
## # A tibble: 4 x 2
##   rate   corm_number
##   <chr>     <dbl>
## 1 1X        249.
## 2 2X        285.
## 3 3X        254.
## 4 4X        240.
```

Now let's add in a column with our coefficients:

```
saffron_coefficients = saffron_means %>%
  mutate(coefficient = c(1/2, 1/2, -1/2, -1/2))
```

```
saffron_coefficients
```

```
## # A tibble: 4 x 3
##   rate   corm_number coefficient
##   <chr>     <dbl>      <dbl>
## 1 1X        249.       0.5
## 2 2X        285.       0.5
## 3 3X        254.      -0.5
## 4 4X        240.      -0.5
```

We see that R has converted these to decimals. We then create a new column, “mean\_x\_coefficient”, that is the product of the original mean and the coefficient. We see these products are approximately half the value of the original sample mean (some may be less than half because of rounding errors).

```
saffron_products = saffron_coefficients %>%
  mutate(mean_x_coefficient = corm_number * coefficient)

saffron_products
```

## # A tibble: 4 x 4	##   rate  corm_number  coefficient  mean_x_coefficient	##   <chr>     <dbl>        <dbl>            <dbl>	## 1 1X       249.         0.5           125.	## 2 2X       285.         0.5           142.	## 3 3X       254.        -0.5          -127.	## 4 4X       240.        -0.5          -120.
----------------------	---	--	---	---	---	---

Finally, we can sum the mean\_x\_coefficient column to get the total difference among treatments.

```
total_difference = sum(saffron_products$mean_x_coefficient)

paste("total difference = ", total_difference, sep="")
```

## [1] "total difference = 19.75"

One critical rule about requirements is their sum must always equal zero. Otherwise you are not completely identifying two groups to compare. Using coefficients that do not sum to zero can also suggest you are weighting one group unfairly compared to another.

####Contrast Calculations

To calculate  $t$ , of course, we must divide  $L$  by the standard error of the difference:

$$t = \frac{L}{SED}$$

So how is the standard error of the difference calculated?

Well, we know that the error mean square from our ANOVA is equal to the mean variance within treatments. And we know that if we take the square root of a variance, we get the standard deviation. And if we divide the standard deviation by the number of observations in each sample mean, we get the standard error.

In a paired t-test, where we are simply calculating a standard error for one set of numbers, the differences between each pair, the standard error of the difference is calculated the same as the standard error:

$$SED = SE = \frac{s}{\sqrt{n}}$$

Where  $s$  is the standard deviation and  $n$  is the number of observations (reps) per treatment. This formula is equal to that below, where we use the variance,  $s^2$ , in place of the standard deviation.

$$SED = \sqrt{\frac{s^2}{n}}$$

When we work with two-treatment trials where the treatments are not paired or blocked, we account for the variance and number replicates individually. For treatment levels “1” and “2”, the standard error of the difference becomes

$$SED = \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}$$

In most trials, however, we assume that the variances and number of replications are equal among treatments. So we can also express the above trial as:

$$SED = \sqrt{\frac{2 \cdot s^2}{n}}$$

Recall that in the analysis of variance for a multiple treatment trial, the error mean squares *is* the mean variance within treatments. So the equation above is equivalent to:

$$SED = \sqrt{\frac{2 \cdot EMS}{n}}$$

In a linear contrast, however, our standard error of the difference must be scaled according to the coefficients we use, since we are no longer comparing two treatments, but multiple. So our equation becomes:

$$SED = c \cdot \sqrt{\frac{EMS}{n}}$$

Where  $c$  is the square root of the sum of the squared constants is the For our example above, this sum,  $c$ , would be:

$$c = s \sqrt{\sum \left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^2 + \left(-\frac{1}{2}\right)^2 + \left(-\frac{1}{2}\right)^2}$$

Which is equal to:

$$c = \sqrt{\frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4}} = \sqrt{1} = 1$$

We can now calculate the standard error of the difference for our contrast. First, lets go back to our analysis of variance.

```
saffron_model = aov(corm_number ~ rate, saffron)
summary(saffron_model)
```

```
##           Df Sum Sq Mean Sq F value    Pr(>F)
## rate       3   4473  1490.9  94.66 1.28e-08 ***
## Residuals 12    189     15.7
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We can see from the ANOVA output thtat our error mean square is 15.75. We know from the design itself that we had 4 replicates

So our final standard error of the difference for our contrast is:

$$SED = 1 \cdot \sqrt{\frac{15.75}{4}} = 1 \cdot \sqrt{3.9375} = 1.984$$

The t-value for our test, would then be:

$$t = \frac{L}{SED} = \frac{19.75}{1.984} = 9.954$$

The probability of a t-value of 10, given the 12 degrees of freedom associated with the error sum of squares, would be:

```
pt(9.954, 12, lower.tail = FALSE)
```

```
## [1] 1.882084e-07
```

### 8.6.2 Linear Contrasts with R

We can automate linear contrast testing, of course, using R and the *glht()* function of the *multcomp* package. First, we need to define a matrix (table) of contrast coefficients for R. To get the coefficients in the correct order, lets double check the order in which the rate levels are listed in R. It is important to make sure our treatment is classified as a factor. We can do this using the *as.factor()* function

```
saffron$rate = as.factor(saffron$rate)
```

We can then list the order of the levels in treatment rate:

```
levels(saffron$rate)
```

```
## [1] "1X" "2X" "3X" "4X"
```

We can see they follow the order 1X, 2X, 3X, 4X. We will therefore form a matrix, K, with the appropriate coefficients.

```
K = matrix(c(1/2, 1/2, -1/2, -1/2), 1)
```

```
K
```

```
##      [,1] [,2] [,3] [,4]
## [1,]  0.5  0.5 -0.5 -0.5
```

We are now ready to run our contrast. First, we need to slightly alter our ANOVA model by adding a zero (0) between the tilde (~) and the treatment name. This is one of of those one-off ideoyncrasies of R.

```
library(multcomp)

## Loading required package: mvtnorm

## Loading required package: survival

## Loading required package: TH.data

## Loading required package: MASS

## 
## Attaching package: 'MASS'
```

```

## The following object is masked from 'package:dplyr':
##
##      select

##
## Attaching package: 'TH.data'

## The following object is masked from 'package:MASS':
##
##      geyser

saffron_model_for_contrast = aov(corm_number ~ 0 + rate, saffron)

low_vs_high = glht(saffron_model_for_contrast, linfct=K)
summary(low_vs_high)

##
##   Simultaneous Tests for General Linear Hypotheses
##
## Fit: aov(formula = corm_number ~ 0 + rate, data = saffron)
##
## Linear Hypotheses:
##             Estimate Std. Error t value Pr(>|t|)
## 1 == 0     19.750    1.984   9.953 3.77e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)

```

And there we have it, our contrast in seconds. Now let's interpret this. Estimate is the difference between groups. Since we subtracted the mean of the 3x and 4X rates from the mean of the 1X and 2X rates, the positive estimate value indicates the lower rates produced greater corm numbers than the higher rates.

“Std. Error” is the standard error of the difference, as we calculated above. The t-value is equal to the estimate divided by the standard error of the difference. Finally, “Pr( $>|t|$ )” is the probability of observing the t-value by chance. In this case, it is  $3.77 \times 10^{-7}$ , very close to zero. We conclude the lower two rates, as a group, produce greater corms than the upper two rates as a group.

We can quickly answer other questions of our data. For example, do the middle two rates (2X and 3X) produce a greater corm number than the lowest (1X and highest (4X) rates? Again, let's examine the order of our rate levels.

```
levels(saffron$rate)

## [1] "1X" "2X" "3X" "4X"
```

In order to subtract the mean corm number of rates 1X and 4X from the mean corm number of rates 2X and 3X, we will need to calculate the difference as:

$$L = \left(-\frac{1}{2}\right)\bar{x}_{1X} + \left(\frac{1}{2}\right)\bar{x}_{2X} + \left(\frac{1}{2}\right)\bar{x}_{3X} + \left(-\frac{1}{2}\right)\bar{x}_{4X} +$$

So our contrasts coefficients would be  $-\frac{1}{2}$ ,  $\frac{1}{2}$ ,  $\frac{1}{2}$ ,  $-\frac{1}{2}$ . Our contrast maxtrix is then:

```
K = matrix(c(-1/2, 1/2, 1/2, -1/2), 1)

K

##      [,1] [,2] [,3] [,4]
## [1,] -0.5  0.5  0.5 -0.5
```

We are now ready to run our contrast. First, we need to slightly alter our ANOVA model by adding a zero (0) between the tilde (~) and the treatment name. This is one of of those one-off ideoyncrasies of R.

```
library(multcomp)

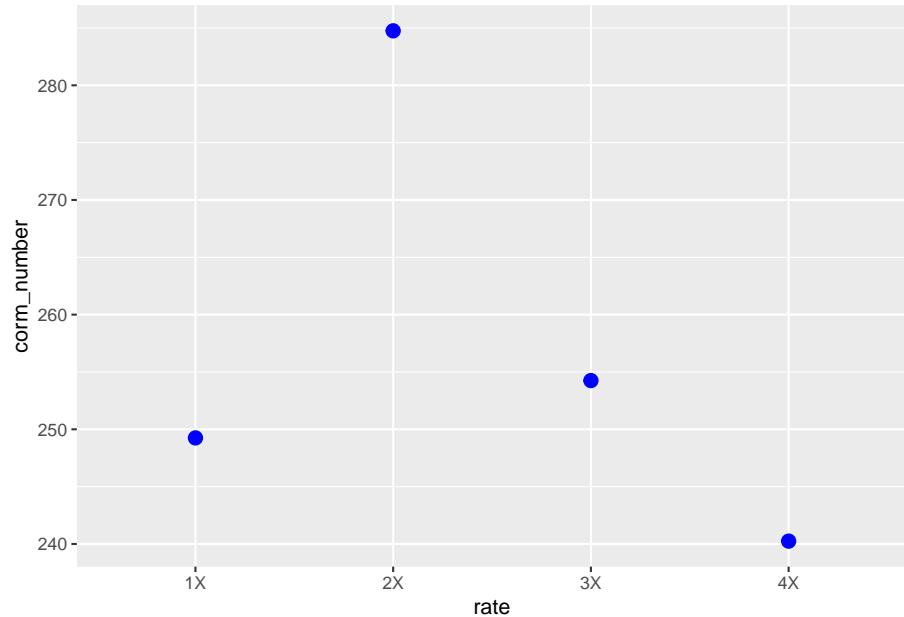
saffron_model_for_contrast = aov(corm_number ~ 0 + rate, saffron)

low_high_vs_middle = glht(saffron_model_for_contrast, linfct=K)
summary(low_high_vs_middle)

##
##  Simultaneous Tests for General Linear Hypotheses
##
## Fit: aov(formula = corm_number ~ 0 + rate, data = saffron)
##
## Linear Hypotheses:
##             Estimate Std. Error t value Pr(>|t|)    
## 1 == 0     24.750     1.984   12.47 3.14e-08 ***
## ---                                                 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
```

We see this estimated difference is even greater than the previous contrast. The significant t-value suggests there may be a parabolic (curved) response of corm number to nutrient solution rate. Indeed, if we plot the mean, we can see a parabolic response

```
saffron_means %>%
  ggplot(aes(x=rate, y=corm_number)) +
  geom_point(size = 3, color="blue")
```



## 8.7 Means Presentation

Something I have always overlooked in teaching this course is also something that is most basic: once we have separated our means, how do we present them to others? There are two ways to present means: in a table, or in a plot. Both have advantages and disadvantages. In a table, you provide raw statistics (treatment means, standard errors, and perhaps, means groupings from an LSD or Tukey test). A reader can use these values to recalculate your statistics, say, if they wanted to separate the means at  $p=0.10$ ,  $p=0.1$ , or  $p=0.001$ .

A figure, one the otherhand, allows the reader to quickly grasp treatment differences, or patterns among treatments. In addition, they are colorful and – lets face it – more inviting than a dense table of numbers.

Whichever format we use, however, we need to present our treatment means and some reference for the reader to gauge whether those means are statistically equal or different.

### 8.7.1 Means Tables

Tables can be tricky, especially when many types of measures are included. If we are reporting results from a factorial experiment, we may be tempted to list the levels of one factor down rows, and the other factor across columns. I would generally discourage this, however, unless required to fit the table neatly into a publication space. Generally, the long form of means presentation is best.

In the long form of data, cases (individual observations or treatment means) are listed down the rows. Measurements from each cases are listed across rows.

For our saffron data, our table would start like this:

```
final_table = saffron_means  
knitr::kable(final_table)
```

rate	corm_number
1X	249.25
2X	284.75
3X	254.25
4X	240.25

To this table, we may wish to add the standard error of the difference. Remember, the standard error of the difference for an LSD or Tukey test is equal to:

$$SED = \sqrt{\frac{(2 \cdot EMS)}{n}}$$

So for our saffron trial, where the error mean square is 15.75 (we can get this from either the ANOVA table or the LSD output) and the number of replications is 4, the standard error of the difference is:

$$SED = \sqrt{\frac{(2 \cdot 15.75)}{4}} = \sqrt{7.875} = 2.80$$

We can add this below the means

```
final_table = saffron_means %>%
  mutate(rate = as.character(rate)) %>%
  rbind(c("SED", 2.80))

knitr::kable(final_table)
```

rate	corm_number
1X	249.25
2X	284.75
3X	254.25
4X	240.25
SED	2.8

It is also important to indicate the number of replications of the treatments. We can add another row to the table with N.

```
final_table = saffron_means %>%
  mutate(rate = as.character(rate)) %>%
  rbind(c("SED", 2.80)) %>%
  rbind(c("N", 4))

knitr::kable(final_table)
```

rate	corm_number
1X	249.25
2X	284.75
3X	254.25
4X	240.25
SED	2.8
N	4

We should add the LSD, to make it easy for readers to compare treatments. We will want also, to include the  $\alpha$  which was used to calculate the LSD. That way, the reader will know whether the risk of a Type I error – that the LSD will separate treatments that are not truly different – is 5% or some other probability.

```
final_table = saffron_means %>%
  mutate(rate = as.character(rate)) %>%
  rbind(c("SED", 2.80)) %>%
  rbind(c("N", 4)) %>%
  rbind(c("LSD (0.05)", 6.11))

knitr::kable(final_table)
```

rate	corm_number
1X	249.25
2X	284.75
3X	254.25
4X	240.25
SED	2.8
N	4
LSD (0.05)	6.11

We might want to include the p-value from the ANOVA table, so the reader knows that the LSD is protected. The p-value for the saffron trial is  $1.28 \times 10^{-8}$ . This is an extremely small number. It is acceptable for us to simplify this in the table, indicating that the probability of F was less than 0.001.

```
final_table = saffron_means %>%
  mutate(rate = as.character(rate)) %>%
  rbind(c("SED", 2.80)) %>%
  rbind(c("N", 4)) %>%
  rbind(c("LSD (0.05)", 6.11)) %>%
  rbind(c("Pr<F", "<0.001"))

knitr::kable(final_table)
```

rate	corm_number
1X	249.25
2X	284.75
3X	254.25
4X	240.25
SED	2.8
N	4
LSD (0.05)	6.11
Pr<F	<0.001

Finally, if imperial (pounds, acres, etc) or metric units were used to measure the response variable, it is important to indicate that in the table. Indicating those in parentheses after the variable name is appropriate.

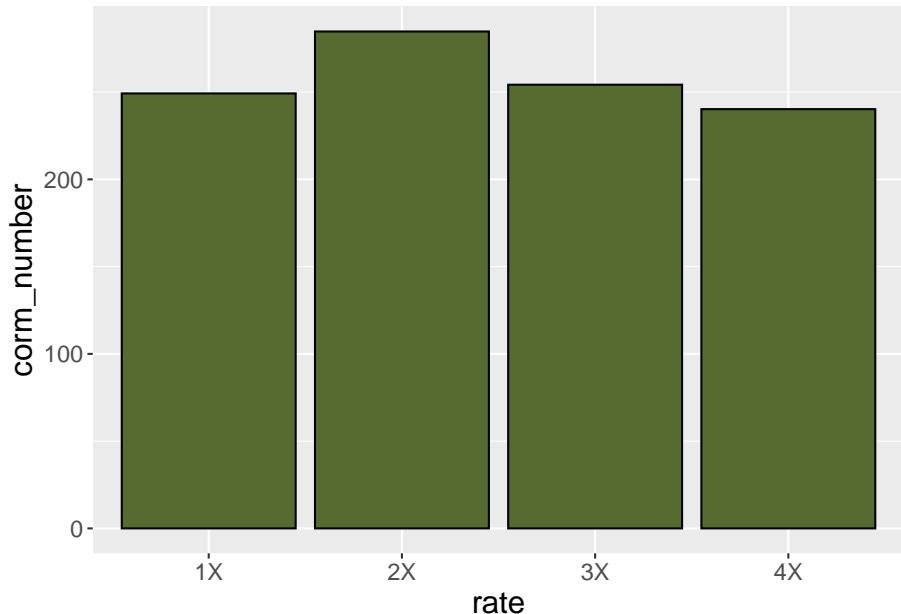
### 8.7.2 Plotting Means

When we work with categorical treatments (that is, treatments levels that are defined by words), or even numerical variables treated as categorical variables (as in the saffron example) we should use a *bar plot*, not a line plot, to visualize the data. An easy way to determine whether a bar plot should be used is this: if you are using an LSD or Tukey's test to separate your means, you should use a bar plot. A line plot, which suggests treatment levels are numerically related

to (higher or lower than) each other should be used to fit regression models, where the analysis defines a continuous relationship between Y and X.

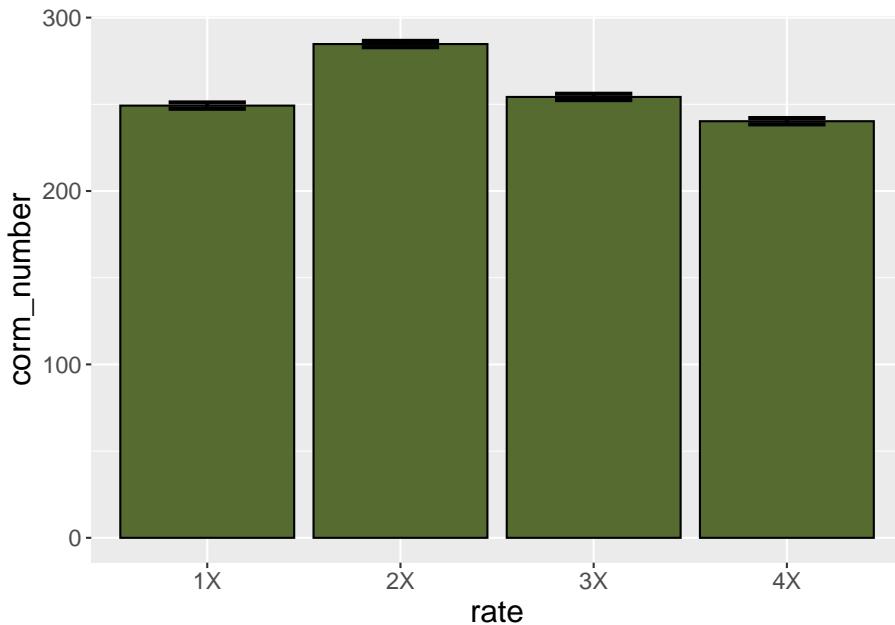
A basic bar plot will have a bar representing each treatment mean. The treatment level is indicated along the x (horizontal) axis. The sample mean is indicated along the y (vertical) axis. The bar height indicates the sample mean.

```
saffron_means %>%
  ggplot(aes(x=rate, y=corm_number)) +
  geom_bar(stat="identity", color="black", fill = "darkolivegreen") +
  theme(axis.text = element_text(size=12),
        axis.title = element_text(size=16))
```



This plot, however, does not provide the viewer any sense of whether corm number is significantly different among treatments. For that purpose, we can add error bars.

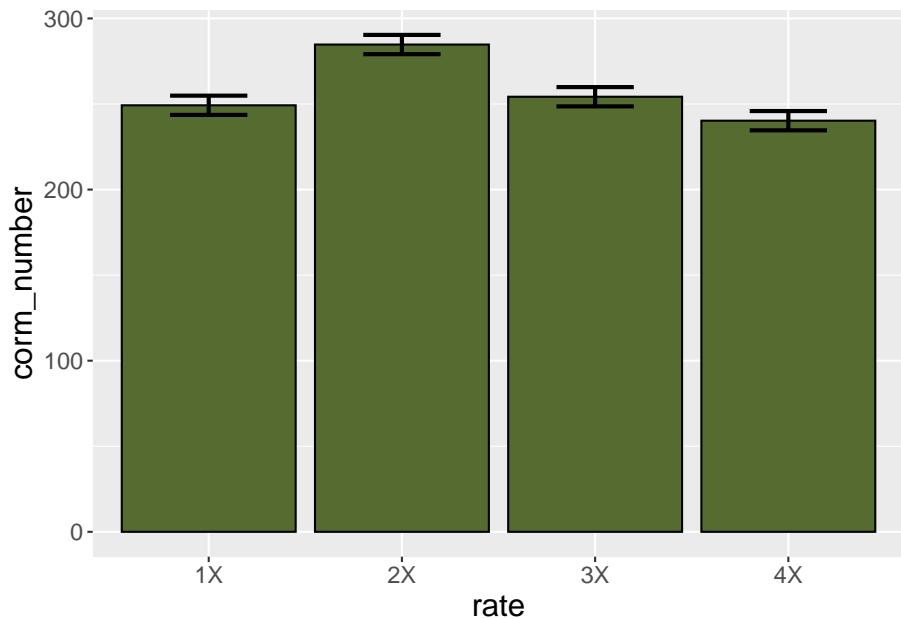
```
saffron_means %>%
  ggplot(aes(x=rate, y=corm_number)) +
  geom_bar(stat="identity", color="black", fill = "darkolivegreen") +
  geom_errorbar(aes(ymin=corm_number-1.81, ymax=corm_number+1.81), width=0.4, size=1) +
  theme(axis.text = element_text(size=12),
        axis.title = element_text(size=16))
```



These error bars stretch from one standard error of the mean below the sample mean to one standard error of the mean above the sample mean. As a general rule of thumb, the least significant difference is approximately 2 standard errors of the mean. If the error bars from two treatment levels don't overlap, then the two treatments are likely different. We can see in the plot that the corm number for the 1X and 3X fertilizer rates are very similar.

Alternatively, we could set the error bar height equal to the least significant difference itself.

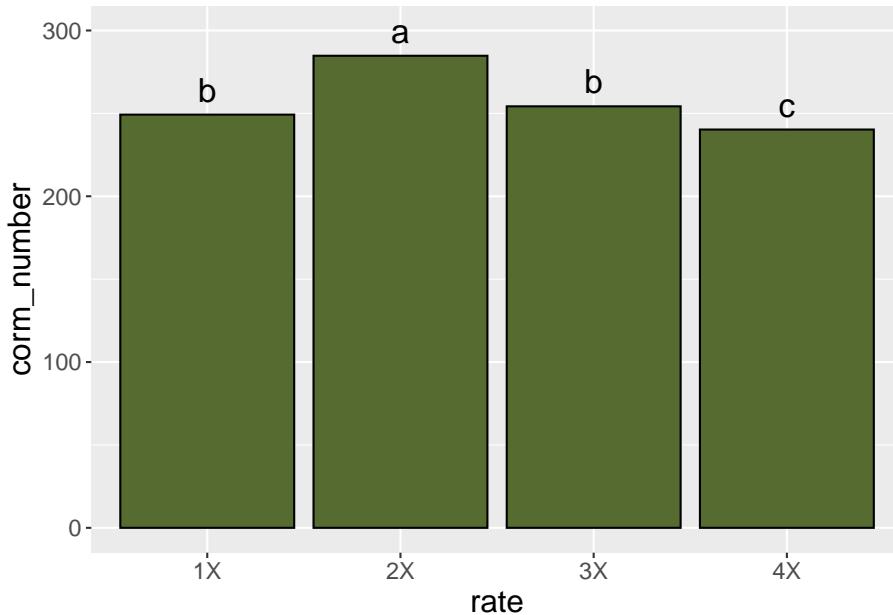
```
saffron_means %>%
  ggplot(aes(x=rate, y=corm_number)) +
  geom_bar(stat="identity", color="black", fill = "darkolivegreen") +
  geom_errorbar(aes(ymin=corm_number-5.63, ymax=corm_number+5.63), width=0.4, size=1) +
  theme(axis.text = element_text(size=12),
        axis.title = element_text(size=16))
```



This allows the viewer to use the actual LSD to separate treatment means. If the range of the error bar from one treatment does not include the mean of another treatment, then the two treatments are not equal.

Significant differences among treatments can also be indicated by including letter groupings from an LSD or Tukey's test.

```
lsd$groups %>%
  rownames_to_column(var="rate") %>%
  ggplot(aes(x=rate, y=corm_number)) +
  geom_bar(stat="identity", color="black", fill = "darkolivegreen") +
  geom_text(aes(x=rate, y=corm_number+15, label=groups), size=6) +
  theme(axis.text = element_text(size=12),
        axis.title = element_text(size=16))
```



We will learn more about graphics in the exercises this unit. A very comprehensive resource for creating plots in R is *R Graphics Cookbook*, by Winston Chang. An online copy is available at <https://r-graphics.org/>. Print copies can also be purchased from common booksellers. This book explains not only how to create plots, but how to adjust labels, legends, axis labels, and so on.

## 8.8 Exercise: LSD and Tukey's HSD

In the lecture, we learned two ways of grouping treatment means: by *Least Significant Difference* and by Tukey's HSD (Honest Significant Difference). Both of these tests can be run very quickly in R.

### 8.8.1 Case Study: Common Bean

The yield of organically-grown common bean was assessed following four crops: conventional broccoli, green\_manure, fallow, and organic broccoli. Yield is in tons per hectare. We wish to separate the treatment means.

```
common_beans = read.csv("data-unit-8/exercise_data/common_beans.csv")
head(common_beans)

##   block      prev_crop fresh_weight
```

```

## 1      1    conv_broccoli     52.15924
## 2      1    green_manure     49.82191
## 3      1 organic_broccoli   36.80926
## 4      1           fallow    54.05342
## 5      2    conv_broccoli     52.79726
## 6      2           fallow    47.68364

```

First, we need to create our linear model and run our analysis of variance. The trial is a randomized complete block design, so our linear model is:

`fresh_weight = mu + block + prev_crop + error`

Our model statement in R is:

```
common_beans_model = aov(fresh_weight ~ block + prev_crop, data=common_beans)
```

Our analysis of variance is:

```
summary(common_beans_model)
```

```

##              Df Sum Sq Mean Sq F value Pr(>F)
## block          1   2.4   2.37   0.179 0.6802
## prev_crop      3 428.9 142.98 10.835 0.0013 **
## Residuals     11 145.1 13.20
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

We see the effect of previous crop is significant, so now we want to separate the treatment means.

### 8.8.2 Least Significant Difference

Both the least significant test and Tukey's Honest Significant Test will be run using functions from the *agricolae* package. We load the *agricolae* package to our session using the *library()* command

```
library(agricolae)
```

After this. We can run the least significant test using the *LSD.test()* function. This argument takes two arguments: a linear model, and the treatment we wish to separate (which must be given in quotes). We can simply reference the `common_beans_model` created above.

```

lsd_common_bean = LSD.test(common_beans_model, "prev_crop")
lsd_common_bean

## $statistics
##   MSerror Df      Mean       CV t.value      LSD
##   13.19522 11 48.04057 7.561363 2.200985 5.653408
##
## $parameters
##           test p.adjusted    name.t ntr alpha
## Fisher-LSD     none prev_crop    4  0.05
##
## $means
##           fresh_weight      std r      LCL      UCL      Min      Max
## conv_broccoli    52.79032 1.242884 4 48.79276 56.78788 51.68091 54.52386
## fallow          50.53382 2.718281 4 46.53625 54.53138 47.68364 54.05342
## green_manure    49.52910 5.135442 4 45.53153 53.52666 44.06679 56.34669
## organic_broccoli 39.30905 3.723475 4 35.31149 43.30662 36.10887 44.26354
##           Q25      Q50      Q75
## conv_broccoli    52.03966 52.47825 53.22891
## fallow          48.93562 50.19911 51.79730
## green_manure    46.92744 48.85145 51.45311
## organic_broccoli 36.63416 38.43190 41.10679
##
## $comparison
## NULL
##
## $groups
##           fresh_weight groups
## conv_broccoli    52.79032     a
## fallow          50.53382     a
## green_manure    49.52910     a
## organic_broccoli 39.30905     b
##
## attr(,"class")
## [1] "group"

```

The test results are returned in a list, which in R is a collection of objects grouped under one name ("lsd\_common\_bean" in our code above). We can reference different parts of this list the same way you would reference columns within a data frame: using the dollar sign to separate the list name and the object. For example, if we just wanted to look at our \$statistics, we would type:

```

lsd_common_bean$statistics

##   MSerror Df      Mean       CV t.value      LSD
##   13.19522 11 48.04057 7.561363 2.200985 5.653408

```

```
##   13.19522 11 48.04057 7.561363 2.200985 5.653408
```

We can reference the means groupings, as well:

```
lsd_common.Bean$groups
```

```
##               fresh_weight groups
## conv_broccoli      52.79032    a
## fallow            50.53382    a
## green_manure      49.52910    a
## organic_broccoli   39.30905    b
```

This selectivity is handy if we are writing a report and just want to reference part of the output.

In the grouping above, we see that common bean fresh weight was statistically similar following conventional broccoli, fallow, and green manure. Common bean fresh weight following organic broccoli, however, was statistically less than following the other three treatments.

### 8.8.3 Tukey HSD

We can separate the means, very similarly, using Tukey's Honest Significant Difference. We use the *HSD.test()* function. The first two arguments are the same as for the *LSD.test()*: the linear model and treatment (in quotes) we wish to separate. We add a third argument, “group = TRUE”. This tells the *HSD.test* to return a letter-grouping of treatment means.

```
common_bean_hsd = HSD.test(common_bean_model, "prev_crop", group = TRUE)
common_bean_hsd

## $statistics
##   MSerror Df     Mean      CV      MSD
##   13.19522 11 48.04057 7.561363 7.730267
##
## $parameters
##   test     name.t ntr StudentizedRange alpha
##   Tukey prev_crop  4        4.256143  0.05
##
## $means
##               fresh_weight      std.r      Min      Max      Q25      Q50
## conv_broccoli      52.79032 1.242884 4 51.68091 54.52386 52.03966 52.47825
## fallow            50.53382 2.718281 4 47.68364 54.05342 48.93562 50.19911
```

```

## green_manure      49.52910 5.135442 4 44.06679 56.34669 46.92744 48.85145
## organic_broccoli 39.30905 3.723475 4 36.10887 44.26354 36.63416 38.43190
##
## Q75
## conv_broccoli    53.22891
## fallow           51.79730
## green_manure     51.45311
## organic_broccoli 41.10679
##
## $comparison
## NULL
##
## $groups
##               fresh_weight groups
## conv_broccoli   52.79032    a
## fallow         50.53382    a
## green_manure   49.52910    a
## organic_broccoli 39.30905    b
##
## attr(,"class")
## [1] "group"

```

The HSD.test output, like the lsd output is a list. We can again reference individual tables within it:

```
common.Bean_hsd$groups
```

```

##               fresh_weight groups
## conv_broccoli   52.79032    a
## fallow         50.53382    a
## green_manure   49.52910    a
## organic_broccoli 39.30905    b

```

### 8.8.4 Practice: Apple

Apple yield was measured after treatment with four biostimulants. The design was a randomized complete block. The yield was measured in tons per hectare.

```
apple = read.csv("data-unit-8/exercise_data/apple.csv")
```

#### 8.8.4.1 Analysis of Variance

Run the analysis of variance. Your results should be:

```
Df Sum Sq Mean Sq F value Pr(>F)

block 1 0.40 0.40 0.076 0.78800
foliar_trt 3 162.10 54.03 10.345 0.00156 ** Residuals 11 57.45 5.22
Signif. codes: 0 ‘‘ 0.001 ‘‘ 0.01 ‘‘ 0.05 ‘‘ 0.1 ‘‘ 1
```

#### 8.8.4.2 Least Significant Difference

Run the Least Significant Difference test on the treatment means. Your results should look like:

```
$statistics MSerror Df Mean CV t.value LSD 5.222995 11 47.96522 4.764676
2.200985 3.55682
$parameters test p.adjusted name.t ntr alpha Fisher-LSD none foliar_trt 4 0.05
$means yield std r LCL UCL Min Max Q25 Q50 Q75 conv 45.70984 1.678378 4
43.19478 48.22489 44.44665 48.15233 44.71817 45.12018 46.11185 conv+calcium
44.08293 1.274706 4 41.56788 46.59798 42.34804 45.30697 43.58313 44.33835
44.83815 conv+seaweed 51.90854 2.073312 4 49.39349 54.42359 50.18903
54.78184 50.51099 51.33165 52.72920 conv+si+zn 50.15958 3.246956 4 47.64452
52.67463 46.88179 54.60709 48.55789 49.57471 51.17639
$comparison NULL
$groups yield groups conv+seaweed 51.90854 a conv+si+zn 50.15958 a conv
45.70984 b conv+calcium 44.08293 b
attr(“class”) [1] “group”
```

#### 8.8.4.3 Tukey’s HSD

Run the Honest Significant Difference test on the treatment means. Your results should look like:

```
$statistics MSerror Df Mean CV MSD 5.222995 11 47.96522 4.764676 4.863468
$parameters test name.t ntr StudentizedRange alpha Tukey foliar_trt 4
4.256143 0.05
$means yield std r Min Max Q25 Q50 Q75 conv 45.70984 1.678378 4
44.44665 48.15233 44.71817 45.12018 46.11185 conv+calcium 44.08293 1.274706
4 42.34804 45.30697 43.58313 44.33835 44.83815 conv+seaweed 51.90854
2.073312 4 50.18903 54.78184 50.51099 51.33165 52.72920 conv+si+zn 50.15958
3.246956 4 46.88179 54.60709 48.55789 49.57471 51.17639
$comparison NULL
$groups yield groups conv+seaweed 51.90854 a conv+si+zn 50.15958 ab conv
45.70984 bc conv+calcium 44.08293 c
```

```
attr("class") [1] "group"
```

### 8.8.5 Practice: Wheat Treatment with Mildew

Four foliar treatments against powdery mildew in wheat were tested in Rothemsted, England, for their effect on grain yield. The same fungicide was tested with four timings: 0 (none), 1 (early), 2 (late), R (repeated).

The treatment was a randomized complete block design. Yield was measured in tons per hectare.

```
wheat_mildew = read.csv("data-unit-8/exercise_data/mildew.csv")
head(wheat_mildew)
```

```
##   plot trt block yield
## 1     1   T2    B1  5.73
## 2     2     R    B1  6.08
## 3     3   T0    B1  5.26
## 4     4   T1    B1  5.89
## 5     5   T0    B2  5.37
## 6     6   T2    B2  5.95
```

#### 8.8.5.1 Analysis of Variance

Run the analysis of variance on the response of wheat yield to fungicide treatment. Your results should look like:

```
Df Sum Sq Mean Sq F value    Pr(>F)
block 8 5.085 0.6356 17.53 2.79e-08  trt 3 3.129 1.0432 28.77 4.05e-08
Residuals 24 0.870 0.0363
Signif. codes: 0 ‘’ 0.001 ‘‘ 0.01 ‘‘ 0.05 ‘‘ 0.1 ‘ ’ 1
```

#### 8.8.5.2 Least Significant Difference

Run the Least Significant Difference test on the treatment means. Your results should look like:

```
$statistics MSerror Df Mean CV t.value LSD 0.03625532 24 5.801944 3.281802
2.063899 0.1852542
```

```
$parameters test p.adjusted name.t ntr alpha Fisher-LSD none trt 4 0.05
```

```
$means yield std r LCL UCL Min Max Q25 Q50 Q75 R 5.942222 0.4649403 9
5.811228 6.073217 5.06 6.54 5.76 6.01 6.18 T0 5.310000 0.4519956 9 5.179006
5.440994 4.38 5.82 5.16 5.26 5.73 T1 5.867778 0.4602928 9 5.736783 5.998772
5.04 6.45 5.59 5.89 6.26 T2 6.087778 0.3347304 9 5.956783 6.218772 5.63 6.48
5.76 6.14 6.43

$comparison NULL

$groups yield groups T2 6.087778 a R 5.942222 ab T1 5.867778 b T0 5.310000
c

attr(“class”) [1] “group”
```

### 8.8.5.3 Tukey’s Honest Significant Difference Test

Run the HSD test on the treatment means. Your results should look like:

```
$statistics MSerror Df Mean CV MSD 0.03625532 24 5.801944 3.281802
0.2476109

$parameters test name.t ntr StudentizedRange alpha Tukey trt 4 3.901262 0.05

$means yield std r Min Max Q25 Q50 Q75 R 5.942222 0.4649403 9 5.06 6.54
5.76 6.01 6.18 T0 5.310000 0.4519956 9 4.38 5.82 5.16 5.26 5.73 T1 5.867778
0.4602928 9 5.04 6.45 5.59 5.89 6.26 T2 6.087778 0.3347304 9 5.63 6.48 5.76 6.14
6.43

$comparison NULL

$groups yield groups T2 6.087778 a R 5.942222 a T1 5.867778 a T0 5.310000 b
```

## 8.9 Exercise: Linear Contrasts

Linear contrasts are different from least significant difference (LSD) or Tukey’s honest significant difference (HSD) tests in that they can test groups of treatments. As we learned in the lecture, the simplest way to think of a linear contrast is as a t-test between two groups.

We also learned in the lecture that we use contrast coefficients to define these groups. The coefficients for one group should add up to 1; the coefficients of the other group should add up to -1. That way, we subtract the average value for treatments in one group from the average value for treatments in the other group.

### 8.9.1 Case Study: Winter Canola Cultivar Trial.

Six winter canola cultivars were tested in a randomized complete block trial in Manitoba. Yield is in kg / ha.

```
canola = read.csv("data-unit-8/exercise_data/canola_gd.csv")
head(canola)
```

```
##   block cultivar     yield
## 1      1      Bob 1508.424
## 2      2      Bob 1962.072
## 3      3      Bob 1364.861
## 4      4      Bob 1832.715
## 5      1    Donna 1212.190
## 6      2    Donna 1629.680
```

### 8.9.1.1 ANOVA

First, let's run our analysis of variance

```
canola_model = aov(yield ~ block + cultivar, data = canola)
summary(canola_model)
```

```
##              Df  Sum Sq Mean Sq F value    Pr(>F)
## block          1 18600   18600   0.171 0.684822
## cultivar       5 4551035  910207   8.344 0.000388 ***
## Residuals     17 1854462  109086
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

### 8.9.1.2 Least Significant Difference Test

Next, lets run an LSD test.

```
library(agricolae)
lsd_canola = LSD.test(canola_model, "cultivar")
lsd_canola
```

```
## $statistics
##   MSerror Df      Mean       CV  t.value      LSD
##   109086 17 2296.983 14.37894 2.109816 492.7357
##
## $parameters
##           test p.adjusted   name.t ntr alpha
## Fisher-LSD      none cultivar   6  0.05
##
## $means
```

```

##          yield     std r      LCL      UCL    Min    Max    Q25
## Bill    2296.268 326.6421 4 1947.851 2644.685 1865.903 2614.081 2142.888
## Bob     1667.018 277.4666 4 1318.601 2015.435 1364.861 1962.072 1472.533
## Donna   1833.282 514.0721 4 1484.865 2181.699 1212.190 2358.635 1525.308
## Jerry   2669.161 176.6376 4 2320.745 3017.578 2545.158 2923.794 2552.218
## Mickey  2406.509 263.5791 4 2058.092 2754.926 2130.413 2656.553 2207.614
## Phil    2909.658 275.1842 4 2561.242 3258.075 2565.683 3218.583 2776.782
##          Q50      Q75
## Bill    2352.545 2505.925
## Bob     1670.570 1865.055
## Donna   1881.152 2189.126
## Jerry   2603.847 2720.790
## Mickey  2419.535 2618.430
## Phil    2927.183 3060.060
##
## $comparison
## NULL
##
## $groups
##          yield groups
## Phil    2909.658      a
## Jerry   2669.161      ab
## Mickey  2406.509      b
## Bill    2296.268      bc
## Donna   1833.282      cd
## Bob     1667.018      d
##
## attr(,"class")
## [1] "group"

```

### 8.9.1.3 Contrasts

Contrasts should be based on specific questions we want to ask of the data. In this case, there are two questions we would like to answer with contrasts.

First, cultivars “Donna” and “Bob” are from a line of canola called “PITB”. Sometimes they perform well, but other times their performance can be rather discordant. Our first question is, do cultivars from the PITB line perform worse than cultivars from other lines.

Cultivars “Mickey” and “Bill” are from the “FOTM” line of cultivars. They regularly perform better than cultivars with “PITB” pedigrees. But in bad seasons they have a field performance that has been likened to “sneakers in a drier”. Meanwhile, cultivars “Phil” and “Jerry”, from the DKSTR line, under appropriate growing conditions, have a performance potential that is almost unlimited. Our second question, then, is whether cultivars from the DKSTR

outperform those from the FOTM line.

#### 8.9.1.4 Figuring Out Which Coefficients to Use

To answer these questions, we need to define our coefficients. The order of the coefficients must reflect the order of the factor levels in R. To determine this order we use the *levels()* function.

```
levels(canola$cultivar)
## NULL
```

What coefficients do we use, then? Our first hypothesis is that “Bob” and “Donna” perform worse than others.

- 1) So we will assign a -1 to both those cultivars.

Coefficients: 0 -1 -1 0 0 0

- 2) Since we are comparing against the other four cultivars, we assign them +1:

Coefficients: +1 -1 -1 +1 +1 +1

- 3) The coefficients of each group must sum to 1. So we will divide the coefficients for Bob and Donna by 2:

Coefficients: +1 -1/2 -1/2 +1 +1 +1

- 4) And we will divide the coefficients for the other cultivars by 4:

Coefficients: +1/4 -1/2 -1/2 +1/4 +1/4 +1/4

- 5) Now, let’s check our math. The coefficients for Bob and Donna should sum to -1

Coefficients: 0 -1/2 -1/2 0 0 0 = -1

- 6) The coefficients for the other four cultivars should sum to 1.

Coefficients: +1/4 0 0 +1/4 +1/4 +1/4 = 1

- 7) Finally, all the coefficients together should sum to zero:

Coefficients: +1/4 -1/2 -1/2 +1/4 +1/4 +1/4 = 0

### 8.9.1.5 Telling R the Coefficients to Use

Running the linear coefficient in R requires we define a “coefficient matrix”. A matrix is a slightly more primitive version of the data.frame – still akin to a table, but its rows and columns tend to be defined by numbers instead of names.

To create a coefficient matrix in R, we need to define it as follows:

```
K = matrix(c(insert coefficients here),1)
```

“c(coefficients)” tells R to fill in the coefficients across columns. The “,1” that follows tells R to put those values in the first row. That second part of the matrix, the “,1” is *really important*. In creating this lesson, I spent at least a half-hour trying to figure out why one of my contrasts would not work. It was because I had written it:

```
K_bd_vs_others = matrix(c(+1/4, -1/2, -1/2, +1/4, +1/4, +1/4))
```

Instead of:

```
K_bd_vs_others = matrix(c(+1/4, -1/2, -1/2, +1/4, +1/4, +1/4), 1)
```

```
K_bd_vs_others = matrix(c(+1/4, -1/2, -1/2, +1/4, +1/4, +1/4), 1)
```

```
K_bd_vs_others
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,] 0.25 -0.5 -0.5 0.25 0.25 0.25
```

Above is our coefficient matrix for “Bob” and “Donna” versus the others.

Now let’s create our second matrix, for “Mickey” and “Bill” versus “Phil” and “Jerry”. Let’s check the order of the factor levels again.

```
levels(canola$cultivar)
```

```
## NULL
```

Our hypothesis is that “Phil” and “Jerry” perform better than “Mickey” and “Bill”, so we will assign them the initial coefficients of 1

- 1) So we will assign a -1 to both those cultivars.

Coefficients: 0 0 0 1 0 1

- 2) We then assign Mickey and Bill the coefficients -1:

Coefficients: -1 0 0 +1 -1 +1

“Donna” and “Bob” are assigned the coefficient zero since they are not involved in this comparison.

- 3) The coefficients of each group must sum to 1. So we will divide the coefficients for “Jerry” and “Phil” by 2:

Coefficients: -1 0 0 +1/2 -1 +1/2

- 4) We do the same with “Mickey” and “Bill”:

Coefficients: -1/2 0 0 +1/2 -1/2 +1/2

- 5) Now, let’s check our math. The coefficients for Jerry and Phil should sum to 1

Coefficients: -0 0 0 +1/2 0 +1/2 = 1

- 6) The coefficients for Mickey and Bill should sum to -1.

Coefficients: -1/2 0 0 0 -1/2 0 = -1

Finally, all the coefficients together should sum to zero:

Coefficients: -1/2 0 0 +1/2 -1/2 +1/2 = 0

Our correlation coefficient for our second contrast, then, is:

```
K_jp_vs_mb = matrix(c(-1/2, 0, 0, +1/2, -1/2, +1/2), 1)
K_jp_vs_mb
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,] -0.5   0    0   0.5 -0.5  0.5
```

### 8.9.2 Running the Contrast

Defining the contrast coefficients is difficult. Running the contrast afterwards is relatively easy. We use the *glht()* function from the *multicomp()* package. “glht” stands for “generalized linear hypothesis test”.

*glht()* requires two arguments: a linear model, and the coefficient contrast we created above. The linear model is a little different than what we defined for the ANOVA above. It only contains the terms “0” and the treatment name:

```
canola_contrast_model = aov(yield ~ 0 + cultivar, data=canola)
```

The reason for this is because our contrast is isolating the effect of cultivar. The zero forces our contrast to calculate the actual difference between the two groups' means (which we may like to report).

Now we can run our contrast

```
library(multcomp)
bd_vs_others = glht(canola_contrast_model, linfct = K_bd_vs_others)
```

We will use *summary()* to summarise these results.

```
summary(bd_vs_others)
```

```
##
##  Simultaneous Tests for General Linear Hypotheses
##
## Fit: aov(formula = yield ~ 0 + cultivar, data = canola)
##
## Linear Hypotheses:
##             Estimate Std. Error t value Pr(>|t|)
## 1 == 0     820.2      139.7   5.872 1.47e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
```

Ok, let's review our results. The Estimate, 820.2 kg / ha, is the difference between the group means. Since we subtracted the mean of Bob and Donna from the mean of the other four cultivars, we concluded that Bob and Donna as a group yielded *less* than the other four cultivars as a group.

The Estimate was divided by Std. Error, the standard error of the difference, to produce the t value of 5.872. The probability of observing a t-value of this size or greater, if in fact the true difference between groups was zero, is very small,  $1.47 \times 10^{-5}$ . The difference between groups is significant.

We conclude that cultivars from the PITB line yielded worse than other cultivars as a group.

We can now run the second contrast, Jerry and Phil versus Mickey and Bill. We again use the canola\_contrast\_model as our linear model, and this time, the coefficient matrix K\_jp\_vs\_mb

```
jp_vs_mb = glht(canola_contrast_model, linfct=K_jp_vs_mb)
summary(jp_vs_mb)

##
##  Simultaneous Tests for General Linear Hypotheses
##
## Fit: aov(formula = yield ~ 0 + cultivar, data = canola)
##
## Linear Hypotheses:
##          Estimate Std. Error t value Pr(>|t|)
## 1 == 0    438.0     161.3   2.716  0.0142 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
```

The mean of Jerry and Phill was 438.0 kg/ha greater than the mean of Mickey and Bill. (Note: when you look at the Estimate, always ask yourself whether the number, based on treatment means, makes sense. The first time I ran this contrast, for example, the estimate was greater than 2000. This caused me to review my contrast coefficients, where I discovered one of the coefficients was positive when it should have been negative.)

Our t value again has a probability of occurring by chance of less than P=0.05, so the difference between the mean of Jerry and the mean of Phil and Mickey and Bill is significant. We conclude that cultivars from the DKSTR line yielded better than cultivars from the FOTM line.

### 8.9.3 Practice: Corn Nitrogen Source and Timing

This is a really powerful example of how contrasts can be used, not just to test significances, but generate much broader understandings than either the ANOVA or LSD/HSD alone would allow. Our practice data set is from a nitrogen management trial was conducted near Whitehouse, Ohio, to compare 7 nitrogen management strategies in corn. All treatments (other than the control) provided 200 units (lbs) total N during the growing season.

```
n_source_timing = read.csv("data-unit-8/exercise_data/corn_nitrogen_source_timing.csv")
head(n_source_timing)

##
##      trt      yield block  B      Error
## 1 control 147.9109    B1 -4  3.9108674
## 2 aa_pre  178.1029    B1 -4 -1.8970790
## 3 aa_post 173.5400    B1 -4  5.5400054
```

```
## 4 aa_pre_post 184.2516    B1 -4 -2.7483991
## 5      uan_pre 170.4471    B1 -4 -0.5529385
## 6      uan_post 165.4751    B1 -4 -1.5249071
```

Lets look at the seven levels of nitrogen treatment:

```
levels(n_source_timing$trt)
```

```
## NULL
```

Treatment abbreviations are: “aa” = anhydrous ammonia, “uan” = urea ammonium nitrate, “pre” = pre-plant, and “post” = post-emergence (sidedress).

### 8.9.3.1 ANOVA

First, let’s look at our analysis of variance:

```
corn_nitrogen_model = aov(yield ~ block + trt, data=n_source_timing)
summary(corn_nitrogen_model)
```

```
##           Df Sum Sq Mean Sq F value    Pr(>F)
## block       3   283    94.3   2.414     0.1
## trt         6  3421   570.1  14.591 4.81e-06 ***
## Residuals  18   703    39.1
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Our nitrogen treatments are significantly different. What does the least significant test tell us differences among treatment means?

```
lsd = LSD.test(corn_nitrogen_model, "trt")
lsd$groups
```

```
##           yield groups
## aa_pre_post 191.2750    a
## aa_pre      181.7686    b
## uan_pre_post 179.3731    b
## uan_pre     175.9014    bc
## aa_post     174.7789    bc
## uan_post     167.9677    c
## control      153.2038    d
```

We have four questions: 1) is the mean yield of the six nitrogen treatments significantly different from the untreated control?

- 2) is the mean yield of preplant nitrogen treatments significantly different from the mean of postemergence nitrogen treatments?
- 3) is the mean yield of anhydrous ammonia treatments significantly different from the mean of urea ammonium nitrate treatments?
- 4) is the mean yield of split application treatments significantly different from the mean of single application treatments?

#### 8.9.3.2 Nitrogen Treatments vs Control

Lets walk through this one together. What contrast coefficients do we use? Lets first double-check the order of factor levels (treatments):

```
levels(n_source_timing$trt)
```

```
## NULL
```

So we will assign -1 to the control treatment and 1 to all other treatments:

coefficients: 1 1 1 -1 1 1 1

The coefficients assigned to each group should add up to an absolute value. Since there are six nitrogen treatments, we need to divide each of their coefficients by six.

coefficients: 1/6 1/6 1/6 -1 1/6 1/6 1/6

Now, if we check, we will see the coefficients assigned to the six nitrogen treatments sum to 1, the control coefficient is zero, and the sum of all coefficients is 0.

Now let's define our first contrast coefficient matrix by filling in the matrix below:

```
K_n_vs_control = matrix(c(1/6, 1/6, 1/6, -1, 1/6, 1/6, 1/6), 1) # don't forget the ", 1" after t
```

We need to define our nitrogen contrast model. Remember, the model only has 0 and the factor name on the right side of the equation. Complete our equation below.

```
n_contrast_model = aov(yield ~ 0 + trt, data=n_source_timing)
```

Finally, run the contrast using the glht() and summary() functions below:

```
n_vs_control = glht(n_contrast_model, linfct=K_n_vs_control)
summary(n_vs_control)

##
##  Simultaneous Tests for General Linear Hypotheses
##
## Fit: aov(formula = yield ~ 0 + trt, data = n_source_timing)
##
## Linear Hypotheses:
##             Estimate Std. Error t value Pr(>|t|)
## 1 == 0     25.307      3.701   6.838 9.26e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
```

What do you conclude?

### 8.9.3.3 Preplant vs Postemergence Treatments

Compare the two pre-plant treatments (“\_pre”) to the two post-emergence (“\_post”) treatments. Your results should look like:

```
K_pre_vs_post = matrix(c(1/2,-1/2,0,0,1/2,-1/2,0),1)
pre_vs_post = glht(n_contrast_model, K_pre_vs_post)
summary(pre_vs_post)

##
##  Simultaneous Tests for General Linear Hypotheses
##
## Fit: aov(formula = yield ~ 0 + trt, data = n_source_timing)
##
## Linear Hypotheses:
##             Estimate Std. Error t value Pr(>|t|)
## 1 == 0     -7.462      3.427  -2.178   0.041 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
```

Simultaneous Tests for General Linear Hypotheses

Fit: aov(formula = yield ~ 0 + trt, data = n\_source\_timing)  
 Linear Hypotheses: Estimate Std. Error t value Pr(>|t|)  
 $1 == 0$  -7.462 3.427 -2.178 0.041 \* Signif. codes: 0 ‘**0.001**’ ‘0.01’ ‘0.05’ ‘0.1’ ‘1’ (Adjusted p values reported – single-step method)

### 8.9.3.4 Anhydrous Ammonium vs Urea Ammonium Nitrate

Compare the three anhydrous ammonia (“aa\_”) treatments to the three urea ammonium nitrate (“\_uan”) treatments. Your results should look like:

```
K_aa_vs_uan = matrix(c(1/3,1/3,1/3,0,-1/3,-1/3,-1/3),1)
aa_vs_uan = glht(n_contrast_model, K_aa_vs_uan)
summary(aa_vs_uan)

## 
##   Simultaneous Tests for General Linear Hypotheses
##
## Fit: aov(formula = yield ~ 0 + trt, data = n_source_timing)
##
## Linear Hypotheses:
##           Estimate Std. Error t value Pr(>|t|)
## 1 == 0     8.193     2.798   2.929  0.00803 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)

Simultaneous Tests for General Linear Hypotheses

Fit: aov(formula = yield ~ 0 + trt, data = n_source_timing)

Linear Hypotheses: Estimate Std. Error t value Pr(>|t|)
1 == 0 8.193 2.798 2.929 0.00803 ** Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 (Adjusted p values reported - single-step method)
```

### 8.9.3.5 Split vs Single Applications

Compare the two split application (“\_pre\_post”) treatments to the four single-application (“\_pre” or “\_post”) treatments. Your results should look like:

```
K_split_vs_single = matrix(c(-1/4, -1/4, 1/2,0,-1/4,-1/4,1/2),1)
split_vs_single = glht(n_contrast_model, K_split_vs_single)
summary(split_vs_single)

## 
##   Simultaneous Tests for General Linear Hypotheses
##
## Fit: aov(formula = yield ~ 0 + trt, data = n_source_timing)
##
## Linear Hypotheses:
```

```

##          Estimate Std. Error t value Pr(>|t|)
## 1 == 0    10.220     2.967   3.444  0.00243 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)

```

#### Simultaneous Tests for General Linear Hypotheses

Fit: aov(formula = yield ~ 0 + trt, data = n\_source\_timing)

Linear Hypotheses: Estimate Std. Error t value Pr(>|t|)  
 $1 == 0$  10.220 2.967 3.444 0.00243 \*\*

Signif. codes: 0 ‘**0.001**’ ‘0.01’ ‘0.05’ ‘0.1’ ‘1’ (Adjusted p values reported – single-step method)

attr(“class”) [1] “group”

## 8.10 Exercise: Means Tables

Rarely have I seen a statistics course that addresses how to present your summarise your data for others to use. Sure, you learn about tests and P-values. But how do you summarise your data for a report publication or presentation? I don’t want to go too deep in the weeds, but I want you to be aware how to organize your data, and how R can be used to present it.

### 8.10.1 Case Study: Corn Nitrogen Source and Timing

This is the same dataset we used for practice in the linear contrasts exercise. Seven N fertilizer treatments (6 combinations of source and timing, plus one unfertilized control) were tested in a randomized complete block trial near Whitehouse, Ohio.

```

corn_n = read.csv("data-unit-8/exercise_data/corn_nitrogen_source_timing.csv")
head(corn_n)

```

	trt	yield	block	B	Error
## 1	control	147.9109	B1	-4	3.9108674
## 2	aa_pre	178.1029	B1	-4	-1.8970790
## 3	aa_post	173.5400	B1	-4	5.5400054
## 4	aa_pre_post	184.2516	B1	-4	-2.7483991
## 5	uan_pre	170.4471	B1	-4	-0.5529385
## 6	uan_post	165.4751	B1	-4	-1.5249071

### 8.10.1.1 Calculating Means

We can quickly calculate means using the `LSD.test()` function from the *agricolae* package.

```

corn_n_model = aov(yield ~ block + trt, data=corn_n)
summary(corn_n_model)

##          Df Sum Sq Mean Sq F value    Pr(>F)
## block      3   283   94.3   2.414     0.1
## trt        6  3421  570.1  14.591 4.81e-06 ***
## Residuals 18   703   39.1
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

library(agricolae)
lsd = LSD.test(corn_n_model, "trt")
lsd

## $statistics
##      MSerror Df      Mean       CV t.value      LSD
## 39.07036 18 174.8955 3.573922 2.100922 9.285785
##
## $parameters
##      test p.adjusted name.t ntr alpha
## Fisher-LSD    none      trt    7  0.05
##
## $means
##           yield      std.r      LCL      UCL      Min      Max      Q25
## aa_post    174.7789 3.277009 4 168.2129 181.3449 170.7412 177.8955 172.8403
## aa_pre     181.7686 7.217879 4 175.2026 188.3347 174.6550 191.2908 177.2409
## aa_pre_post 191.2750 5.197261 4 184.7089 197.8410 184.2516 196.7370 189.6756
## control    153.2038 7.242153 4 146.6378 159.7699 147.6271 163.0634 147.8399
## uan_post   167.9677 1.734666 4 161.4016 174.5337 165.4751 169.4693 167.5775
## uan_pre    175.9014 12.448641 4 169.3354 182.4675 162.5066 191.5746 168.4619
## uan_pre_post 179.3731 5.336127 4 172.8070 185.9391 171.7472 184.1562 178.2194
##           Q50      Q75
## aa_post    175.2394 177.1780
## aa_pre     180.5644 185.0921
## aa_pre_post 192.0556 193.6550
## control    151.0624 156.4262
## uan_post   168.4631 168.8533
## uan_pre    174.7623 182.2018
## uan_pre_post 180.7945 181.9482
##
```

```

## $comparison
## NULL
##
## $groups
##           yield groups
## aa_pre_post 191.2750      a
## aa_pre       181.7686      b
## uan_pre_post 179.3731      b
## uan_pre      175.9014      bc
## aa_post      174.7789      bc
## uan_post     167.9677      c
## control      153.2038      d
##
## attr(),"class")
## [1] "group"

```

We can create a separate table with just the \$groups section:

```

trt_means=lsd$groups
trt_means

```

```

##           yield groups
## aa_pre_post 191.2750      a
## aa_pre       181.7686      b
## uan_pre_post 179.3731      b
## uan_pre      175.9014      bc
## aa_post      174.7789      bc
## uan_post     167.9677      c
## control      153.2038      d

```

One thing we need to fix are the row\_names. These are our treatment names and we want them in a column for our final table. The *rownames\_to\_column()* argument quickly fixes this (and is a trick it took me quite a while to figure out!). The *rownames\_to\_column* function takes one argument, *vars* = “”, which tells R what to name the new column.

```

table_w_trt_column = trt_means %>%
  rownames_to_column(var = "trt")

table_w_trt_column

```

```

##           trt   yield groups
## 1  aa_pre_post 191.2750      a
## 2      aa_pre 181.7686      b

```

```
## 3 uan_pre_post 179.3731      b
## 4      uan_pre 175.9014      bc
## 5      aa_post 174.7789      bc
## 6      uan_post 167.9677      c
## 7      control 153.2038      d
```

### 8.10.1.2 Additional Statistics to Include

It would be good to add some additional statistics to this, particularly the standard error of the difference and the number of replicatons. We can calculate the standard error of the difference from the Error Mean Square given in the LSD output. First, let's start by retrieving the error mean square from the \$statistics section of our lsd output. This is a case where we can use multiple dollar signs to drill down through the list and dataframe within the list, to get to the individual statistic.

```
ems = lsd$statistics$MSerror
ems
```

```
## [1] 39.07036
```

Here is another important trick. Analyses of variance and LSD outputs rarely report the standard error of the difference. But we can quickly calculate it if we know the error mean square and the number of replicates:

$$\text{SED} = \sqrt{2 * \text{EMS} / r}$$

Where r is the number of replicates. We will set r as a variable and call it into the equation (that way we can call it into our table later as well.)

```
r = 4
sed = sqrt(2 * ems/r)
sed
```

```
## [1] 4.419862
```

Now we can add the standard error of the difference and the number of replicates to the bottom of the column yield. We will put the name of the statistic in the “treatment” column and the value of the statistic in the “yield” column. All we need to do is use the *add\_row()* function and tell R what the value of treatment and yield are in this new row.

```
table_w_stats = table_w_trt_column %>%
  add_row(trt="SED", yield=sed) %>%
  add_row(trt="Number of Reps", yield=r)

table_w_stats
```

	trt	yield	groups
## 1	aa_pre_post	191.274962	a
## 2	aa_pre	181.768650	b
## 3	uan_pre_post	179.373088	b
## 4	uan_pre	175.901428	bc
## 5	aa_post	174.778906	bc
## 6	uan_post	167.967676	c
## 7	control	153.203813	d
## 8	SED	4.419862	<NA>
## 9	Number of Reps	4.000000	<NA>

### 8.10.1.3 Making a Presentation-Quality Table

At this point, our table content is complete. If we are going to include this in a document or presentation, however, we need to spruce it up a little. While there are multiple packages to create tables in R, I think that the *gt* table is one you are most likely to use.

We'll start loading the *gt* package, then using the *gt()* function to flow in the means table we have created above. This will create a preliminary table.

```
library(gt)

## Warning: package 'gt' was built under R version 4.1.3

gt_tbl = gt(table_w_stats)

gt_tbl
```

	trt	yield	groups
	aa_pre_post	191.274962	a
	aa_pre	181.768650	b
	uan_pre_post	179.373088	b
	uan_pre	175.901428	bc
	aa_post	174.778906	bc
	uan_post	167.967676	c

control	153.203813	d
SED	4.419862	NA
Number of Reps	4.000000	NA

**8.10.1.3.1 Choose Decimal Places** One of the first things we notice is that the means in our middle column, yield, has have way too many decimal places. We can correct this with the *fmt\_number()* function. This function takes two arguments. First, “columns - vars(yield) tells R which column to format. The second argument,”decimals=1”, tells the number of decimal places to round to.

```
gt_tbl %>%
  fmt_number(columns = vars(yield), decimals = 1)

## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead

## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead
```

trt	yield	groups
aa_pre_post	191.3	a
aa_pre	181.8	b
uan_pre_post	179.4	b
uan_pre	175.9	bc
aa_post	174.8	bc
uan_post	168.0	c
control	153.2	d
SED	4.4	NA
Number of Reps	4.0	NA

**8.10.1.3.2 Adjust Table Width** Our table looks a little skinny, so let’s adjust the width. We can do this with the *cols\_width* function. The “everything()” argument tells R to set each of the columns to the same width.”~200px” sets the width at 200px.

```
gt_tbl %>%
  fmt_number(columns = vars(yield), decimals = 1) %>%
  cols_width(everything() ~ "200px")

## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
```

```
## * please use `columns = c(...)` instead

## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead
```

trt	yield	groups
aa_pre_post	191.3	a
aa_pre	181.8	b
uan_pre_post	179.4	b
uan_pre	175.9	bc
aa_post	174.8	bc
uan_post	168.0	c
control	153.2	d
SED	4.4	NA
Number of Reps	4.0	NA

**8.10.1.3.3 Align Columns** We use the `cols_align()` function to align our columns. The first argument to this function, “`cols_align =`”, specifies whether the column(s) should be aligned to the left, center, or right. The second argument, “`columns = vars()`” tells R which columns to which the alignment should be applied.

```
gt_tbl %>%
  fmt_number(columns = vars(yield), decimals = 1) %>%
  cols_width(everything() ~ "200px") %>%
  cols_align(align = "left", columns = vars(trt)) %>%
  cols_align(align = "center", columns = vars(yield, groups))

## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead

## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead

## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead

## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead
```

trt	yield	groups
-----	-------	--------

aa_pre_post	191.3	a
aa_pre	181.8	b
uan_pre_post	179.4	b
uan_pre	175.9	bc
aa_post	174.8	bc
uan_post	168.0	c
control	153.2	d
SED	4.4	NA
Number of Reps	4.0	NA

**8.10.1.3.4 Assign Formal Column Headers** While it is great to use simple column names when we are processing our data, when we go to present a table we should use more formal column headings. We use the function `cols_label()`. We specify the current column name and desired name in our argument. For example, the current name of the left column is `trt`, but we want to rename it “Treatment”.

The second argument is a little fancier. We want to rename our column so that it includes “Yield” and the measurement units, “bu/acre”. But we want to use the more scientifically accepted practice of foregoing the forward-slash “/” and instead write “bu acre-1” where the “-1” is in superscript.

To get a superscript “-1”, we need to use a couple of html arguments: “” specifies the following text is superscript; ends the superscript text.

So our text argument is “Yield (bu acre-1)”. To have R render it in html, however, we need to place our text string inside the `html()` function.

```
gt_tbl %>%
  fmt_number(columns = vars(yield), decimals = 1) %>%
  cols_width(everything() ~ "200px") %>%
  cols_align(align = "left", columns = vars(trt)) %>%
  cols_align(align = "center", columns = vars(yield, groups)) %>%
  cols_label(trt = "Treatment",
             yield = html("Yield (bu acre<sup>-1</sup>)"),
             groups = "LSD Groupings")

## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead

## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead

## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead
```

```
## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead
```

Treatment	Yield (bu acre <sup>-1</sup> )	LSD Groupings
aa_pre_post	191.3	a
aa_pre	181.8	b
uan_pre_post	179.4	b
uan_pre	175.9	bc
aa_post	174.8	bc
uan_post	168.0	c
control	153.2	d
SED	4.4	NA
Number of Reps	4.0	NA

**8.10.1.3.5 Table Aesthetics** At this point, our table is essentially complete. But we would like tweak a couple of things. The first thing we want to do is remove those “NA”s and just leave those cells blank. The *fmt\_missing()* function will take care of that for us. The first argument, “columns = TRUE”, tells R to search the entire table for occurrences of “NA”. The second argument, *missing\_text* = “”, specifies they be replaced with a blank (“”).

```
gt_tbl %>%
  fmt_number(columns = vars(yield), decimals = 1) %>%
  cols_width(everything() ~ "200px") %>%
  cols_align(align = "left", columns = vars(trt)) %>%
  cols_align(align = "center", columns = vars(yield, groups)) %>%
  cols_label(trt = "Treatment",
             yield = html("Yield (bu acre-1)"),
             groups = "LSD Groupings") %>%
  fmt_missing(columns = TRUE, missing_text = "")
```

```
## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead
```

```
## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead
```

```
## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead
```

```
## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead
```

```
## Warning: `columns = TRUE` has been deprecated in gt 0.3.0:
## * please use `columns = everything()` instead
```

Treatment	Yield (bu acre <sup>-1</sup> )	LSD Groupings
aa_pre_post	191.3	a
aa_pre	181.8	b
uan_pre_post	179.4	b
uan_pre	175.9	bc
aa_post	174.8	bc
uan_post	168.0	c
control	153.2	d
SED	4.4	
Number of Reps	4.0	

Finally, we would like to make a couple of the lines a little wider: the top and bottom lines of the table, as well as the line under the column labels. We can do this with the `tab_options()` function, which then allows us to format dozens of table aspects: lines, fonts, background colors, etc. We will use a table of “trio.border” arguments to widen those three lines.

```
gt_tbl %>%
  fmt_number(columns = vars(yield), decimals = 1) %>%
  cols_width(everything() ~ "200px") %>%
  cols_align(align = "left", columns = vars(trt)) %>%
  cols_align(align = "center", columns = vars(yield, groups)) %>%
  cols_label(trt = "Treatment",
             yield = html("Yield (bu acre-1"),
             groups = "LSD Groupings") %>%
  fmt_missing(columns = TRUE, missing_text = "") %>%
  tab_options(table.border.top.width = 4,
              table.border.bottom.width = 4,
              column_labels.border.bottom.width = 4)

## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead

## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead

## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead

## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead
```

```
## Warning: `columns = TRUE` has been deprecated in gt 0.3.0:
## * please use `columns = everything()` instead
```

Treatment	Yield (bu acre <sup>-1</sup> )	LSD Groupings
aa_pre_post	191.3	a
aa_pre	181.8	b
uan_pre_post	179.4	b
uan_pre	175.9	bc
aa_post	174.8	bc
uan_post	168.0	c
control	153.2	d
SED	4.4	
Number of Reps	4.0	

**8.10.1.3.6 Saving Your Table** Finally, we can assign our table to an object, “corn\_n\_gt”, and save it as an .html file to use it in other documents.

```
corn_n_gt = gt_tbl %>%
  fmt_number(columns = vars(yield), decimals = 1) %>%
  cols_width(everything() ~ "200px") %>%
  cols_align(align = "left", columns = vars(trt)) %>%
  cols_align(align = "center", columns = vars(yield, groups)) %>%
  cols_label(trt = "Treatment",
             yield = html("Yield (bu acre-1)"),
             groups = "LSD Groupings") %>%
  fmt_missing(columns = TRUE, missing_text = "") %>%
  tab_options(table.border.top.width = 4,
              table.border.bottom.width = 4,
              column_labels.border.bottom.width = 4)

## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead

## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead

## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead

## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead

## Warning: `columns = TRUE` has been deprecated in gt 0.3.0:
## * please use `columns = everything()` instead
```

```
gtsave(corn_n_gt, "corn_n_means_table.html")
```

We could go on and on, but that is enough for now, and should get you pretty far. More information on the `gt()` package can be found at: <https://gt.rstudio.com/index.html>.

### 8.10.2 Practice: Canola

```
canola = read.csv("data-unit-8/exercise_data/canola_gd.csv")
head(canola)
```

```
##   block cultivar    yield
## 1      1     Bob 1508.424
## 2      2     Bob 1962.072
## 3      3     Bob 1364.861
## 4      4     Bob 1832.715
## 5      1   Donna 1212.190
## 6      2   Donna 1629.680
```

Since the focus of this exercise is table-building, I will generate the initial lsd means table for you.

```
library(agricolae)

canola_model = aov(yield ~ block + cultivar, data = canola)

lsd_canola = LSD.test(canola_model, "cultivar")
lsd_canola

## $statistics
##   MSerror Df      Mean       CV   t.value      LSD
##   109086 17 2296.983 14.37894 2.109816 492.7357
##
## $parameters
##           test p.adjusted   name.t ntr alpha
## Fisher-LSD      none cultivar   6  0.05
##
## $means
##        yield      std r      LCL      UCL      Min      Max      Q25
## Bill    2296.268 326.6421 4 1947.851 2644.685 1865.903 2614.081 2142.888
## Bob     1667.018 277.4666 4 1318.601 2015.435 1364.861 1962.072 1472.533
## Donna   1833.282 514.0721 4 1484.865 2181.699 1212.190 2358.635 1525.308
```

```

## Jerry  2669.161 176.6376 4 2320.745 3017.578 2545.158 2923.794 2552.218
## Mickey 2406.509 263.5791 4 2058.092 2754.926 2130.413 2656.553 2207.614
## Phil   2909.658 275.1842 4 2561.242 3258.075 2565.683 3218.583 2776.782
##          Q50      Q75
## Bill   2352.545 2505.925
## Bob    1670.570 1865.055
## Donna  1881.152 2189.126
## Jerry  2603.847 2720.790
## Mickey 2419.535 2618.430
## Phil   2927.183 3060.060
##
## $comparison
## NULL
##
## $groups
##           yield groups
## Phil   2909.658     a
## Jerry  2669.161     ab
## Mickey 2406.509     b
## Bill   2296.268     bc
## Donna  1833.282     cd
## Bob    1667.018     d
##
## attr(,"class")
## [1] "group"

```

First, extract the \$groups table from the canola\_lsd output.

```
canola_means = lsd_canola$groups
```

Next, we need to convert our row names to a column

```

canola_table_w_trt_column = canola_means %>%
  rownames_to_column(var="cultivar")

canola_table_w_trt_column

```

```

##   cultivar     yield groups
## 1   Phil 2909.658     a
## 2   Jerry 2669.161     ab
## 3   Mickey 2406.509     b
## 4   Bill 2296.268     bc
## 5   Donna 1833.282     cd
## 6   Bob 1667.018     d

```

Finally, we need to calculate the SED and number of reps and add those to the table

```
r = 4
MSE = lsd_canola$statistics$MSerror

SED = sqrt(2*MSE/r)

canola_table_w_stats = canola_table_w_trt_column %>%
  add_row(cultivar="SED", yield=SED, groups="") %>%
  add_row(cultivar="Number of Reps", yield=r, groups="")

canola_table_w_stats
```

	cultivar	yield	groups
## 1	Phil	2909.6584	a
## 2	Jerry	2669.1615	ab
## 3	Mickey	2406.5090	b
## 4	Bill	2296.2682	bc
## 5	Donna	1833.2820	cd
## 6	Bob	1667.0181	d
## 7	SED	233.5444	
## 8	Number of Reps	4.0000	

At this point, the easiest way for you to make your table is to tweak the code below (this is why coding is so powerful).

You will need to:

- change the name of the data.frame from which you are building the table (line 1)
- change the number of decimals in the fmt\_number table to zero (line 2)
- substitute “cultivar” for treatment wherever it occurs in the code (lines 3 and 5)
- change the formal heading for the first column from “Treatment” to “Cultivar”

```
canola_means_table = gt(canola_table_w_trt_column) %>%
  fmt_number(columns = vars(yield), decimals = 1) %>%
  cols_width(everything() ~ "200px") %>%
  cols_align(align = "left", columns = vars(cultivar)) %>%
  cols_align(align = "center", columns = vars(yield, groups)) %>%
  cols_label(cultivar = "Cultivar",
             yield = html("Yield (bu acre-1"),
             groups = "LSD Groupings")
```

```
## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead

## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead

## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead

## Warning: `columns = vars(...)` has been deprecated in gt 0.3.0:
## * please use `columns = c(...)` instead
```

canola\_means\_table

Cultivar	Yield (bu acre <sup>-1</sup> )	LSD Groupings
Phil	2,909.7	a
Jerry	2,669.2	ab
Mickey	2,406.5	b
Bill	2,296.3	bc
Donna	1,833.3	cd
Bob	1,667.0	d

### 8.10.3 Practice: Broccoli LSD

Common bean was grown with organic practices following four previous crops.

```
bean = read.csv("data-unit-8/exercise_data/common.Bean.csv")
head(bean)
```

```
##   block      prev_crop fresh_weight
## 1     1    conv_broccoli    52.15924
## 2     1  green_manure     49.82191
## 3     1 organic_broccoli   36.80926
## 4     1        fallow     54.05342
## 5     2    conv_broccoli    52.79726
## 6     2        fallow     47.68364
```

#### 8.10.3.1 Create the LSD Groups table

```
library(agricolae)
# model statement
# LSD.test
# extract groups table from lsd output
# use rownames_to_column to create new column with rownames
```

#### 8.10.3.2 Prepare the table contents

```
# r = number of reps
# extract MSE from lsd output
# calculate SED
# use add_rows to add SED and r to lsd groups table
```

#### 8.10.3.3 Create the gt() table

```
# recycle table from above:
# change data.frame you feed to gt (line 1)
# adjust decimals as appropriate (line 2)
# change variable references for treatment and, if appropriate, yield (lines 3-5)
# change column labels (lines 5-7)
#
```



## Chapter 9

# Messy and Missing Data

You have to love the nice, complete datasets I have served up to you in this and other statistics texts. Indeed, some trials will work out this way – in general, the simpler the treatment (and the technology used to apply it) the greater your chances of a complete dataset. Planters will jam, nozzles will plug, and if a trial has 12 treatments a couple of them may end up in the wrong plot. Best to avoid those if possible.

As well, the smaller the experiment, or the more controlled the environment, the greater your odds of a complete dataset. For that reason, decades of university and industry research has been performed in 10- x 40- plots stacked closely together on manicured, table-top-flat ground. If you were a seed-breeder, you had dibs on these fields. If you were an ecologist like me, you might have to head to the back 40 (j/k).

If you can farm in your head or your basement, drop me a note and I will exempt you, with envy from this unit. For those of us who farm by the acre or section, however, the issue of data quality is an agonizing and sometimes subjective topic – but critical. Remember, most of our statistics are models, and the saying goes: “junk in, junk out.” Your models are only as good as the data you use to build them.

Part of your job as a researcher or end-user of any data, before you conduct or read the results from any test, is to ask yourself – are the data reliable enough to support the inferences? A trial with wildly-different experimental units may bias results – if all the replicates of one treatment end up in the better units and others are concentrated in the poorer ones. You may falsely recommend a product if you don’t catch this.

At the other extreme, the failure conduct research on similar experimental units will introduce background variance (or noise) that prevents a statistical test from concluding a difference is not the result of chance, even though the treatments

are, in fact, different. In that case, you may fail to introduce – or adopt – a product or technology with real promise.

In this unit, we will first learn ways to inspect datasets for extreme values which, even given the inherent variability of data, may be suspect. Boxplots, histograms, and probability plots will be our tools for these exercises.

We will then address the uncomfortable question of what to do when we have missing data, either because a plot was compromised during the trial, or because we rejected that plot because its extreme value.

## 9.1 Inspecting data for Normal Distributions

I know, I know, it is so exciting to have data! The hard physical work of the research is done, the data is painstakingly entered into Excel. Let's run the ANOVAs and regressions now – if we hurry we can still make it to happy hour!

It is so tempting to jump right into deep analyses as the data roll in. But it is important to realize these analyses are based on assumptions:

- That the observations for each treatment level are normally distributed around the treatment mean
- That the variance or spread of observations around each level of treatment is roughly equal.
- That experimental units are comparable among treatment levels, so that the treatment and error effects are appropriately separated.

It is very possible, based on the nature of trials, that one or more of these assumptions may be violated. If you have ever counted weeds in a herbicide trial, you have noted that well-treated plots have weed counts that are consistently near zero – but that weedy checks have wildly variable counts (100, 200, 300 weeds). Growth analysis (where plants of different sizes are measured) is also prone to messy data issues, because variance in measurements increases numerically as plants grow. Experimental units (plots) in both commercial and research farm fields can vary because of prior management unknown to the researcher.

### 9.1.1 Histograms

In the very first unit of this course Unit 2, you were introduced to the histogram. Recall the histogram is a vertical bar chart in which the width of bars defines the different ranges into which observations are grouped, and the height represents the count or proportion of observations falling into each range.

In the data below, we have a dataset with 500 observations of corn yield. The mean is approximately 180. We can see the data distribution is approximately normal.

```
library(tidyverse)

## -- Attaching packages ----- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr    1.0.8
## v tidyrr   1.2.0     v stringr  1.4.0
## v readr    2.1.2     vforcats 0.5.1

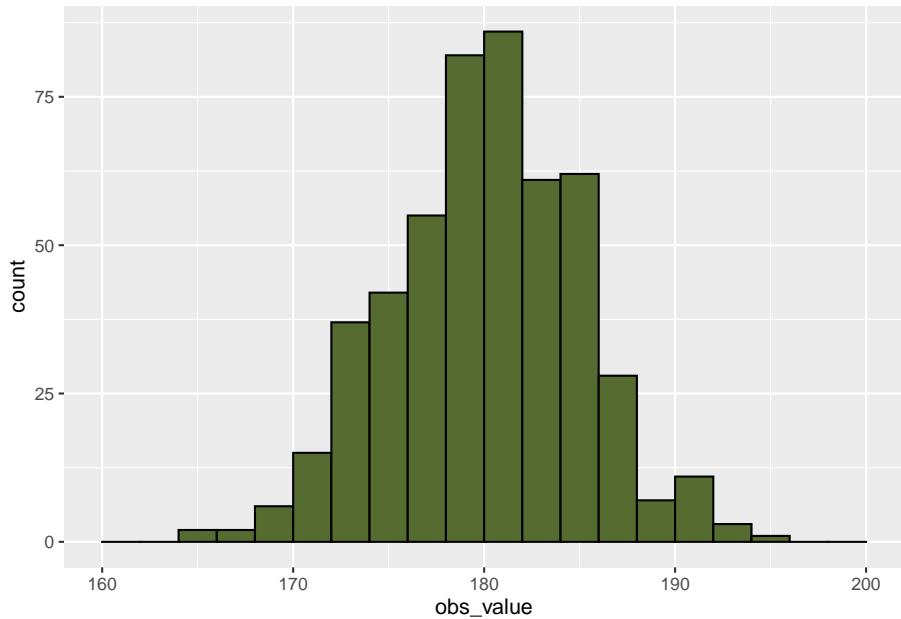
## Warning: package 'ggplot2' was built under R version 4.1.3

## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()   masks stats::lag()

# library(fGarch)
#
set.seed(082720)
norm_data = data.frame(obs_value = rnorm(500, mean=180, sd=5)) %>%
  mutate(dataset="normal")
# set.seed(5)
# skewed_data = data.frame(obs_value = rsnorm(1000, mean = 20, sd = 10, xi = 100)) %>%
#   mutate(dataset="skewed")
#
# norm_and_skewed = rbind(norm_data, skewed_data) %>%
#   as.data.frame() %>%
#   dplyr::filter(obs_value>0)
#
#
# write.csv(norm_and_skewed, "data/norm_and_skewed.csv", row.names = FALSE)

# norm_and_skewed = read.csv("data/norm_and_skewed.csv")

norm_data %>%
  # filter(dataset=="normal") %>%
  ggplot(aes(x=obs_value)) +
  geom_histogram(breaks = seq(160,200,2), fill="darkolivegreen", color="black")
```



The summary data are shown below. We can see that the median and mean are both approximately 180 bushels per acre. We can also see the 1st and 3rd quantiles (equal to the 25th and 75th percentiles) are a little over three bushels from the median. The minimum and maximum observations are also similarly spaced from the median.

```
summary(norm_data$obs_value)
```

```
##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
##    164.9   176.8  180.2   180.0  183.4   194.2
```

When we are dealing with data such as pest counts, our data may be non-normal. Rather than being symmetrical, the data may be skewed to one side or another. For example, in the dataset below, total velvetleaf dry weight in grams per square meter was measured. If you have worked like me with weed populations, you realize weed competitiveness is all about outracing the crop to the sun. If the weed loses, which it will in most cases, it will be small. But the proud few weeds who beat the crop will be huge. That is reflected in the data below.

```
library(fGarch)
```

```
## Loading required package: timeDate
```

```

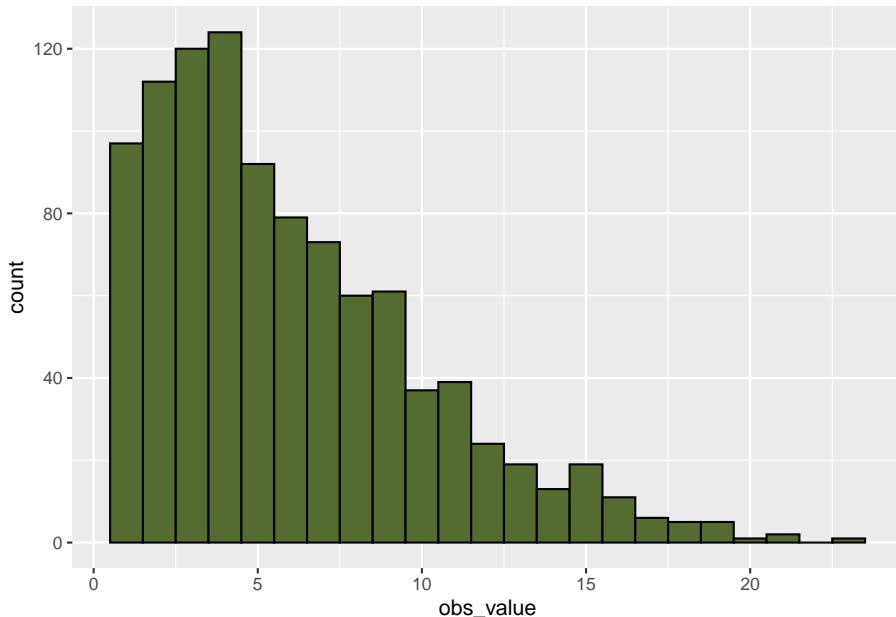
## Loading required package: timeSeries

## Loading required package: fBasics

set.seed(5)
velvetleaf_skewed = data.frame(obs_value = rsnorm(1000, mean = 6, sd = 4, xi = 50))

velvetleaf_skewed %>%
  ggplot(aes(x=obs_value)) +
  geom_histogram(fill="darkolivegreen", color="black", binwidth = 1)

```



When we look at the histogram, the data are skewed to the right. The histogram is not symmetrical.

```
summary(velvetleaf_skewed$obs_value)
```

```

##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
##  0.7387  2.8225  5.0536  6.0104  8.3332 23.2785

```

When we look at the summary data, we first notice the mean and median are different. For a dataset this size (1000 observations, we would expect them to be more similar.) We notice that the first quantile is closer to the median than the third quantile. The greatest indication the data is skewed, however, is in

that the minimum is about 4 plants less than the median, while the maximum is about 18 plants greater.

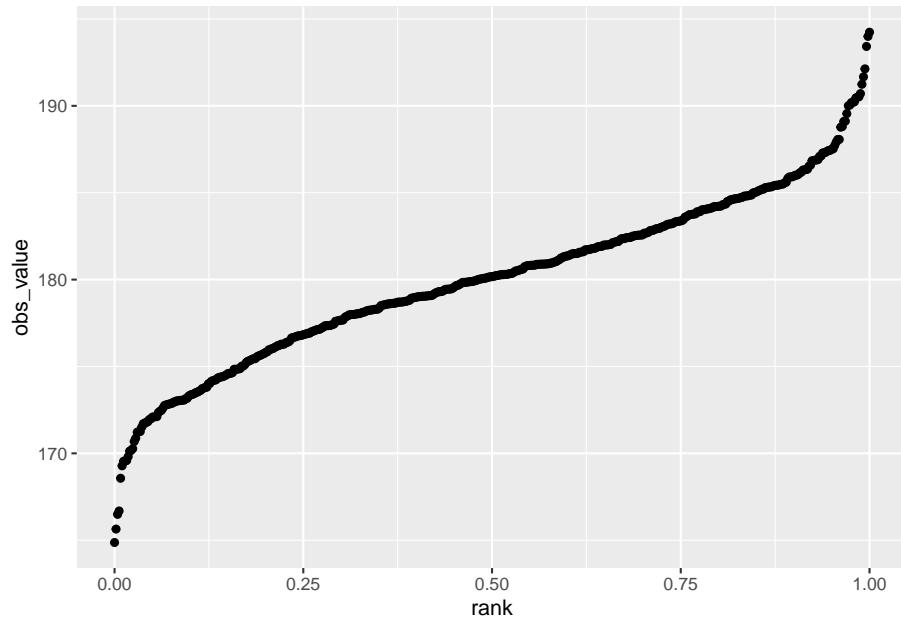
Data like this may be transformed (mathematically re-scaled) so that it is more normal for analyses. We will cover this below.

### 9.1.2 Rank Percentile Plots

Another way to inspect the normality of datasets is to use a rank percentile plot. This plot uses the percentile rank of each observation, from lowest to highest, as the x-value of each point. The y-value of the point is its observed value.

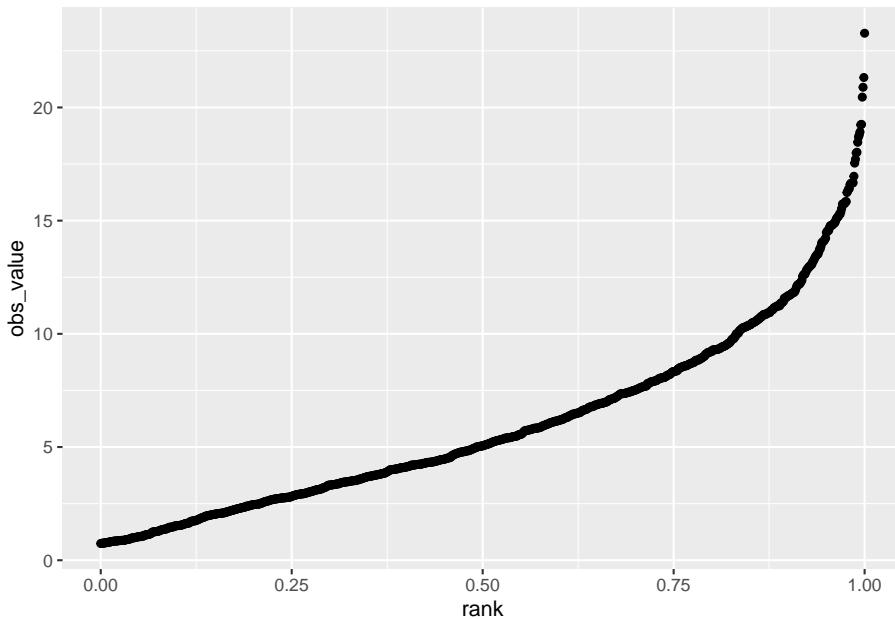
The data for our normally-distributed corn yield dataset are plotted in the rank percentile plot below. Normally-distributed data tend to be strongly linear in the middle of the plot. If we draw a regression line through the plot, you can see most of the data are close to that line. The lowest percentile points fall below the line. That means they are a little lower in value than the normal distribution function might predict. The opposite is true of the highest percentile points. This indicates our distribution is a little bit wider than normal, but not enough that we cannot use it for analysis.

```
norm_data %>%
  mutate(rank = percent_rank(obs_value)) %>%
  ggplot(aes(x=rank, y=obs_value)) +
  geom_point()
```



Our skewed data, however, shows up quite differently in the rank percentile plot. We can see that most of the data closely fit a line. But starting around the 75th percentile, the observed values are much greater than the predicted values – almost twice as much. This means the distribution is much wider to the right of the distribution curve than to the left, and that the data are non-normal

```
velvetleaf_skewed %>%
  mutate(rank = percent_rank(obs_value)) %>%
  ggplot(aes(x=rank, y=obs_value)) +
  geom_point()
```



### 9.1.3 Box Plots

The first two methods I have shown you, the histogram and rank percentile plots, are useful if you have a few treatments with a large number of replicates. They are taught in every statistics course and you should know about them. But, in my experience, they are not useful if you have a trial with fewer replications. A normal distribution is not going to appear in a histogram if you only have four replicates – instead you will just see the four individual measurements.

Box plots, on the other hand, are very useful for inspecting multiple treatments. In the plot below, boxplots for four corn treatments are shown. The treatments are labeled A, B, C, and D. The data are plotted so their treatments are listed along the vertical axis, and their values are listed along the y-axis.

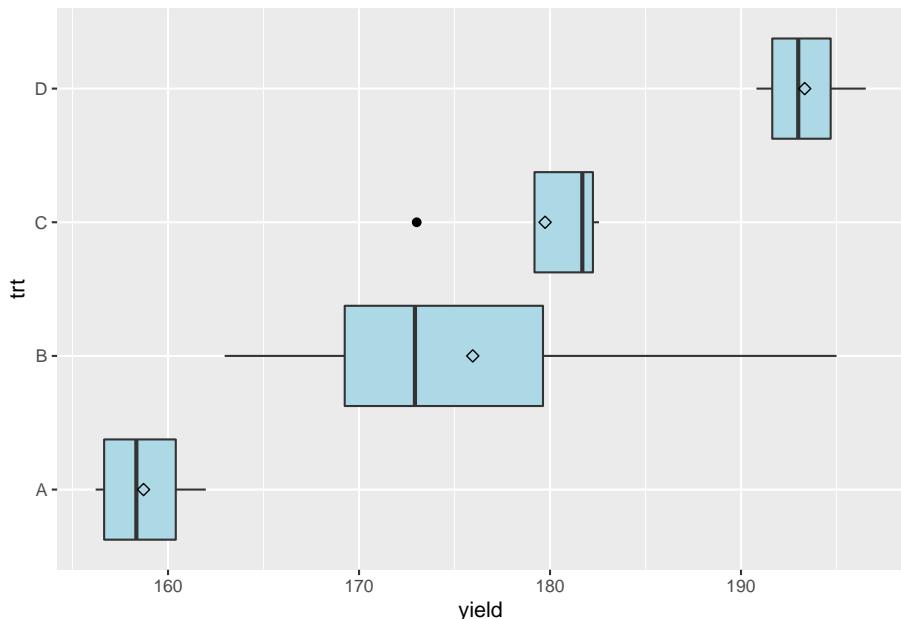
```

norm_data_mult_trts = data.frame(trt = rep(c("A", "B", "C", "D"), each=4),
                                  mean = rep(c(160,170,180,190), each=4))

set.seed(082720)
norm_data_mult_trts = norm_data_mult_trts %>%
  mutate(error = rnorm(16, 0, 5)) %>%
  mutate(yield = mean + error) %>%
  # create outlier
  mutate(yield = if_else(row_number()==8, 195, yield))

norm_data_mult_trts %>%
  ggplot(aes(x=trt, y=yield)) +
  geom_boxplot(outlier.colour="black", outlier.shape=16,
               outlier.size=2, notch=FALSE, fill="lightblue") +
  coord_flip() +
  stat_summary(fun=mean, geom="point", shape=23, size=2)

```



The boxplots help us understand the distribution of the data. Lets start with the box, which tells about the spread of the data. The left side of the box is the 25th percentile, the line in the middle is the 50th percentile (or median), and the right side of the box is the 75th percentile. So the box shows us the spread of the middle half of the observations for each treatment.

The diamond shown within the box is the mean. In a normal distribution, the median and mean should be close.

The lines extending from the left and right side of the box are called whiskers. The whiskers extend to the lowest and highest observations for a treatment. The whiskers extend no more than 1.5 times the *inter-quartile range*, which for the lower whisker is the difference between the 25th and 50th percentiles, and for the upper whisker is the difference between the 50th and 75th percentiles.

In treatment B, we can see the upper whisker is missing, and instead there is a point to the right of the bar. If an observation is beyond 1.5 times the interquartile range, the whisker is not shown and the observation is instead represented by a point. This observation is called an *outlier*, meaning that it is outside the range of values expected in a normal distribution. We will talk more about outliers in the next section.

The boxplot tells us something beyond the distribution of the individual treatments. If the boxes are markedly different in their width, the data may have substantially different variances. We should investigate these further using a *mean-variance plot*, and perhaps a statistical *test of heterogeneity*.

## 9.2 Inspecting Data for Equal Variances

So far, we have learned to use the t-test and analysis of variance to test named treatments (that is, hybrids, management practices, and other products that can be described by name). These tests generally assume not only that observed values are normally distributed, but that the variances are approximately equal among the different treatments in our experiment. If the variances are unequal, we may calculate least significant differences (LSDs) or honest significant differences (HSDs) that are inappropriate. Among treatments that have smaller variances, our LSD or HSD may be overestimated; among treatments that have larger variances, the LSD or HSD may be underestimated.

The visual inspection of individual treatment distributions in the box plot above, followed by a scatter plot of the treatment variances versus their means, can give us a visual sense of unequal variances. These suspicions can then be tested using a Test for Homogeneity that calculates the probability of differences in variances as greater as those observed.

### 9.2.1 Mean-Variance Plot

In a mean-variance plot, the treatment means are plotted along the horizontal axis and the variances are plotted along the vertical axis. The plot for the corn yield dataset we have used so far is shown below.

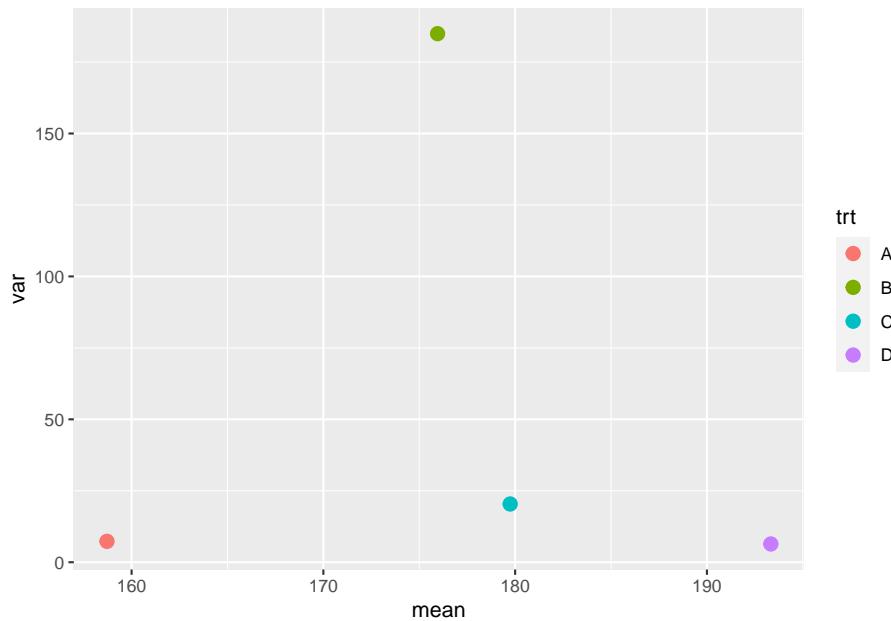
```
norm_stats = norm_data_mult_trts %>%
  group_by(trt) %>%
  summarise(mean=mean(yield),
```

```

var=var(yield)) %>%
ungroup()

ggplot(data=norm_stats, aes(x=mean, y=var)) +
  geom_point(aes(color=trt), size=3)

```



We can see the variance of treatment B is many times greater than that of the other treatments. In general, we like to see the variances differ by no more than a factor of 2.

In cases where we are dealing with populations that either “thrive or die” based on environment – particularly pest populations – we may see relationships between the mean and variance. Pest count data is often like this. In our velvetleaf counts, for example, we might find that our greater treatment means are also associated with greater variations in counts between plots.

```

rnorm2 <- function(n,mean,sd) { mean+sd*scale(rnorm(n)) }
r <- rnorm2(100,4,1)
mean(r) ## 4

## [1] 4

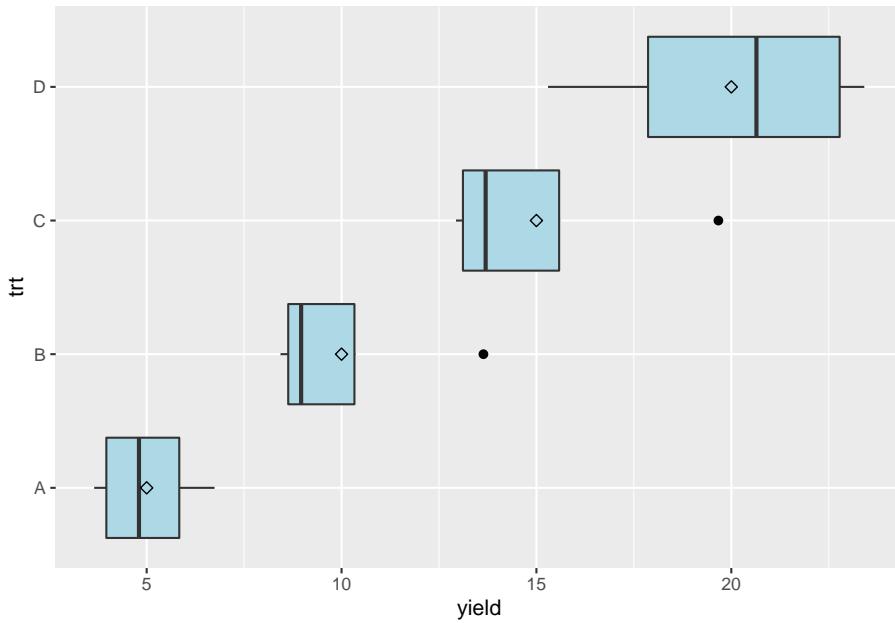
```

```
sd(r)
```

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

```
set.seed(3)
norm_data_prop_var = data.frame(A = rnorm2(4, 5, sqrt(2)),
                                 B = rnorm2(4, 10, sqrt(6)),
                                 C = rnorm2(4, 15, sqrt(10)),
                                 D = rnorm2(4, 20, sqrt(14))) %>%
gather(trt, yield)

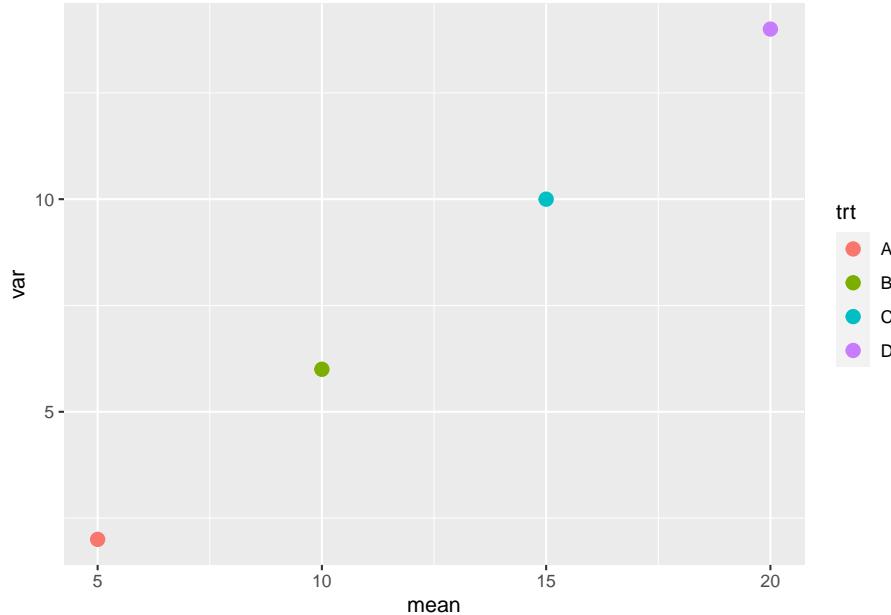
norm_data_prop_var %>%
  ggplot(aes(x=trt, y=yield)) +
  geom_boxplot(outlier.colour="black", outlier.shape=16,
               outlier.size=2, notch=FALSE, fill="lightblue") +
  coord_flip() +
  stat_summary(fun=mean, geom="point", shape=23, size=2)
```



In this case, the mean-variance plot may show a linear relationship between variance and mean.

```
norm_data_prop_var_stats = norm_data_prop_var %>%
  group_by(trt) %>%
  summarise(mean=mean(yield),
            var=var(yield)) %>%
  ungroup()

norm_data_prop_var_stats %>%
  ggplot(aes(x=mean, y=var)) +
  geom_point(aes(color=trt), size=3)
```



Finally, we may observe a dataset in which the distributions not only increase with means, but seem to do so exponentially.

```
rnorm2 <- function(n,mean,sd) { mean+sd*scale(rnorm(n)) }
r <- rnorm2(100,4,1)
mean(r) ## 4

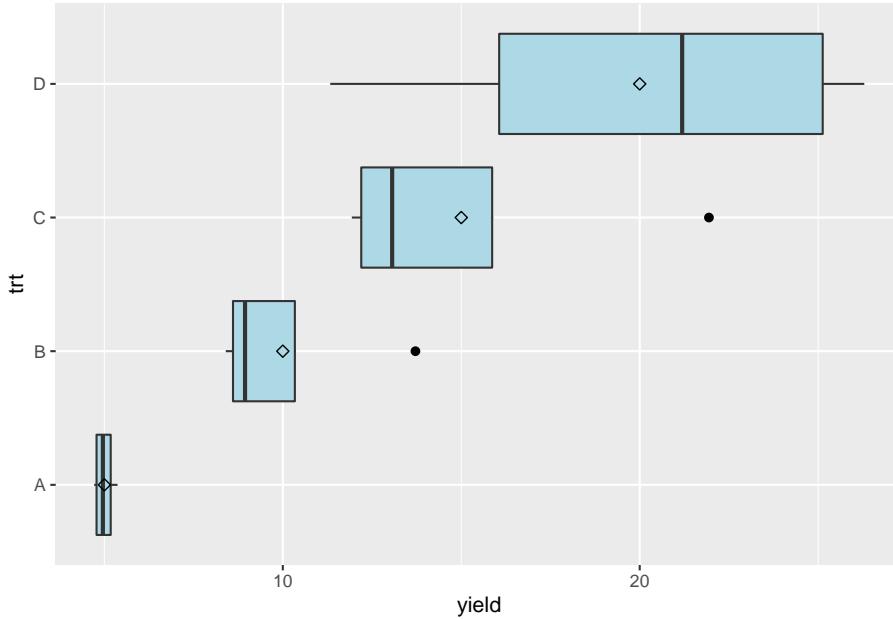
## [1] 4
```

```
sd(r)

## [1] 1

set.seed(3)
norm_data_prop_sd = data.frame(A = rnorm2(4, 5, 0.3),
                                B = rnorm2(4, 10, 2.5),
                                C = rnorm2(4, 15, 4.7),
                                D = rnorm2(4, 20, 6.9)) %>%
  gather(trt, yield)

norm_data_prop_sd %>%
  ggplot(aes(x=trt, y=yield)) +
  geom_boxplot(outlier.colour="black", outlier.shape=16,
               outlier.size=2, notch=FALSE, fill="lightblue") +
  coord_flip() +
  stat_summary(fun=mean, geom="point", shape=23, size=2)
```



In this case, the mean-variance plot may show a curved relationship between variance and mean.

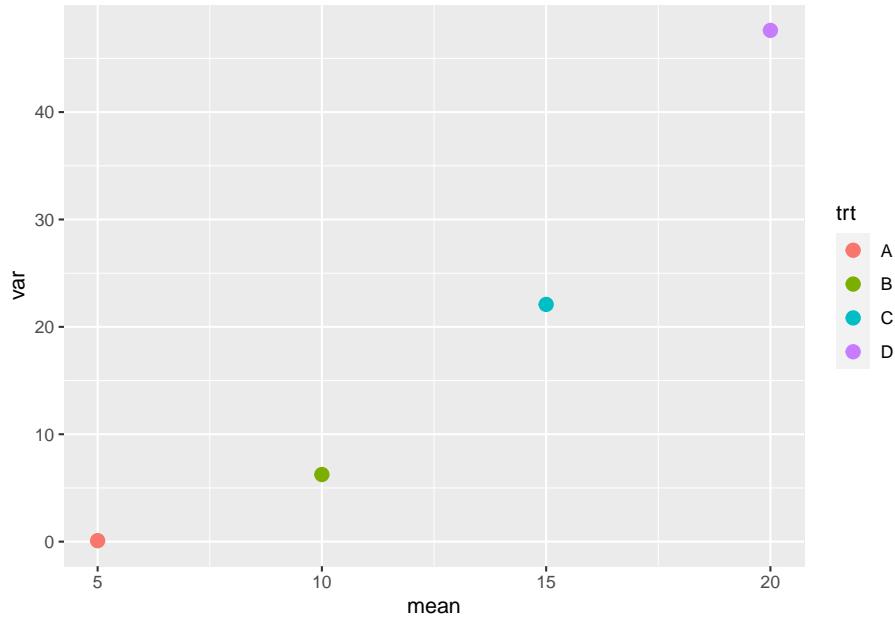
```
s_norm_stats = norm_data_prop_sd %>%
  group_by(trt) %>%
  summarise(mean=mean(yield),
```

```

    var=var(yield),
    sd=sd(yield)) %>%
ungroup()

ggplot(data=s_norm_stats, aes(x=mean, y=var)) +
  geom_point(aes(color=trt), size=3)

```

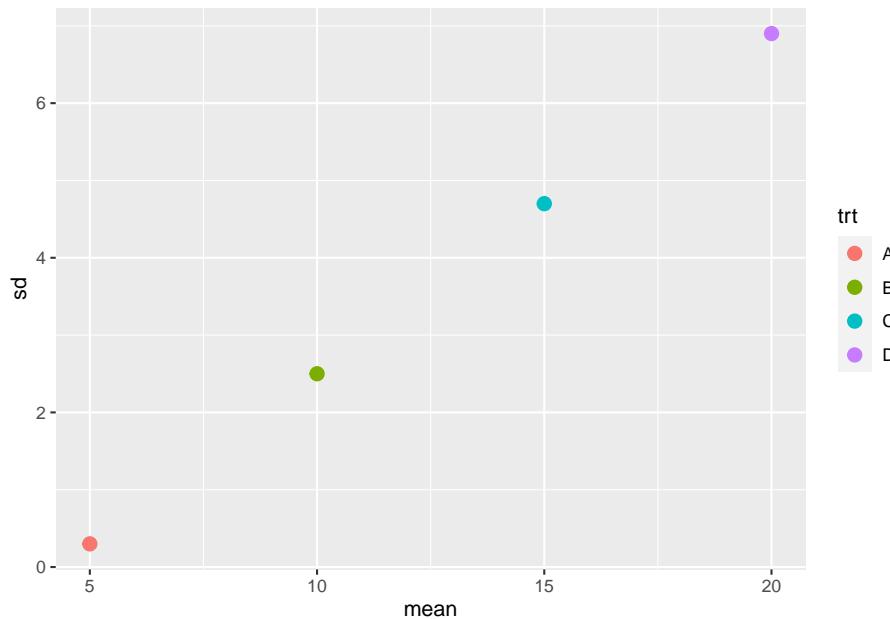


We may want to check whether the standard deviation has a more linear relationship to mean.

```

ggplot(data=s_norm_stats, aes(x=mean, y=sd)) +
  geom_point(aes(color=trt), size=3)

```



The largest mean has a significant difference of 6.9, while the smallest mean has a significant difference of 0.3. In other words, the largest significant difference is 23 times the smallest significant difference.

### 9.2.2 Homogeneity of Variance Tests

In previous units, we learned how to compare two variances – using the F-test. In the Analysis of Variance, we tested whether the variance among treatment means was greater than the variance within treatments. If the treatment variance was sufficiently greater than the error variance, we concluded the treatment effect explained a significant amount of the variation in observed values.

In this unit, we want to do something similar – we want to compare the variances associated with multiple treatments to see if they are significantly different. When data are normally-distributed, the method for comparing multiple variances is *Bartlett's Test*. (If you continue your statistical travels, you may come across Levene's Test, but that is for non-normal data.)

Bartlett's test, as best as I can tell (the formula is awful and no one seems willing to explain it), acts like a sum of squares for variances, comparing the variances of the individual treatments to their mean when pooled together. This is an incomplete explanation, but I hope it will satisfy the curious. Fortunately for us, the test is easy to implement in R and the output simple.

If we run Bartlett's test on our corn data above, we get the following results.

```
bartlett.test(yield~trt, norm_data_mult_trts)

##
##  Bartlett test of homogeneity of variances
##
## data: yield by trt
## Bartlett's K-squared = 10.355, df = 3, p-value = 0.01578
```

Let's go through the output. "Bartlett's K-squared" is the statistic produced by the nasty formula I referenced above. Don't worry about that. The degrees of freedom refers to the four treatments whose variances we are comparing. Most important, of course, is our p-value. There are many opinions on when to transform data – but I would recommend against transforming data unless the p-value is less than 0.01. I would also recommend running your data on both the transformed and untransformed data and comparing results. If transformation does not change your inferences, then skip it.

Here is the Bartlett's test on our velevleaf data where the mean and standard deviation were linearly related:

```
bartlett.test(yield~trt, norm_data_prop_sd)

##
##  Bartlett test of homogeneity of variances
##
## data: yield by trt
## Bartlett's K-squared = 14.217, df = 3, p-value = 0.002625
```

In this case, we will want to analyze both the transformed and untransformed data before deciding which to us for our final inferences.

### 9.3 Dealing with Messy Data

Dealing with messy data is one of the more uncomfortable aspects of statistics, but also one of the most important. Our tests and summary statistics are based on assumptions. For tests, we assume the data are from a populations that have approximately normal distributions. We also assume they have variances that are equal – otherwise our mean separation tests will fail.

And finally, and this is a concept that I learned just while writing this: the validity of our inferences is based on the assumption that our samples represent the population that we are studying. Which brings us back to outliers.

### 9.3.1 Outliers

Outliers can have a powerful effect in skewing data. Particularly in smaller datasets (i.e. fewer than 10 replicates per treatment), an outlier can have noticeable effects on a distribution's normality and its variance. In regression analyses, one outlier can significantly change the slope of the model.

Does this mean that outliers should be omitted from the dataset? Not necessarily – first we should inspect the data more closely. The outlier might be a mistake in recording a measurement. It could be an inconsistent scale in a combine. These would be experimental errors that mean the outlier is an *artifact* of our methods, rather than a representative sample from our population. In that case, we may want to remove that observation from our dataset.

But investigating the outlier may also include examining the location where it was taken. This can be difficult if you are not the primary researcher and on-site where the data were gathered. But if you can overlay your plot map with a soils map, or work with aerial imagery, or examine as-applied maps, you may be able to identify differences in environment or management that caused a dramatic difference in the observed value.

In such a case, you may decide that plot did not represent the environment about which you were trying to draw inferences, and choose to omit it from the dataset. At the same time, however, knowing that the outlier's environment or management had a dramatic effect on its performance, you may generate new hypotheses about that product. In fact, you may learn more from your outlier, through the new research it inspires, than you do from the original experiment.

Also, before removing an outlier, it is a good idea to run your tests with and without it to see whether it changes your conclusions. When you run your model, look at the standardized residuals. How many standard errors is the outlier from the predicted value? As a rule, if an observed value is more than two standard deviations from the predicted value, I scrutinize it before allowing it into the final analysis.

If you are comparing different treatments, does it change the significance of tests or differences among treatments? If you are generating a regression model, does the change in slope have a dramatic effect on the values you will predict? Will the change in slope have a dramatic effect on grower inputs and cost, or will the effect be more modest?

These are important questions, because as uncomfortable as it is to identify and/or remove outliers, working with incomplete datasets can be even nastier. If the statistical significance of tests or means separations are not affected by the outlier, it is best to leave it in the dataset if possible, especially if treatment replications are limited.

### 9.3.2 Non-normal Data and Unequal Variances

Above, we discussed two other problems with data: data that were skewed (or non-normal) and therefore could not be modelled based on the normal distribution, and data where treatment variances were not equal – they were, in statistical terminology, they suffered from *heterogeneity of variances*.

Both issues can arise out of trials where observation values vary widely: counts that include rare events or where one treatment (like a check plot) can “blow up” and have a huge value. Growth studies, where size increases occur at exponential rates, are another.

These two issues may be similarly addressed by transforming the data. When we transform data, we use a different measuring system to rescale it. The easiest example of rescaling data is pH. Recall pH is the concentration of hydrogen atoms in a solution. This concentration can range from 0 to  $10^{-14}$

So when is the last time you read that your soil pH was  $6.5 \times 10^{-6}$  ? You wouldn't. We commonly speak of pH as ranging from 1 (highly acidic) through 7 (neutral) to 14 (highly basic). You are used to using the logarithmic scale(10, 1, 0.10, 0.010), rather than the arithmetic scale (1,2,3,4,5). The decibel scale for sound and the Richter scale for earthquakes also use the logarithmic scale.

#### 9.3.2.1 Natural Logarithm

There are several ways to transform data, but the one I have most-often used is the natural logarithm. The natural logarithm transformation is often used when we are working with data that have a wide range of values. What constitutes a wide range of values? Think growth analysis, or counts of common events (for example, weed counts in a herbicide trial that includes treatments that vary widely in effectiveness). In these trials, it is not uncommon for observed values to vary by two or more orders of magnitude (powers of 10).

Our process for working with transformed data is as follows:

- Transform the original observations to their natural logs.
- Calculate the ANOVA
- Calculate treatment means using transformed data
- Back-transform the treatment means to the original measurement scale so they are more intuitive to users

Lets work with our velvetleaf data above. Below is the analysis of variance and means separation using the least significant difference test.

```

library(agricolae)

##
## Attaching package: 'agricolae'

## The following objects are masked from 'package:timeDate':
##
##      kurtosis, skewness

vleaf_model_before = aov(yield~trt, norm_data_prop_sd)
summary(vleaf_model_before)

##           Df Sum Sq Mean Sq F value    Pr(>F)
## trt          3 500.0 166.67   8.767 0.00237 ***
## Residuals    12 228.1   19.01
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

lsd_before = LSD.test(vleaf_model_before, "trt")
lsd_before$groups

##   yield groups
## D     20     a
## C     15     ab
## B     10     bc
## A      5     c

```

The treatment effect is significant, and some of the treatments are significantly different from each other, but there are also noticeable overlaps.

Now let's transform the data using the natural log. The original and transformed data are show below.

```

vleaf_log = norm_data_prop_sd %>%
  mutate(log_yield = log(yield))
vleaf_log

##   trt     yield log_yield
## 1 A 4.802644 1.569167
## 2 A 5.113508 1.631886
## 3 A 5.369530 1.680740
## 4 A 4.714318 1.550604

```

```

## 5   B  9.211467  2.220449
## 6   B  8.401127  2.128366
## 7   B  8.671603  2.160054
## 8   B 13.715802  2.618549
## 9   C 11.933265  2.479330
## 10  C 21.939493  3.088288
## 11  C 13.841261  2.627654
## 12  C 12.285982  2.508459
## 13  D 11.328175  2.427293
## 14  D 26.292831  3.269296
## 15  D 24.739139  3.208387
## 16  D 17.639854  2.870161

```

Now when we run our Bartlett's test, we see the p-value is 0.09 – there is no longer a significant difference among the treatment variances.

```

bartlett.test(log_yield~trt, vleaf_log)

##
##  Bartlett test of homogeneity of variances
##
## data: log_yield by trt
## Bartlett's K-squared = 6.4701, df = 3, p-value = 0.09085

```

The largest standard deviation is still 6.5 times the smallest standard deviation – but the difference has decreased dramatically. When we run our analysis of variance, we see that our p-value has decreased by several orders of magnitude.

```

vleaf_model_after = aov(log_yield~trt, vleaf_log)
summary(vleaf_model_after)

##
##          Df Sum Sq Mean Sq F value    Pr(>F)
## trt        3  4.043  1.3478   18.95 7.58e-05 ***
## Residuals 12  0.853  0.0711
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

When we run our LSD test, we notice more significant differences among the means, especially treatments A and B, which were associated with lower treatment means.

```

lsd_after = LSD.test(vleaf_model_after, "trt")
lsd_after$groups

```

```
##   log_yield groups
## D  2.943784      a
## C  2.675933      ab
## B  2.281854      b
## A  1.608099      c
```

Our last step is to back-transform the means from our LSD test.

```
back_transformed_means = lsd_after$groups %>%
  rownames_to_column(var="treatment") %>%
  mutate(yield = exp(log_yield))

back_transformed_means %>%
  dplyr::select(treatment, yield, groups)
```

```
##   treatment      yield groups
## 1          D 18.987563      a
## 2          C 14.525893      ab
## 3          B  9.794827      b
## 4          A  4.993311      c
```

In the above table, we have back-transformed the means in our LSD table to their original scale.

## 9.4 Dealing with Missing Data

Missing data can be very problematic. Whether missing data are the result of deleting outliers, or plots lost to weather or human damage, there are three options:

- Drop the treatment entirely from the dataset
- Drop one observation from each of the other treatments; if a Randomized Complete Block Design was used, delete an entire block
- Predict the value for the plot that was omitted or lost

As you see, these are ugly choices to make. If you drop the treatment entirely from the dataset, you lose all ability to test it against the other treatments. If you drop other replicates or the remainder of a block, you retain all treatments but reduce the degrees of freedom for statistical tests, rendering them less sensitive.

The third option, predicting the value that is missing, comes with its own challenges. The missing value for a plot is generally calculated using the linear additive model. For a completely randomized design, the linear model is:

$$Y_{ij} = \mu + T_i + \epsilon_{(i)j}$$

So the missing value would be equal to  $\mu + T_i$ , where  $i$  would be whatever treatment level the missing plot received.

In a randomized complete block design, the linear additive model is:

$$Y_{ij} = \mu + B_i + T_j + BT_{ij}$$

The missing value would be equal to  $\mu + B_i + T_j$ , where  $i$  is the block to which the missing plot occurred, and  $j$  is the treatment the treatment the missing plot received.

Note that we did not include  $\epsilon_{(i)j}$  or  $BT_{ij}$  in estimating the missing values. Although this approach is widely used, this is a shortcoming. When we predict a missing value from the effects of treatment or treatment within block, we are using *mean* effects. So the predicted value will be exactly the mean for a given treatment or treatment within block. Because the predicted value is closer to the treatment and block means than it would be otherwise, it will contribute less to the treatment variance than it would normally.

We can demonstrate this with a normally-distributed dataset.

```
set.seed(083020)
more_norm_data_1 = data.frame(yield = rnorm(4, mean=165, sd=5.2)) %>%
  mutate(trt="A")
set.seed(083021)
more_norm_data_2 = data.frame(yield = rnorm(4, mean=169, sd=5.2)) %>%
  mutate(trt="B")
set.seed(083022)
more_norm_data_3 = data.frame(yield = rnorm(4, mean=170, sd=5.2)) %>%
  mutate(trt="C")
set.seed(083023)
more_norm_data_4 = data.frame(yield = rnorm(4, mean=172, sd=5.2)) %>%
  mutate(trt="D")

more_norm_data = rbind(more_norm_data_1, more_norm_data_2, more_norm_data_3, more_norm_data_4)
  mutate(trt = as.factor(trt)) %>%
  mutate(random_no = rnorm(16,0,1)) %>%
  arrange(random_no) %>%
  mutate(plot=row_number()) %>%
  dplyr::select(plot, trt, yield) %>%
  mutate(yield = round(yield,1))

more_norm_data
```

```
##   plot trt yield
## 1    1 A 166.8
## 2    2 C 169.9
## 3    3 B 170.6
## 4    4 C 174.9
## 5    5 B 161.8
## 6    6 A 168.3
## 7    7 A 169.2
## 8    8 C 159.9
## 9    9 B 160.8
## 10  10 C 166.2
## 11  11 A 160.3
## 12  12 D 172.1
## 13  13 D 175.4
## 14  14 D 175.3
## 15  15 B 170.6
## 16  16 D 172.2
```

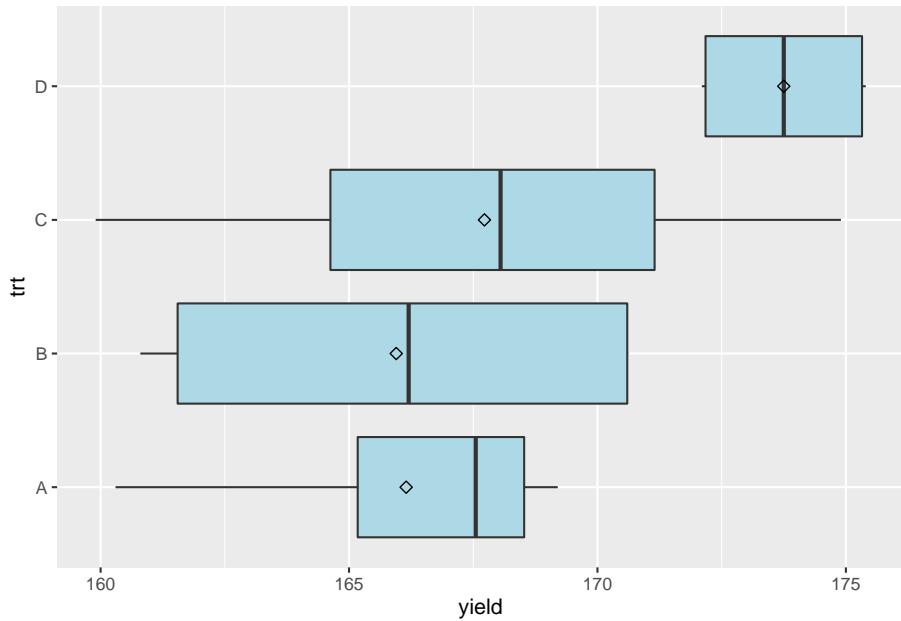
Here is the anova for the original data:

```
summary.aov(lm(yield~trt, data=more_norm_data))
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
## trt	3	160.6	53.52	2.418	0.117
## Residuals	12	265.6	22.13		

And here is the boxplot:

```
more_norm_data %>%
  ggplot(aes(x=trt, y=yield)) +
  geom_boxplot(outlier.colour="black", outlier.shape=16,
               outlier.size=2, notch=FALSE, fill="lightblue") +
  coord_flip() +
  stat_summary(fun=mean, geom="point", shape=23, size=2)
```



Here is the table with the treatment and error effects broken out. Unlike previous effects tables, I have also added the within treatment variance.

```
more_norm_data_effects = more_norm_data %>%
  mutate(mu = mean(yield)) %>%
  group_by(trt) %>%
  mutate(trt_mean = mean(yield),
        trt_var = var(yield)) %>%
  ungroup() %>%
  mutate(trt_effect = trt_mean - mu) %>%
  mutate(error_effect = yield - trt_effect - mu) %>%
  dplyr::select(plot, trt, yield, mu, trt_var, trt_effect, error_effect)

knitr::kable(more_norm_data_effects)
```

plot	trt	yield	mu	trt_var	trt_effect	error_effect
1	A	166.8	168.3938	16.190000	-2.24375	0.650
2	C	169.9	168.3938	39.922500	-0.66875	2.175
3	B	170.6	168.3938	28.996667	-2.44375	4.650
4	C	174.9	168.3938	39.922500	-0.66875	7.175
5	B	161.8	168.3938	28.996667	-2.44375	-4.150
6	A	168.3	168.3938	16.190000	-2.24375	2.150
7	A	169.2	168.3938	16.190000	-2.24375	3.050
8	C	159.9	168.3938	39.922500	-0.66875	-7.825
9	B	160.8	168.3938	28.996667	-2.44375	-5.150
10	C	166.2	168.3938	39.922500	-0.66875	-1.525
11	A	160.3	168.3938	16.190000	-2.24375	-5.850
12	D	172.1	168.3938	3.416667	5.35625	-1.650
13	D	175.4	168.3938	3.416667	5.35625	1.650
14	D	175.3	168.3938	3.416667	5.35625	1.550
15	B	170.6	168.3938	28.996667	-2.44375	4.650
16	D	172.2	168.3938	3.416667	5.35625	-1.550

Plot 8 has a greater error effect than most other plots. Let's treat it as an outlier, delete it, and recalculate the treatment means. Let's delete it and see how that changes the treatment effect for treatment C.

```
plot_8 = data.frame(plot=8 ,trt="C", yield=NA)

more_norm_data_interp = more_norm_data %>%
  dplyr::filter(!plot==8) %>%
  rbind(plot_8) %>%
  arrange(plot) %>%
  mutate(mu = mean(yield, na.rm = TRUE)) %>%
  group_by(trt) %>%
  mutate(trt_mean = mean(yield, na.rm = TRUE),
         trt_var = var(yield, na.rm = TRUE)) %>%
  ungroup() %>%
  mutate(trt_effect = trt_mean - mu) %>%
  mutate(error_effect = yield - trt_effect - mu) %>%
  dplyr::select(plot, trt, yield, mu, trt_var, trt_effect, error_effect)

knitr::kable(more_norm_data_interp)
```

plot	trt	yield	mu	trt_var	trt_effect	error_effect
1	A	166.8	168.96	16.190000	-2.810000	0.6500000
2	C	169.9	168.96	19.063333	1.373333	-0.4333333
3	B	170.6	168.96	28.996667	-3.010000	4.6500000
4	C	174.9	168.96	19.063333	1.373333	4.5666667
5	B	161.8	168.96	28.996667	-3.010000	-4.1500000
6	A	168.3	168.96	16.190000	-2.810000	2.1500000
7	A	169.2	168.96	16.190000	-2.810000	3.0500000
8	C	NA	168.96	19.063333	1.373333	NA
9	B	160.8	168.96	28.996667	-3.010000	-5.1500000
10	C	166.2	168.96	19.063333	1.373333	-4.1333333
11	A	160.3	168.96	16.190000	-2.810000	-5.8500000
12	D	172.1	168.96	3.416667	4.790000	-1.6500000
13	D	175.4	168.96	3.416667	4.790000	1.6500000
14	D	175.3	168.96	3.416667	4.790000	1.5500000
15	B	170.6	168.96	28.996667	-3.010000	4.6500000
16	D	172.2	168.96	3.416667	4.790000	-1.5500000

We can see that removing the yield data from plot 4 causes the treatment effect of treatment C to change – in fact, it has gone from negative to positive. The other treatment effects have also changed. The within-treatment variance for treatment C has also decreased by about one-third. When we re-run our analysis of variance, we see the treatment effect is 0.062 – almost significant at the P=0.05 level.

```
summary.aov(lm(yield ~ trt, data=more_norm_data_interp))
```

```
##          Df Sum Sq Mean Sq F value Pr(>F)
## trt        3 165.3   55.09   3.294 0.0617 .
## Residuals 11 183.9   16.72
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 1 observation deleted due to missingness
```

What would happen if we estimated the yield for plot 8 using the population mean, mu, and the treatment effect?

$$Y_{3,4} = \mu + T_i = 168.96 + 1.37 = 170.33$$

We see that mu, treatment variance, treatment effect, and error have again changed. The variance within Treatment 3 has again decreased by about one-third.

```

plot_8_interp = data.frame(plot=8 ,trt="C", yield=170.33)

more_norm_data_interp_8 = more_norm_data %>%
  dplyr::filter(!plot==8) %>%
  rbind(plot_8_interp) %>%
  arrange(plot) %>%
  mutate(mu = mean(yield, na.rm = TRUE)) %>%
  group_by(trt) %>%
  mutate(trt_mean = mean(yield, na.rm = TRUE),
        trt_var = var(yield, na.rm = TRUE)) %>%
  ungroup() %>%
  mutate(trt_effect = trt_mean - mu) %>%
  mutate(error_effect = yield - trt_effect - mu) %>%
  dplyr::select(plot, trt, yield, mu, trt_var, trt_effect, error_effect)

knitr::kable(more_norm_data_interp_8)

```

plot	trt	yield	mu	trt_var	trt_effect	error_effect
1	A	166.80	169.0456	16.190000	-2.895625	0.6500
2	C	169.90	169.0456	12.708892	1.286875	-0.4325
3	B	170.60	169.0456	28.996667	-3.095625	4.6500
4	C	174.90	169.0456	12.708892	1.286875	4.5675
5	B	161.80	169.0456	28.996667	-3.095625	-4.1500
6	A	168.30	169.0456	16.190000	-2.895625	2.1500
7	A	169.20	169.0456	16.190000	-2.895625	3.0500
8	C	170.33	169.0456	12.708892	1.286875	-0.0025
9	B	160.80	169.0456	28.996667	-3.095625	-5.1500
10	C	166.20	169.0456	12.708892	1.286875	-4.1325
11	A	160.30	169.0456	16.190000	-2.895625	-5.8500
12	D	172.10	169.0456	3.416667	4.704375	-1.6500
13	D	175.40	169.0456	3.416667	4.704375	1.6500
14	D	175.30	169.0456	3.416667	4.704375	1.5500
15	B	170.60	169.0456	28.996667	-3.095625	4.6500
16	D	172.20	169.0456	3.416667	4.704375	-1.5500

And here is the kicker, wait for it....

```

summary.aov(lm(yield ~ trt, data=more_norm_data_interp_8))

##          Df Sum Sq Mean Sq F value Pr(>F)
## trt          3  167.0   55.67   3.632  0.045 *
## Residuals    12  183.9   15.33
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

Our treatment differences are now significant. Why? Because when we estimate a missing value using only the population mean and treatment effect, we decrease the overall variance. And why does that happen? Because we have now created that is almost exactly equal to the treatment mean. Was there a change in the originally observed values associated with this change in significance? No. And this is problem. But there is a way to reduce it.

The problem with the model we have used so far is we did not include the error effect in our yield estimate. If we added it in, our yield estimate for plot 8 would be more appropriate. Of course, we cannot calculate the error effect because it is random and changes among plots. But, knowing that error effects are normally distributed around the treatment mean, we can model that distribution and draw an individual from it at random, to use as the error effect in our estimate.

The error distribution has its own mean, which should be close to zero:

```
err_mean = mean(more_norm_data_interp$error_effect, na.rm=TRUE)
err_mean
```

```
## [1] 1.894781e-15
```

And its own standard deviation:

```
err_sd = sd(more_norm_data_interp$error_effect, na.rm=TRUE)
err_sd
```

```
## [1] 3.624684
```

Knowing these two parameters, we can select a value for our error effect from that distribution.

```
set.seed(12)
err_pred = rnorm(1, err_mean, err_sd)
err_pred
```

```
## [1] -5.36659
```

Let's plug that into our yield estimate and see how our statistics change.

$$Y_{3,4} = \mu + T_i = 168.96 + 1.37 - 5.37 = 164.96$$

We see that mu, treatment variance, treatment effect, and error have again changed. The variance within Treatment 3 has again decreased by about one-third.

```

plot_8_interp_sd = data.frame(plot=8 ,trt="C", yield=164.96)

more_norm_data_interp_8_sd = more_norm_data %>%
  dplyr::filter(!plot==8) %>%
  rbind(plot_8_interp_sd) %>%
  arrange(plot) %>%
  mutate(mu = mean(yield, na.rm = TRUE)) %>%
  group_by(trt) %>%
  mutate(trt_mean = mean(yield, na.rm = TRUE),
        trt_var = var(yield, na.rm = TRUE)) %>%
  ungroup() %>%
  mutate(trt_effect = trt_mean - mu) %>%
  mutate(error_effect = yield - trt_effect - mu) %>%
  dplyr::select(plot, trt, yield, mu, trt_var, trt_effect, error_effect)

knitr::kable(more_norm_data_interp_8_sd)

```

plot	trt	yield	mu	trt_var	trt_effect	error_effect
1	A	166.80	168.71	16.190000	-2.56	0.65
2	C	169.90	168.71	19.927067	0.28	0.91
3	B	170.60	168.71	28.996667	-2.76	4.65
4	C	174.90	168.71	19.927067	0.28	5.91
5	B	161.80	168.71	28.996667	-2.76	-4.15
6	A	168.30	168.71	16.190000	-2.56	2.15
7	A	169.20	168.71	16.190000	-2.56	3.05
8	C	164.96	168.71	19.927067	0.28	-4.03
9	B	160.80	168.71	28.996667	-2.76	-5.15
10	C	166.20	168.71	19.927067	0.28	-2.79
11	A	160.30	168.71	16.190000	-2.56	-5.85
12	D	172.10	168.71	3.416667	5.04	-1.65
13	D	175.40	168.71	3.416667	5.04	1.65
14	D	175.30	168.71	3.416667	5.04	1.55
15	B	170.60	168.71	28.996667	-2.76	4.65
16	D	172.20	168.71	3.416667	5.04	-1.55

When we look at the ANOVA, we see that the mean square and p-value are approximately the same as they were before the missing value was interpolated.

```

summary.aov(lm(yield ~ trt, data=more_norm_data_interp_8_sd))

##           Df Sum Sq Mean Sq F value Pr(>F)
## trt          3 158.6   52.87   3.086  0.068 .
## Residuals    12 205.6   17.13
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

In R, there is a nice package called *mice* that does this for us. We will learn about it in an exercise this week.

One final note, however: we should not interpolate missing values if 5% or more of the data are missing. Why? Because, as we have seen above, that interpolated value can markedly change our interpretation of the data. Restricting interpolation to datasets where a small percentage of data are missing reduces the leverage one observation has on conclusions from the data. It also increases the accuracy of the interpolated values.

In smaller datasets, then, we may to use the approaches above. How important is it to have that treatment in the trial, versus losing a replicaton of the other treatments? For the latter option, you may want to test whether including or omitting that replicate changes your conclusion from the data. If not, it may be easiest to drop the replication.

## 9.5 Summary

Agronomic data are rarely nice and neat. The scale in which we work (acres and hectares), plus the variations in soil types, equipment performance, weather, weeds and other pests, make it virtually impossible to ensure our experimental units are near-equivalent. Above all, our subjects are alive and integrate every aspect of their environment into their growth. Unlike human subjects, corn plants cannot answer a history questionnaire. Months or years of a trial can be erased by one plugged nozzle, one broken singulator, one strong wind, one “white combine”. It’s a nasty business.

It is important that we inspect our data and consider its shortcomings. We have ways to address these shortcomings. Outliers may be trimmed, or we may use other techniques to overcome them. We have also learned how to re-scale data (using logarithms or square roots) so that variances are closer to equal, and how to “fill in” missing values using imputation so that our datasets can be balanced.

If stuck, consult with other data scientists. There are “robust” statistics that are based on using the median, rather than the mean, for summaries and tests. There are also non-parametric tests, some of which we will be introduced to towards the end of this course. Non-parametric methods don’t use linear models – therefore, they are more insulated from the problems discussed above. We will learn about these in a future unit.

## 9.6 Exercise: Visual Data Inspection

Never underestimate the power of data visualizations. Our ability to see patterns in data is profound. In this exercise, we will review how to create his-

tograms and learn new tools, including the rank-percentile, box-plot, and mean-variance plot.

### 9.6.1 Case Study 1

We will work with a wheat uniformity trial. In uniformity trials, there are no treatments. Instead, the wheat yield is measured in gridded plots within a location in order to get a sense of yield consistency (uniformity) or inconsistency (variance).

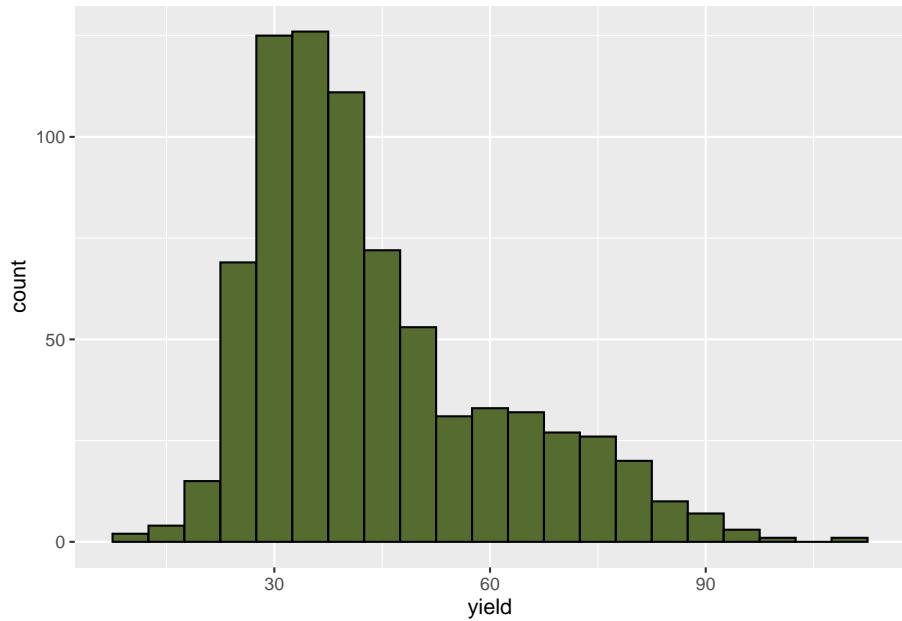
```
library(tidyverse)
wheat_uniformity = read.csv("data-unit-9/exercise_data/wheat_uniformity.csv")
head(wheat_uniformity)
```

```
##   row col yield
## 1   1   1    60
## 2   2   1    78
## 3   3   1    94
## 4   4   1    77
## 5   5   1    55
## 6   6   1    46
```

### 9.6.2 Histogram

At the beginning of this course, we learned how to use the histogram to quickly learn about the distribution of a dataset. This is just a reminder how powerful a tool it is. We will also relate this shape to that of the rank-percentile plot below. note the data are skewed to the right, with wide tails at both the top and bottom of the distribution.

```
wheat_uniformity %>%
  ggplot(aes(x=yield)) +
  geom_histogram(fill="darkolivegreen", color="black", binwidth = 5)
```



### 9.6.3 Rank-Percentile

In a rank-percentile plot, data are sorted from least to greatest and then ranked. This takes just one line of code in R.

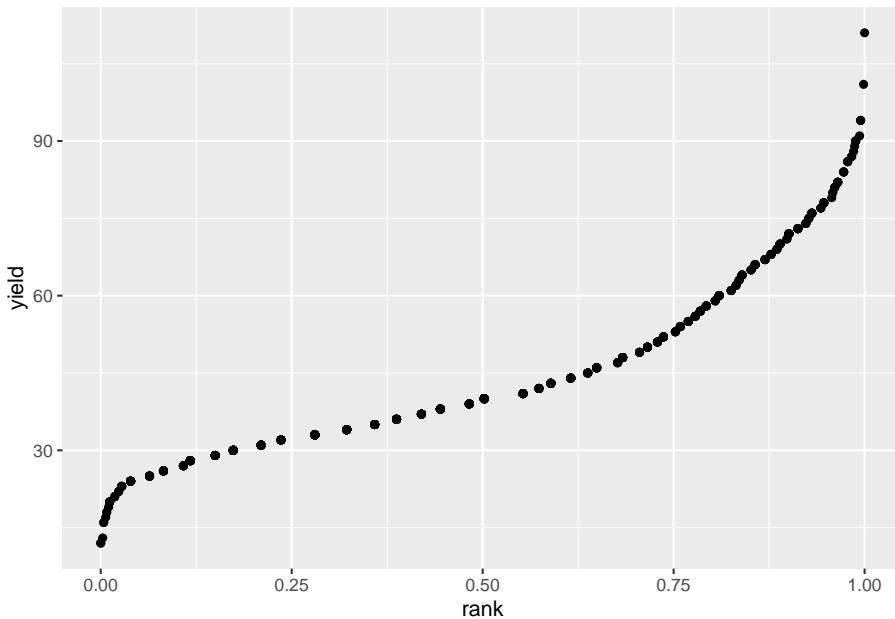
```
ranked_wheat = wheat_uniformity %>%
  mutate(rank = percent_rank(yield))

head(ranked_wheat)

##   row col yield      rank
## 1   1   1    60 0.8096480
## 2   2   1    78 0.9465450
## 3   3   1    94 0.9947849
## 4   4   1    77 0.9426336
## 5   5   1    55 0.7692308
## 6   6   1    46 0.6492829
```

After ranking the observed values, we create a simple scatter plot of observed value (yield, in this case) and rank.

```
ranked_wheat %>%
  ggplot(aes(x=rank, y=yield)) +
  geom_point()                                # this tells R what dataset to use
                                                # this creates a plot where rank and yield will determine the points
                                                # this will draw the points
```



We notice two things about this curve. First, it is not symmetric around the median, reflecting the skewed distribution we observed in the histogram above. Second, if we were to fit a line to the lower two-thirds of the data, we would notice that the entire 75 - 100 percentile range would be above the line. This suggests a long tail to the right (above the mean). That the line dips down at the left side of the curve also suggests that tail is longer than normal.

#### 9.6.4 Case Study 2

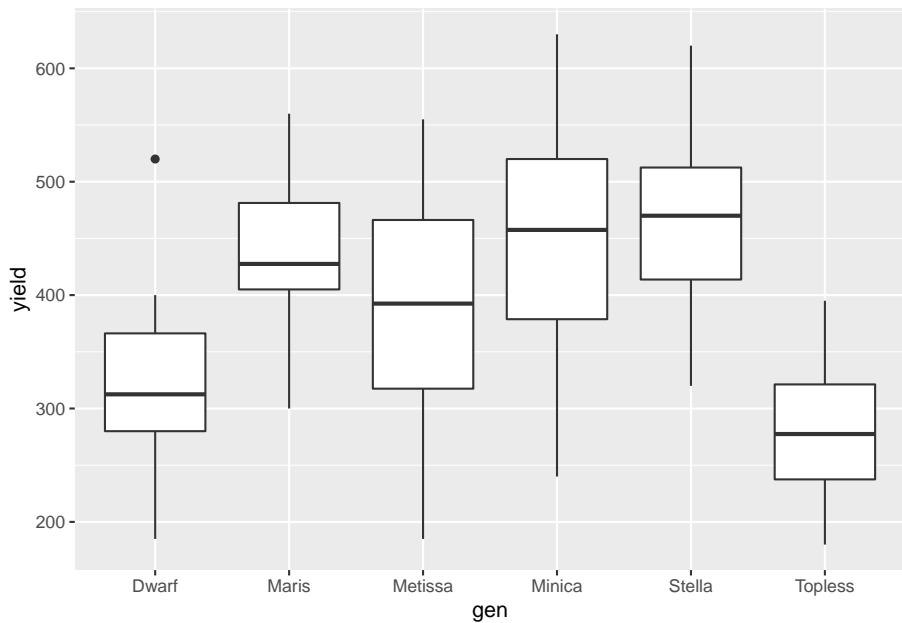
For the box plot, we will use a study of common bean genotypes.

```
bean_genotypes = read.csv("data-unit-9/exercise_data/bean_genotypes.csv")
```

#### 9.6.5 Box Plot

The basic box plot can be drawn using the `geom_boxplot()` function.

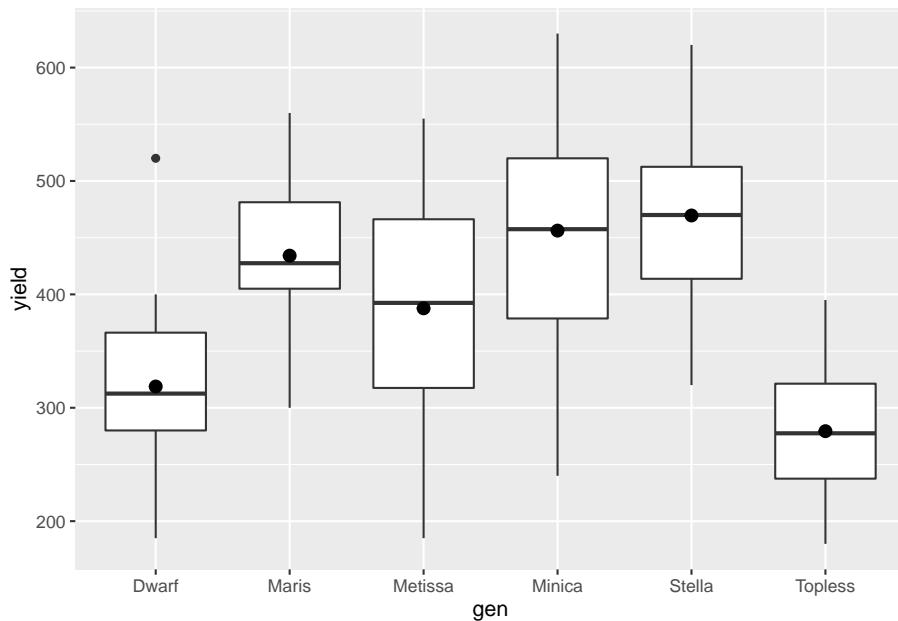
```
bean_genotypes %>%
  ggplot(aes(x=gen, y=yield)) +           # dataset to use
  geom_boxplot() +                         # use gen and type to specify x and y position
                                           # draw box and whisker plots
```



Often a box plot includes the mean. We can add this using the `stat_summary()` command and `fun=mean`. The mean will be represented by a black dot.

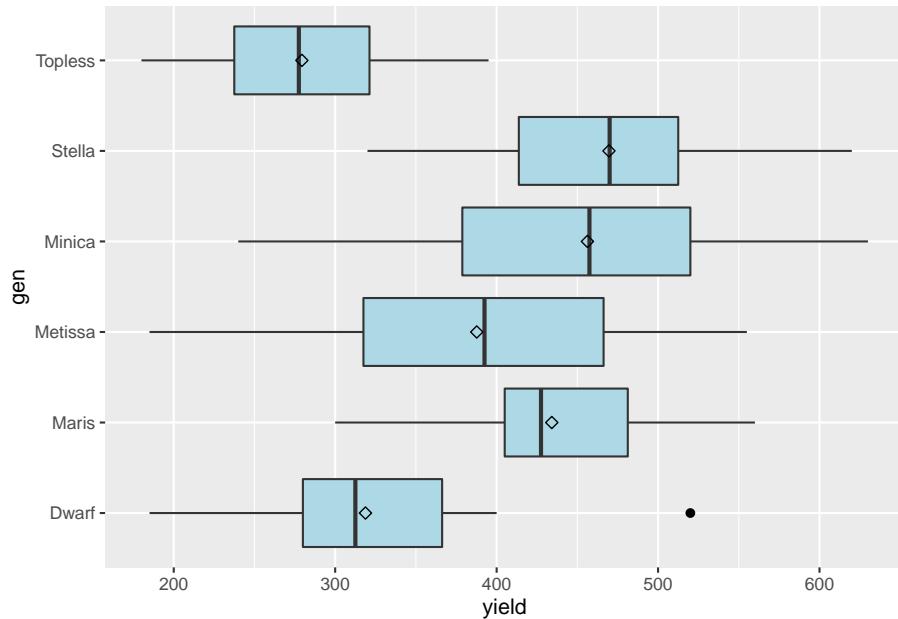
```
bean_genotypes %>%
  ggplot(aes(x=gen, y=yield)) +
  geom_boxplot() +
  stat_summary(fun=mean)                  # dataset to use
                                         # use gen and type to specify x and y position
                                         # draw box and whisker plots
                                         # draw the mean as a point on the plots
```

`## Warning: Removed 6 rows containing missing values (geom_segment).`



We can replicate the style from the lecture using the code below. Just change out the variable names in the second line to use it with other datasets.

```
bean_genotypes %>%
  ggplot(aes(x=gen, y=yield)) +
  geom_boxplot(outlier.colour="black", outlier.shape=16,
               outlier.size=2, notch=FALSE, fill="lightblue") +
  coord_flip() +
  stat_summary(fun=mean, geom="point", shape=23, size=2) # change out the variable names j
```



### 9.6.6 Practice

#### 9.6.6.1 Strawberry Uniformity

Load the “data-unit-9/exercise\_data/strawberry\_uniformity.csv” dataset. Create a rank probability plot. Your plot should look like:

## 9.7 Exercise: Identifying Unequal Variances”

Occassionally, we may encounter data that don’t satisfy our assumptions about variance: that it is equal across the treatments in our experiment or, similarly, not correlated with treatment level. In these cases, it may be useful to transform the data prior to analysis, by calculating and working with the natural logarithms of the original data.

#### 9.7.1 Case Study: Grape Colapsis in Seed Corn

In this case study, we will use data loosely inspired by graduate work by Benjamin Carl Kaeb at Iowa State University, where the researcher had to log-transform data prior to analyses. The measurement is grape colapsis count per liter of soil, after being treated during the season with various numbers of insecticide application.

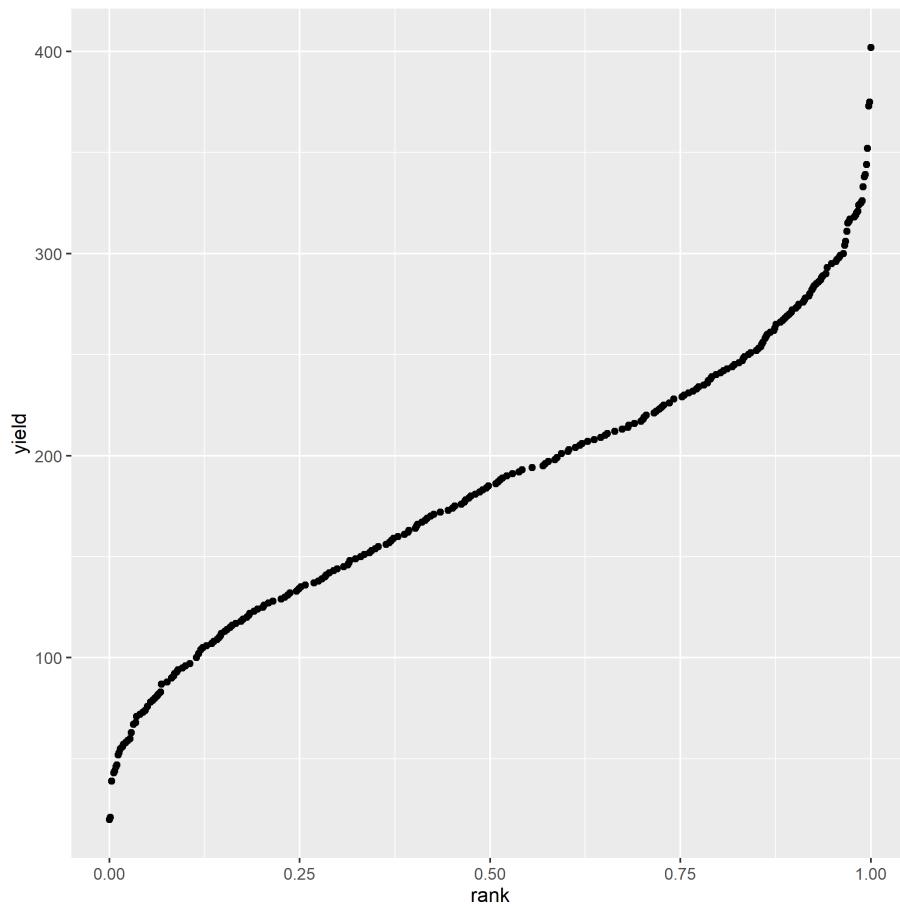


Figure 9.1: strawberry rank probability

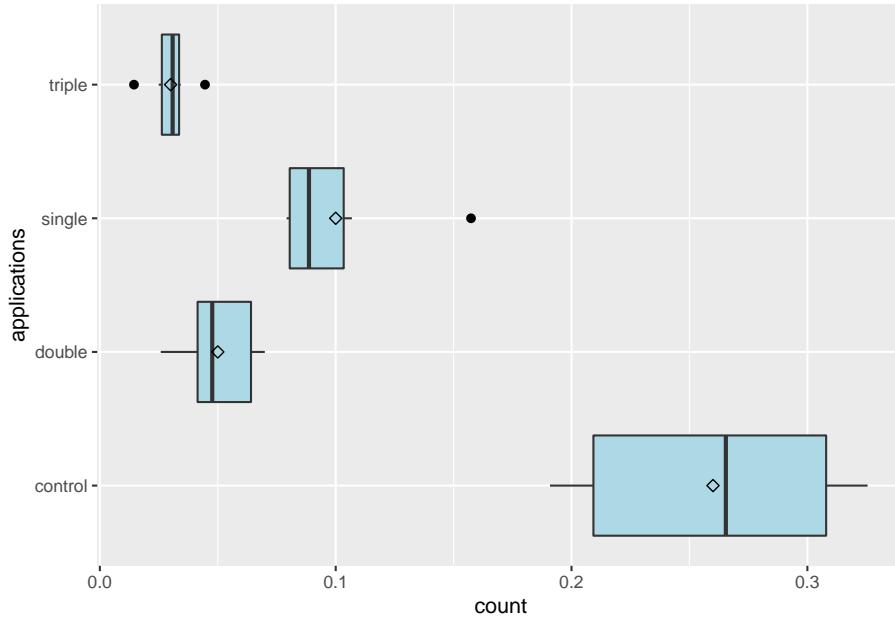
First, lets load and inspect the data.

```
library(tidyverse)
colapsis = read.csv("data-unit-9/exercise_data/grape_colapsis.csv")
head(colapsis)

##   applications      count
## 1       triple 0.02496295
## 2       triple 0.01446486
## 3       triple 0.03428771
## 4       triple 0.03020649
## 5       triple 0.04458152
## 6       triple 0.03149647
```

Next, lets create a box plot of our data.

```
colapsis %>%
  ggplot(aes(x=applications, y=count)) +
  geom_boxplot(outlier.colour="black", outlier.shape=16,
               outlier.size=2, notch=FALSE, fill="lightblue") +
  coord_flip() +
  stat_summary(fun=mean, geom="point", shape=23, size=2)
```

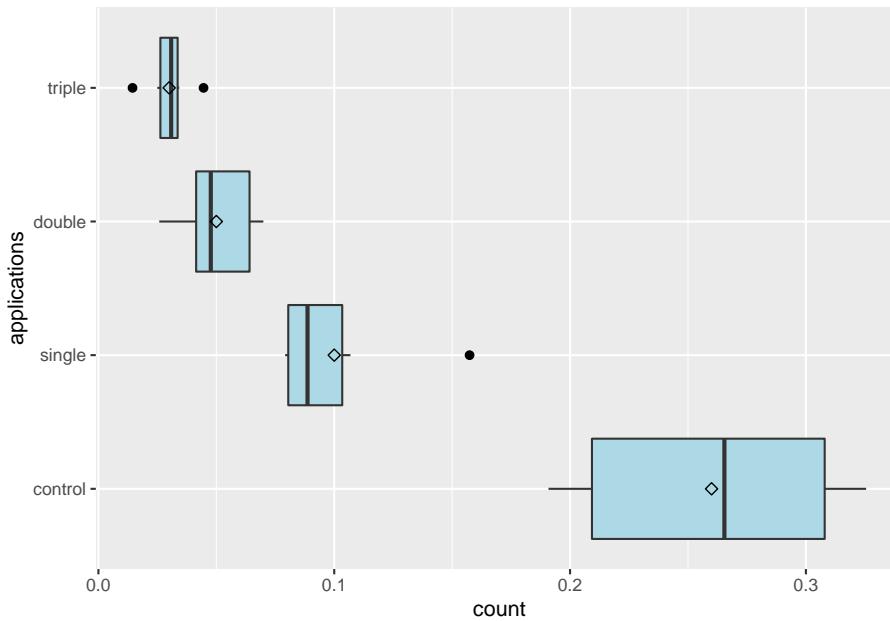


One thing we notice in the plot above is that our applications are not ordered from lowest (control) to highest (triple). Lets fix that. We can use the `factor()`

function to rearrange the order of our levels. We tell R to set the “applications” column of the colapsis dataset as the factor format of itself. The `levels()` argument tells R the order in which the levels should be listed.

```
colapsis$applications = factor(colapsis$applications, levels=c("control", "single", "double", "triple"))

colapsis %>%
  ggplot(aes(x=applications, y=count)) +
  geom_boxplot(outlier.colour="black", outlier.shape=16,
               outlier.size=2, notch=FALSE, fill="lightblue") +
  coord_flip() +
  stat_summary(fun=mean, geom="point", shape=23, size=2)
```



We can see that there is noticeable variation in the spread of observed values among treatments, and that the spread of the observed values seems to increase with the mean number of counts.

Let's calculate our means and variances by number of applications.

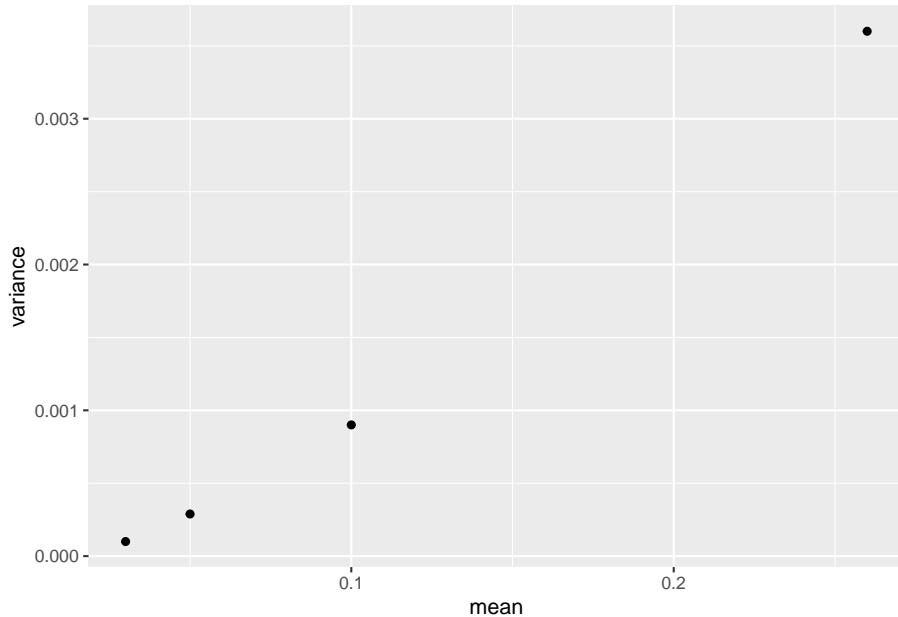
```
colapsis_mean_var = colapsis %>%
  group_by(applications) %>%
  summarise(mean = mean(count),
            variance = var(count)) # create new variable, variance, which is the count variance

colapsis_mean_var
```

```
## # A tibble: 4 x 3
##   applications  mean  variance
##   <fct>        <dbl>    <dbl>
## 1 control      0.26    0.00360
## 2 single       0.1     0.000900
## 3 double       0.05    0.000289
## 4 triple       0.0300  0.000100
```

Here is a simple scatter plot of variance as a function of mean.

```
colapsis_mean_var %>%
  ggplot(aes(x=mean, y=variance)) +
  geom_point()
```



There is a clear trend that variance increases with mean. We can also see the greatest variance is about 16 times greater than the least variance. This is problematic – when we have differences greater than 2X in variances, our analyses may not function properly.

To test whether the variances are equal, we will use the *bartlett.test()* function. The arguments to this function are the same as a model statement: the model, and the data frame

```
bartlett.test(count ~ applications, data=colapsis)
```

```
##
```

```
##  Bartlett test of homogeneity of variances
##
## data: count by applications
## Bartlett's K-squared = 14.636, df = 3, p-value = 0.002156
```

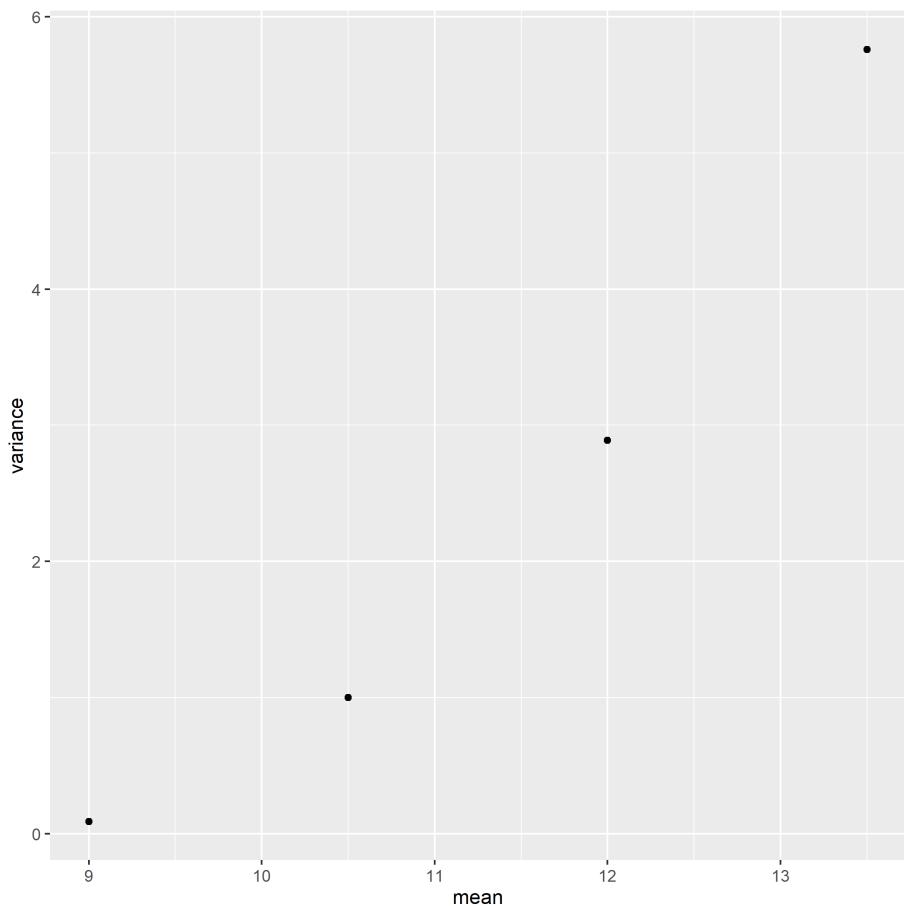
We see that the p-value is less than 0.01. We should consider transforming these data before running an analysis of variance and means separation on them.

### 9.7.2 Practice: Pigweed Height

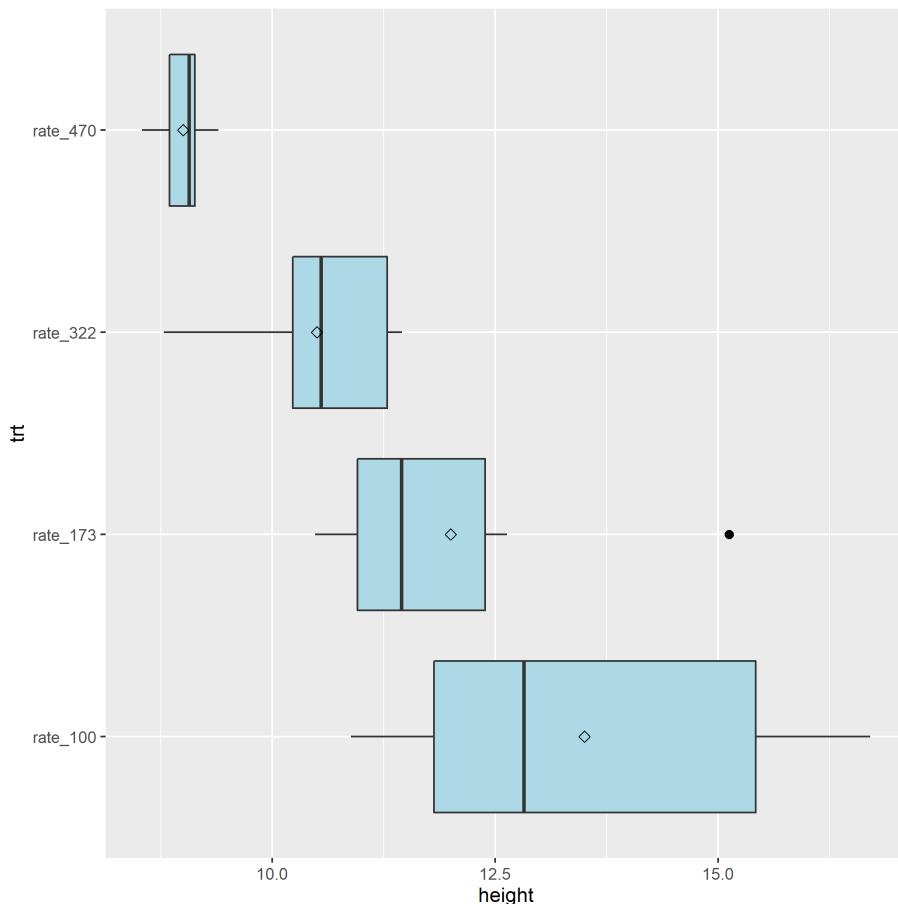
1. Load the data from “data-unit-9/exercise\_data/soybean\_pigweed\_height.csv”. “trt” is the soybean seeding rate. “height” is the height of pigweed species in inches.
2. Create a summary table of means and variances. Your results should look like.

```
trt mean variance rate_100 13.5 5.76 rate_173 12.0 2.89 rate_322 10.5 1.00
rate_470 9.0 0.09
```

3. Create a plot of variance as a function of mean. It should look like:



4. Create a boxplot of height grouped by trt. It should look like this.



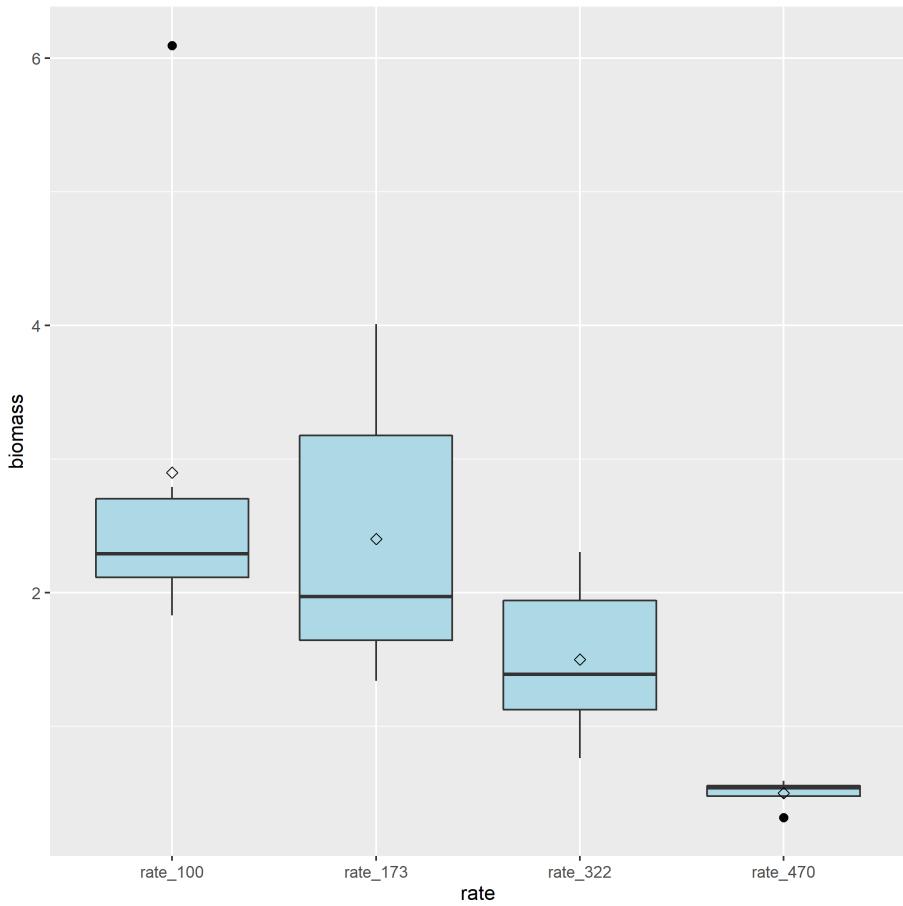
5. Run the Bartlett's Test for Homogeneity of the variances by treatment.  
Your output should look like:

```
Bartlett test of homogeneity of variances
```

```
data: height by trt
Bartlett's K-squared = 14.564, df = 3, p-value = 0.00223
```

```
### Practice: Pigweed Biomass
Pigweed biomass was also measured in the
above experiment.
```

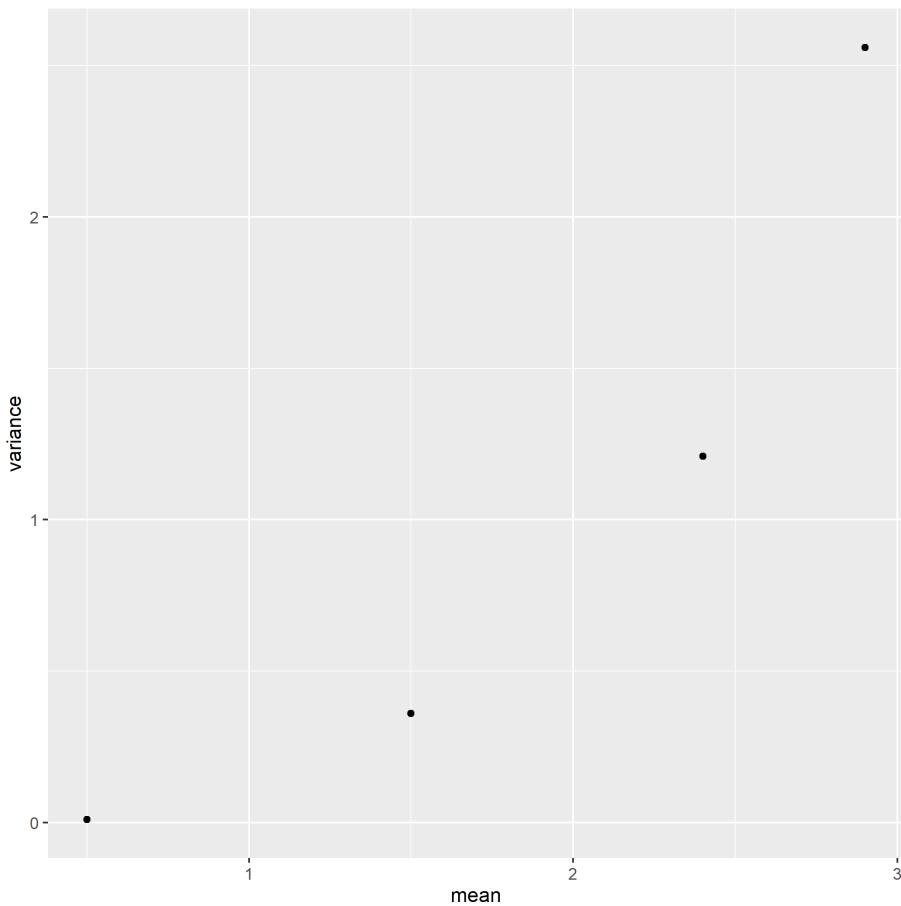
1. Load the dataset “data-unit-9/exercise\_data/soybean\_pigweed\_biomass.csv”
2. Create a boxplot of biomass by seeding rate



3. Create a table of biomass means and variances by seeding rate. Your table should look like:

rate	mean	variance
rate_100	2.9	2.56
rate_173	2.4	1.21
rate_322	1.5	0.36
rate_470	0.5	0.01

4. Plot the variance of biomass as a function of seeding rate. Your plot should look like:



5. Run a Bartlett Test of Homogeneity of Variances on the data. Your results should look like:

```
Bartlett test of homogeneity of variances
```

```
data: biomass by rate
Bartlett's K-squared = 21.387, df = 3, p-value = 8.749e-05
```

## 9.8 Exercise: Transforming and Analyzing Data

In this exercise, we will work with the same three datasets in which we identified unequal variances.

### 9.8.1 Case Study: Grape Colapsis in Seed Corn

First, lets load and inspect the data.

```
library(tidyverse)
colapsis = read.csv("data-unit-9/exercise_data/grape_colapsis.csv")
```

In the previous exercise, we determined the variances were unequal, so we will now create a new column in our colapsis dataset with the natural logarithm of the original counts.

```
colapsis = colapsis %>%
  mutate(log_count = log(count)) # create new variable count_log, based on natural log
```

We can run an Analysis of Variance on the transformed data, the same as we would any other dataset.

```
colapsis_model = aov(log_count ~ applications, data=colapsis)
summary(colapsis_model)
```

```
##           Df Sum Sq Mean Sq F value    Pr(>F)
## applications  3 16.247   5.416  53.03 1.06e-09 ***
## Residuals    20  2.043   0.102
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Our applications effect is significant. We will separate the means using the Least Significant Difference.

```
library(agricolae)
lsd = LSD.test(colapsis_model, "applications")

lsd

## $statistics
##      MSerror Df      Mean       CV t.value      LSD
## 0.1021256 20 -2.578757 -12.39244 2.085963 0.3848694
##
## $parameters
##      test p.adjusted      name.t ntr alpha
## Fisher-LSD     none applications  4  0.05
##
## $means
```

```

##      log_count      std r      LCL      UCL      Min      Max      Q25
## control -1.370036 0.2364918 6 -1.642180 -1.097892 -1.656177 -1.122266 -1.564864
## double  -3.049637 0.3716377 6 -3.321781 -2.777493 -3.656836 -2.660004 -3.183980
## single  -2.334127 0.2634909 6 -2.606270 -2.061983 -2.536044 -1.849017 -2.519757
## triple  -3.561230 0.3808307 6 -3.833373 -3.289086 -4.236033 -3.110436 -3.642697
##          Q50      Q75
## control -1.338709 -1.177759
## double  -3.047062 -2.754693
## single  -2.424058 -2.270959
## triple  -3.478789 -3.394196
##
## $comparison
## NULL
##
## $groups
##      log_count groups
## control -1.370036     a
## single  -2.334127     b
## double  -3.049637     c
## triple  -3.561230     d
##
## attr(,"class")
## [1] "group"

```

At this point, we have all the data we need to evaluate whether the applications effect is significant, and whether the colapsis count differs with the number of applications per season. Before we present our means and groupings above, however, we would like to “back-transform” them to our original scale of measurement.

Recall that we can quickly isolate the \$groups table from our LSD.test output. We will want to convert the rownames to a column before we work further with that data frame, however. We can use the *rownames\_to\_columns()* function to create a new column, “applications”, with the values from the row names.

```

means_from_lsd = lsd$groups
means_from_lsd = means_from_lsd %>%
  rownames_to_column(var = "applications")

```

We will now back-transform the natural log by using  $e$ , Euler’s constant (equal to about 2.78) to “cancel it out”. Specifically, we will tell R to calculate  $e^{\hat{x}}$ , where  $x$  is each treatment mean in the transformed data.

```

means_from_lsd = means_from_lsd %>%
  mutate(count = exp(log_count))

```

```
means_from_lsd

##   applications log_count groups      count
## 1       control -1.370036     a 0.25409776
## 2       single -2.334127     b 0.09689507
## 3      double -3.049637     c 0.04737613
## 4      triple -3.561230     d 0.02840387
```

Finally, let's get rid of the log\_count column and move the groups column to the column furthest right. We will use the `select()` function.

```
means_from_lsd = means_from_lsd %>%
  dplyr::select(applications, count, groups)

means_from_lsd

##   applications      count groups
## 1       control 0.25409776     a
## 2       single 0.09689507     b
## 3      double 0.04737613     c
## 4      triple 0.02840387     d
```

### 9.8.2 Practice: Pigweed Height

1. Load the pigweed height data from the “data/soybean\_pigweed\_height.csv” file.

```
pigweed = read.csv("data-unit-9/exercise_data/soybean_pigweed_height.csv")
```

2. Transform the data using the natural log transformation.

```
# The first few rows will look like:
# trt      height      log_height
# rate_100 11.69514  2.459173
# rate_100 16.06706  2.776771
# rate_100 10.88049  2.386971
# rate_100 13.48499  2.601577
# rate_100 16.70968  2.815988
# rate_100 12.16264  2.49836
```

3. Run the ANOVA.

```
# Your results should look like:
#              Df Sum Sq Mean Sq F value    Pr(>F)
# trt          3 0.5162 0.17206   11.6 0.000127 ***
# Residuals   20 0.2966 0.01483
```

4. Run the LSD test and extract the groups table.
5. Convert the rownames to a column.

```
# Your table should look like.
# trt  log_height  groups
# rate_100 2.589808    a
# rate_173 2.477182    ab
# rate_322 2.347396    b
# rate_470 2.196758    c
```

6. Back-transform the means to the original scale.

```
# Your results should look like:
# trt  log_height  groups  height
# rate_100 2.589808    a   13.327217
# rate_173 2.477182    ab  11.907663
# rate_322 2.347396    b   10.458302
# rate_470 2.196758    c   8.995803
```

7. Use select() to arrange the table so the columns are: trt, height, groups.

```
# Your results should look like:
# trt  height  groups
# rate_100 13.327217    a
# rate_173 11.907663    ab
# rate_322 10.458302    b
# rate_470  8.995803    c
```

### 9.8.3 Practice: Pigweed Biomass

1. Load the pigweed biomass data from the “data/soybean\_pigweed\_biomass.csv” file.

```
biomass = read.csv("data-unit-9/exercise_data/soybean_pigweed_biomass.csv")
```

2. Transform the data using the natural log transformation.

```
# The first few rows will look like:
# rate biomass log_biomass
# rate_100 6.096351 1.8076904
# rate_100 1.828959 0.6037471
# rate_100 2.444420 0.8938077
# rate_100 2.789826 1.0259793
# rate_100 2.105895 0.7447407
# rate_100 2.134549 0.7582552
```

3. Run the ANOVA.

```
# Your results should look like:
#           Df Sum Sq Mean Sq F value Pr(>F)
# rate      3 10.280  3.427   22.45 1.3e-06 ***
# Residuals 20  3.053  0.153
```

4. Run the LSD test and extract the groups table. Your results should look like:

```
# Your results should look like:
# log_biomass groups
# rate_100  0.9723701  a
# rate_173  0.7920214  ab
# rate_322  0.3350568  b
# rate_470  -0.7132060 c
```

5. Convert the rownames to a column.

```
# Your table should look like:
# rate log_biomass groups
# rate_100 0.9723701  a
# rate_173 0.7920214  ab
# rate_322 0.3350568  b
# rate_470 -0.7132060 c
```

6. Back-transform the means to the original scale.

```
# Your results should look like:
# rate log_biomass groups  biomass
# rate_100 0.9723701  a  2.6442039
# rate_173 0.7920214  ab 2.2078550
# rate_322 0.3350568  b  1.3980198
# rate_470 -0.7132060 c  0.4900705
```

7. Use `select()` to arrange the table so the columns are: `trt`, `height`, `groups`.

```
# Your results should look like:
# rate biomass groups
# rate_100 2.6442039 a
# rate_173 2.2078550 ab
# rate_322 1.3980198 b
# rate_470 0.4900705 c
```

## 9.9 Exercise: Imputing Missing Data

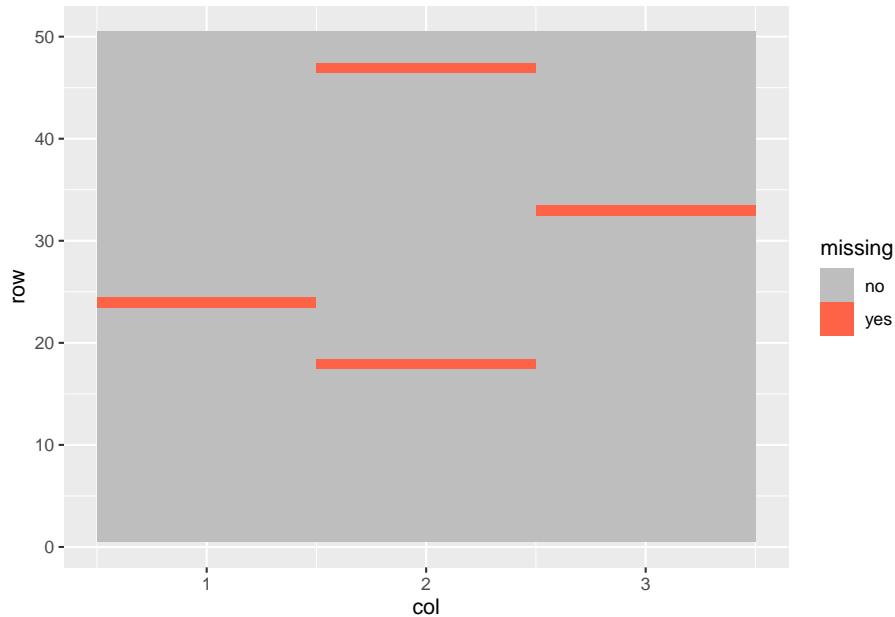
Despite our best efforts, we occassionally encounter missing data. In some cases, plots may be lost to inclement weather, equipment failures, or human error. In other cases, we may fail to record values for every plot. In some cases, we may delete an outlier. In each of these cases, we may wish to replace the missing value in order to maintain a complete dataset.

### 9.9.1 Case Study: Wheat Genotype Trial

For this example, we will use data from a wheat genotype trial.

```
library(tidyverse)
wheat_missing = read.csv("data-unit-9/exercise_data/wheat_missing.csv")

wheat_missing %>%
  mutate(missing = if_else(is.na(yield), "yes", "no")) %>%
  mutate(missing = as.factor(missing)) %>%
  ggplot(aes(x=col, y=row)) +
  geom_tile(aes(fill=missing)) +
  scale_fill_manual(values = c("grey", "tomato"))
```



### 9.9.1.1 Mean Imputation

When we estimate the values of missing data, we *impute* those values. The simplest way to impute a missing value is to use the mean for that treatment. For example, we could calculate the mean for each treatment, as we have done below.

```
genotype_means = wheat_missing %>%
  group_by(gen) %>%
  mutate(mean_yield = mean(yield, na.rm = TRUE)) %>%
  ungroup()

head(genotype_means)
```

```
## # A tibble: 6 x 5
##   col  row  gen  yield mean_yield
##   <int> <int> <chr> <dbl>      <dbl>
## 1     1     1  G01    3.38      3.40
## 2     1     2  G02    3.59      3.09
## 3     1     3  G03    4.33      3.05
## 4     1     4  G04    4.54      3.56
## 5     1     5  G05    2.59      2.30
## 6     1     6  G06    2.41      2.53
```

The mean yield is shown in the far right column. We will now use either the recorded value (“yield”) or the mean yield (where “yield” is a missing value) for our final yield. Don’t worry about the code below (although conditional statements *are* cool) – it is just to set up the rest of this example.

```
mean_imputed_yields = genotype_means %>%
  mutate(final_yield = if_else(is.na(yield), mean_yield, yield))
```

We now have a complete dataset. But there is a problem. Lets look at the variance before imputing the missing values. We calculate variance using the `var()` function. Whenever we are calculating a summary statistic – mean, sd, variance, etc) – we can add `na.rm = TRUE` to tell R to work around the missing values. Otherwise, R will return “NA” instead of the statistic.

Here is the variance before:

```
var(wheat_missing$yield, na.rm = TRUE)

## [1] 0.6638558
```

And after:

```
var(mean_imputed_yields$final_yield)

## [1] 0.6492327
```

What do you notice? That’s right, the variance is decreased between the original dataset and the imputed dataset. When we use means imputation, we artificially lower the variance. This increases the probability we will commit a Type I error: determining that two or more populations are significantly different when in fact they are not. That is why it is better to use the next

### 9.9.1.2 Multivariate Imputation by Chained Equations (MICE)

The MICE algorithm is complex and a detailed explanation is beyond the scope of this course. What you need to know is that

- Instead of estimating a missing value with its treatment mean MICE imputes the missing value using all other variables (hence “multivariate”) in the dataset. In other words, it builds a model and solves for the missing value.
- Missing values are imputed in a way that preserves the original variance of the dataset.

Here is how we use MICE imputation. The algorithm is from the package *mice*. The function is also called `*mice()`. It takes two arguments. First, the name of the dataset with missing values. Second, since MICE works with random numbers, we should set a seed so that we can recreate the exact same imputed values in the future, if necessary. The third argument, “`printFlag = FALSE`”, tells R not to print a bunch extraneous information.

```
library(mice)

## 
## Attaching package: 'mice'

## The following object is masked from 'package:timeSeries':
## 
##     filter

## The following object is masked from 'package:stats':
## 
##     filter

## The following objects are masked from 'package:base':
## 
##     cbind, rbind

wheat_imputed = mice(wheat_missing, seed=1, printFlag = FALSE) # arguments to mice: . . .

## Warning: Number of logged events: 1
```

The output of the `mice()` function is a list with a bunch of information about how the data were imputed: the original dataset, the models used, etc. We just want the complete dataset, which we can get using the `complete()` function.

```
wheat_complete = complete(wheat_imputed)
```

Finally, let's compare the variance of the MICE-imputed dataset to that of the original dataset:

```
var(wheat_complete$yield)

## [1] 0.6580733
```

We can see the variance is decreased slightly from the original dataset, but not as much as when we imputed the data using the treatment means.

### 9.9.2 Practice 1

The yield is missing from row 31 in this dataset. Fill in the code below. Impute the missing value using MICE. If you use seed=3, your imputed value should be 1.402.

```
library(mice)
peanut = read.csv("data-unit-9/exercise_data/peanut_missing.csv")
peanut_imputed = mice(peanut, seed=34, printFlag = FALSE) # arguments to mice: 1) name of dataset

## Warning: Number of logged events: 2

peanut_complete = complete(peanut_imputed)

peanut[31,]

##     rep    gen yield
## 31   R3  mf447     NA

peanut_complete[31,]

##     rep    gen yield
## 31   R3  mf447  1.4588
```

### 9.9.3 Practice 2

Impute the missing value in the “data/barley\_missing.csv” dataset. Observations 18 and 47 are missing their yield value. Use seed=3. Your imputed values should be obs 18 = 377.9 and obs 47 = 362.6

```
barley_missing = read.csv("data-unit-9/exercise_data/barley_missing.csv")
str(barley_missing)

## 'data.frame': 60 obs. of 4 variables:
## $ col : int 1 3 6 9 11 14 17 19 22 25 ...
## $ row : int 1 1 1 1 1 1 1 1 1 1 ...
## $ gen : chr "a" "c" "b" "a" ...
## $ yield: num 236 291 290 292 321 ...

barley_imputed = mice(barley_missing, seed=3, printFlag = FALSE) # arguments to mice: 1) name of dataset

## Warning: Number of logged events: 1
```

```
barley_complete = complete(barley_imputed)
```

```
barley_missing[18,]
```

```
##   col row gen yield  
## 18  14    2     c     NA
```

```
barley_complete[18,]
```

```
##   col row gen yield  
## 18  14    2     c 302.2
```

## Chapter 10

# Correlation and Simple Regression

Let's review our progress so far.

In Units 1 to 3, we learned about populations, distributions, and samples. A population was a group of individuals in which we were interested. We use statistics to describe the spread or distribution of a population. When it is not possible to measure every individual in a population, a sample or subset can be used to estimate the frequency with which different values would be observed were the whole population measured.

In Units 4 and 5, we learned how to test whether two populations were different. The t-distribution allowed us to calculate the probability the two populations were the same, in spite of a measured difference. When the two populations were managed differently, the t-test could be used to test whether they, and therefore their management practices, caused different outcomes.

In Units 6 and 7, we learned how to test differences among multiple qualitative treatments. By qualitative, we mean there was no natural ranking of treatments: they were identified by name, and not a numeric value that occurred along a scale. Different hybrids, herbicides, and cropping systems are all examples of qualitative treatments. Different experimental designs allowed us to reduce error, or unexplained variation among experimental units in a trial. Factorial trials allowed us to test multiple factors and once, as well as test for any interactions among factors.

In Unit 8, we learned about to test for differences among multiple qualitative treatments. The LSD and Tukey tests can be used to test the difference among treatments. Contrasts can be used to compare intuitive groupings of treatments. We learned how to report results in tables and plots.

In Units 9 and 10, we will learn to work with quantitative treatments. Quantitative treatments can be ranked. The most obvious example would be rate trials for fertilizers, crop protection products, or crop seeds. What makes this situation different is that we are trying to describe a relationship between  $x$  and  $y$  along within a range of  $x$  values, rather at only at discrete values of  $x$ .

## 10.1 Correlation

There are two ways of analyzing the relationship between two quantitative variables. If our hypothesis is that an change in  $x$ , the independent variable (for example, pre-planting nitrogen fertilization rate), *causes* a change in  $y$ , the dependent variable (for example, yield), then we use a *regression model* to analyze the data. Not only can the relationship between tested for significance – the model itself can be used to predict  $y$  for any value of  $x$  within the range of those in the dataset.

Sometimes, however, we don't know whether  $x$  causes  $y$ , or  $y$  causes  $x$ . This is the chicken-and-egg scenario. Outside animal science, however, we can run into this situation in crop development when we look at the allocation of biomass to different plant parts or different chemical components of tissue.

### 10.1.1 Case Study: Cucumber

In this study, cucumbers were grown the their number of leaves, branches, early fruits, and total fruits were measured.

First, let's load the data and look at its structure.

```
library(tidyverse)

## -- Attaching packages ----- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr    0.3.4
## v tibble  3.1.6     v dplyr    1.0.8
## v tidyverse 1.2.0    v stringr   1.4.0
## v readr   2.1.2     vforcats  0.5.1

## Warning: package 'ggplot2' was built under R version 4.1.3

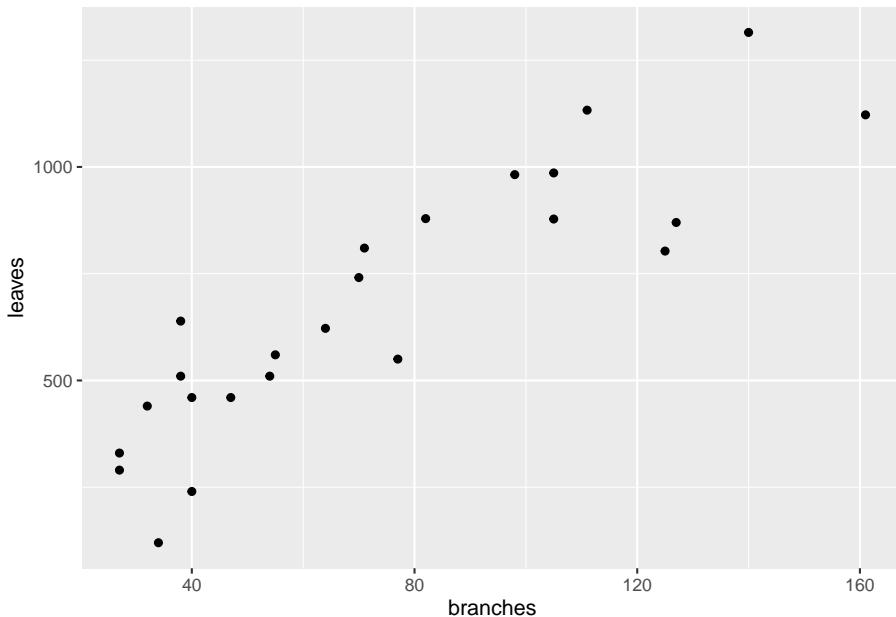
## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()   masks stats::lag()
```

```
library(ggplot2)
cucumber = read.csv("data-unit-10/cucumber_traits.csv")
head(cucumber)

##   cycle rep plants flowers branches leaves totalfruit culledfruit earlyfruit
## 1     1   1      29      22      40     240         1          0          0
## 2     1   2      21      17      34     120        17          2          1
## 3     1   3      31      52      32     440        29         15          3
## 4     1   4      30      49      77     550        25          9          5
## 5     1   5      28      88     140    1315        58         13         27
## 6     1   6      28     162     105     986        39          9         14
```

What is the relationship between branches and leaves? Let's plot the data.

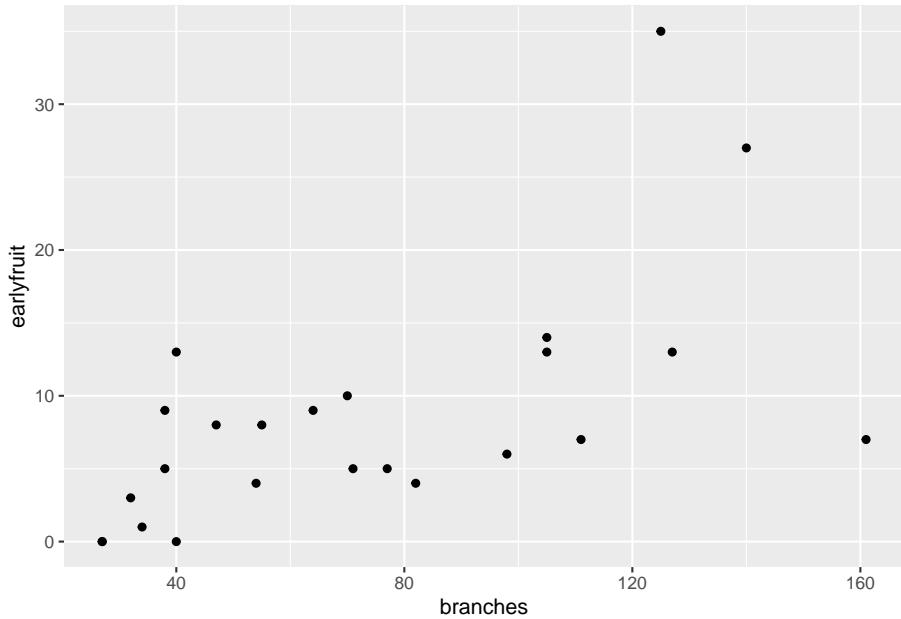
```
ggplot(cucumber, aes(x=branches, y = leaves)) +
  geom_point()
```



We don't know whether leaves cause more branches. You might argue that more branches provide more places for leaves to form. But you could equally argue that the number of leaves affects the production of photosynthates for additional branching.

Similarly, what is the relationship between earlyfruit and the number of branches?

```
ggplot(cucumber, aes(x=branches, y=earlyfruit)) +
  geom_point()
```



In both cases, we can see that as one variable increases, so does the other. But we don't know whether the increase in one causes the increase in the other, or whether there is another variable (measured or unmeasured, that causes both to increase).<sup>4</sup>

## 10.2 Correlation

Correlation doesn't measure *causation*. Instead, it measures *association*. The first way to identify correlations is to plot variables as we have just done. But, of course, it is hard for us to measure the strength of the association just by eye. In addition, it is good to have a way of directly measuring the strength of the correlation. Our measure in this case is the *correlation coefficient*,  $r$ .  $r$  varies between  $-1$  and  $1$ . Values near  $0$  indicate little or no association between  $Y$  and  $X$ . Values close to  $1$  indicate a strong, positive relationship between  $Y$  and  $X$ . A positive relationship means that as  $X$  increases, so does  $Y$ . Conversely, values close to  $-1$  indicate a strong, negative relationship between  $Y$  and  $X$ . A negative relationship means that as  $X$  increases,  $Y$  decreases.

Experiment with the application found at the following link:

<https://marin-harbur.shinyapps.io/10-correlation/>

What happens as you adjust the value of r using the slider control?

For the cucumber datasets above, the correlations are shown below:

```
cor_branches_leaves = round(cor(cucumber$branches, cucumber$leaves), 2)

plot1 = ggplot(cucumber, aes(x=branches, y = leaves)) +
  geom_point() +
  geom_text(x=140, y=0.9*max(cucumber$leaves), label = paste0("r = ", cor_branches_leaves))

cor_branches_earlyfruit = round(cor(cucumber$branches, cucumber$earlyfruit), 2)
max_earlyfruit = max(cucumber$earlyfruit)

plot2 = ggplot(cucumber, aes(x=branches, y = earlyfruit)) +
  geom_point() +
  geom_text(data=cucumber, x=140, y=0.9*max(cucumber$earlyfruit), label = paste0("r = ", cor_branches_earlyfruit))

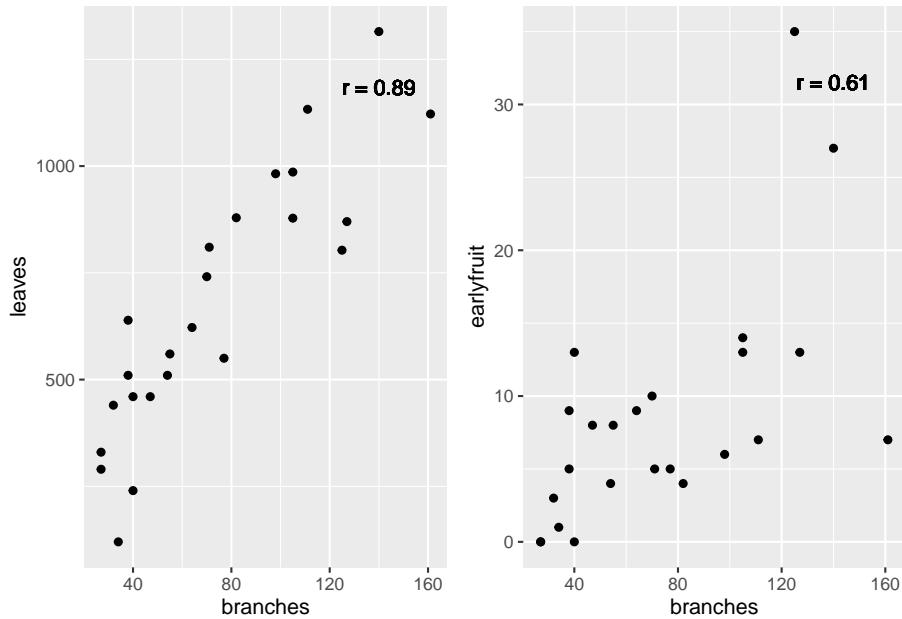
require(gridExtra)

## Loading required package: gridExtra

##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':
##     combine

grid.arrange(plot1, plot2, ncol=2)
```



### 10.2.1 How Is $r$ Calculated (optional reading)

Something that bothered me for years was understanding what  $r$  represented – how did we get from a cloud of data to that number?. The formula is readily available, but how does it work? To find the explanation in plain English is really hard to find, so I hope you enjoy this!

To understand this, let's consider you are in Marshalltown, Iowa, waiting for the next Derecho. You want to go visit your friends, however, who are similarly waiting in Des Moines for whatever doom 2020 will bring there. How will you get there?

First, you could go “as the crow flies” on Routes 330 and 65. This is the shortest distance between Marshalltown and Des Moines. In mathematical terms this is known as the “Euclidian Distance”. Now you probably know the Euclidean Distance by a different name, the one you learned in eighth grade geometry. Yes, it is the hypotenuse, the diagonal on a right triangle!

Second, you might want to stop in Ames to take some barbecue or pizza to your friends in Des Moines. In that case, you would travel a right angle, “horizontally” along US 30 and then “vertically” down I-35. You might call this “going out of your way”. The mathematical term is “Manhattan Distance”. No, not Kansas. The Manhattan distance is named for the grid system of streets in the upper two thirds of Manhattan. The avenues for the vertical axes and the streets form the horizontal axes.

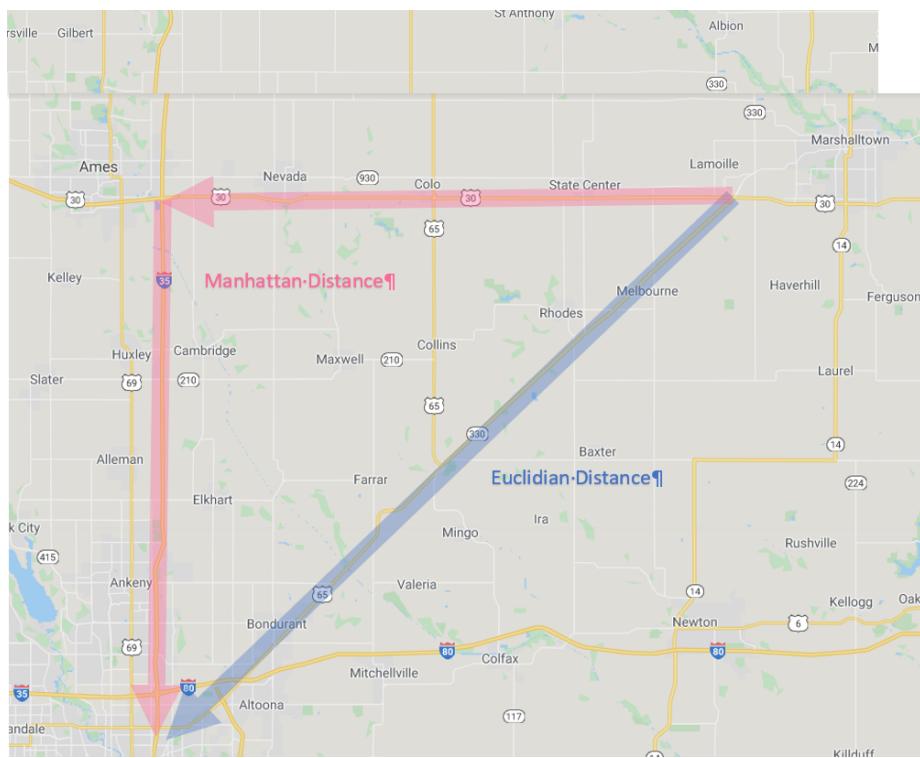


Figure 10.1: Iowa Map

As you might remember, the length of the hypotenuse of a right triangle is calculated as:

$$z^2 = x^2 + y^2$$

Where  $z$  is the distance as the crow flies,  $x$  is the horizontal distance, and  $y$  is the vertical distance. This is the Euclidian Distance. The Manhattan distance, by contrast, is simply  $x + y$ .

Now what if we were simply driving from Marshalltown to Ames? Would the Euclidian distance and the Manhattan distance be so different? No, because both Marshalltown and Ames are roughly on the x-axis? Similarly, what if we drove from Ames to Des Moines? The Euclidian distance and Manhattan distance would again be similar, because we were travelling across the x-axis.

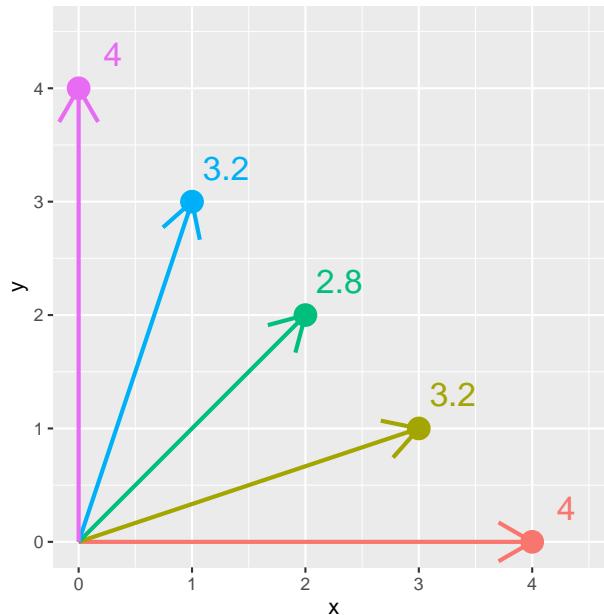
The difference between the Euclidian distance and the Manhattan distance is greatest when we must travel at a 45 degree angle from the X axis. We can demonstrate this with the following plot. Every point below is four units from the origin ( $x = 0, y = 0$ ). Notice that when the point is on the x or y axis, the Euclidian distance and Manhattan distance are equal. But as the angle increases to zero, the Euclidian distance decreases, reaching its lowest value when  $x=y$  and the angle from the axis is 45 degrees.

In the plot below, each point has a Manhattan distance ( $x + y$ ) of 4. The Euclidian distance is shown beside each point. We can see the Euclidian distance is least when  $y = x = 2$ .

```
x = c(4,3,2,1,0)
y = c(0,1,2,3,4)

distance_example = cbind(x,y) %>%
  as.data.frame() %>%
  mutate(euclidian_distance = sqrt(x^2 + y^2),
        manhattan_distance = x+y,
        id = row_number(),
        id = as.factor(id),
        euclidian_distance = round(euclidian_distance, 1))

ggplot(distance_example, aes(x=x, y=y, group=id, label = euclidian_distance)) +
  geom_point(aes(color=id), size=5) +
  geom_segment(aes(x = 0, y = 0, xend = x, yend = y, color=id), arrow = arrow(length =
    5), size=1) +
  geom_text(aes(x = x + 0.3, y= y + 0.3, color = id), size=6) +
  coord_fixed() +
  lims(x=c(0,4.5), y=c(0, 4.5)) +
  theme(legend.position = "none")
```



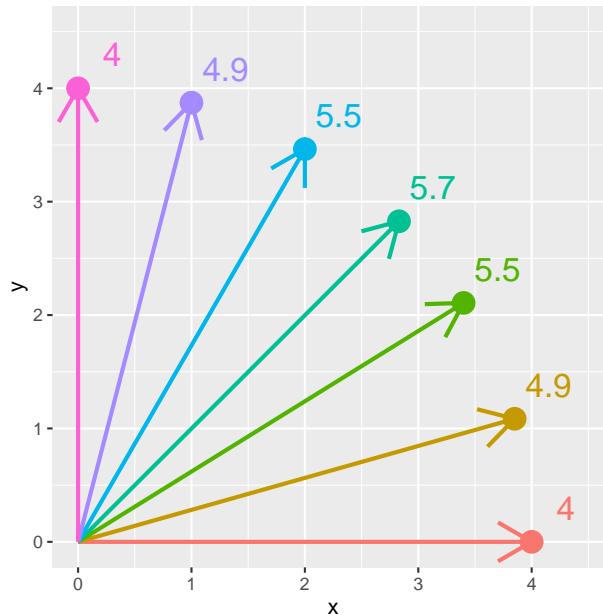
Conversely, in the plot below, each point is the same Euclidean distance (4 units) from the origin ( $x = 0, y = 0$ ). The Manhattan distance is shown beside each point. We can see the Manhattan distance is greatest when the point is at a 45 degree angle from the origin.

```

x = c(4,3.85,3.4, 2.83,2,1,0)
z = 4

distance_example_2 = cbind(x,z) %>%
  as.data.frame() %>%
  mutate(euclidian_distance = z,
        y= sqrt(z^2 - x^2),
        manhattan_distance = x+y,
        id = row_number(),
        id = as.factor(id),
        manhattan_distance = round(manhattan_distance, 1))

ggplot(distance_example_2, aes(x=x, y=y, group=id, label = manhattan_distance)) +
  geom_point(aes(color=id), size=5) +
  geom_segment(aes(x = 0, y = 0, xend = x, yend = y, color=id), arrow = arrow(length = unit(7, "mm")))
  geom_text(aes(x = x + 0.3, y= y + 0.3, color = id), size=6) +
  coord_fixed() +
  lims(x=c(0,4.5), y=c(0, 4.5)) +
  theme(legend.position = "none")
  
```



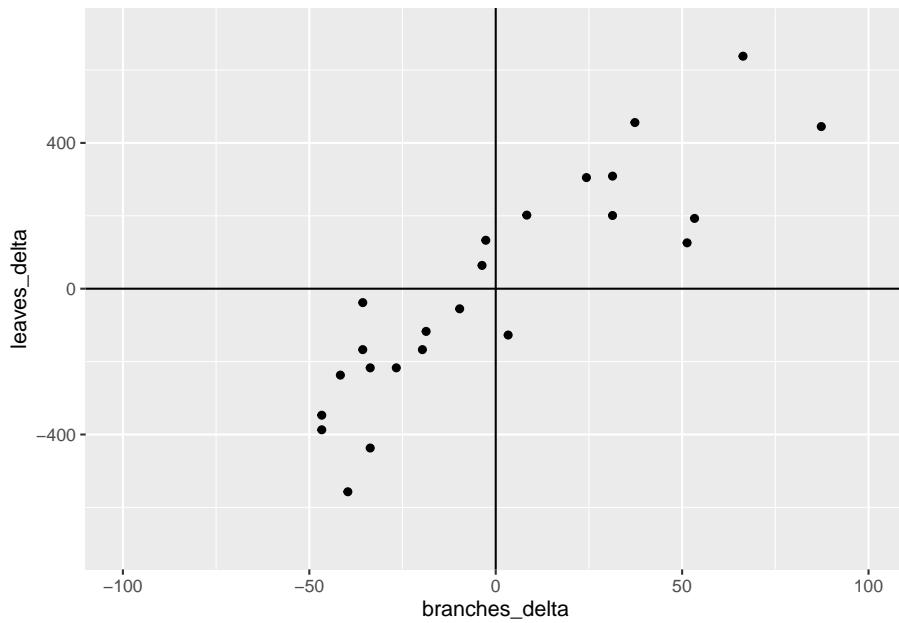
The calculation of the correlation coefficient,  $r$ , depends on this concept of covariance between  $x$  and  $y$ . The covariance is calculated as:

$$S_{xy} = \sum (x_i - \bar{x})(y_i - \bar{y})$$

Let's go back to our cucumber data. We will calculate the difference of each point from the  $\bar{x}$  and  $\bar{y}$ . The points are plotted below.

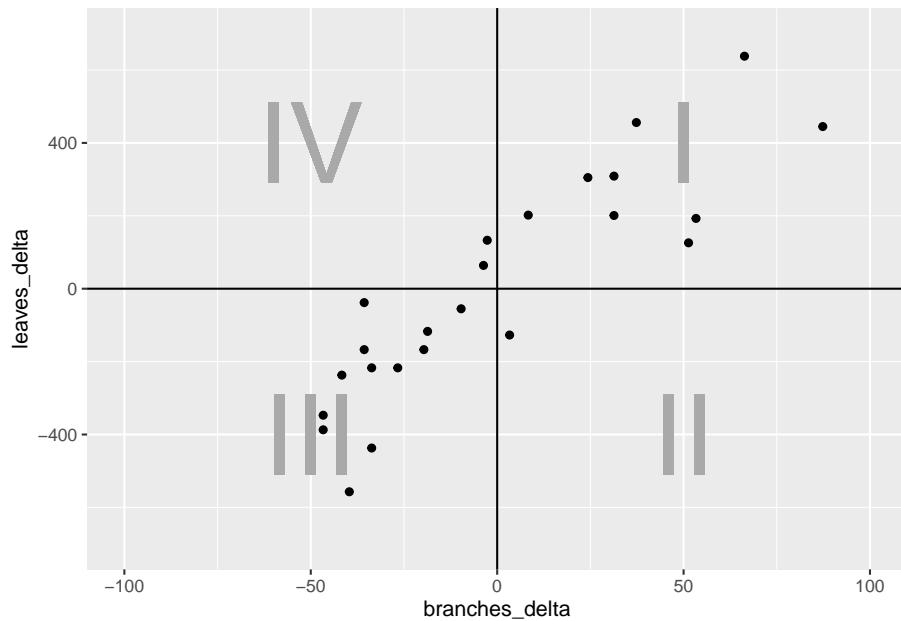
```
cucumber_cov = cucumber %>%
  mutate(branches_delta = branches - mean(branches),
        leaves_delta = leaves - mean(leaves),
        cov = branches_delta * leaves_delta)

ggplot(cucumber_cov, aes(x= branches_delta, y=leaves_delta)) +
  geom_point() +
  geom_vline(xintercept=0) +
  geom_hline(yintercept=0) +
  lims(x=c(-100, 100), y=c(-700, 700))
```



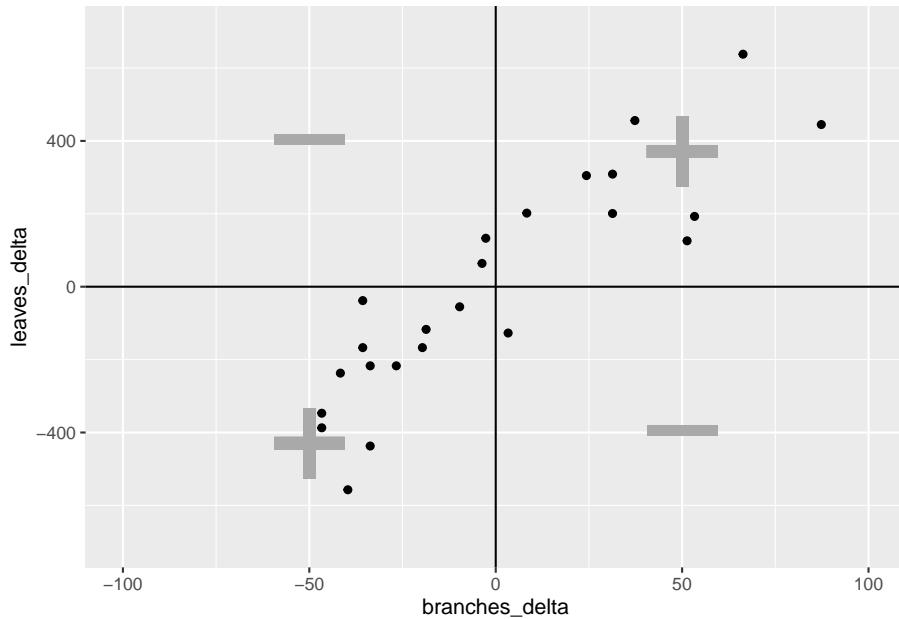
What we have now are four quadrants. The differences between them are important because they affect the *sign* of the covariance. Remember, the covariance of each point is the product of the x-distance and the y-distance of each point.

```
ggplot(cucumber_cov, aes(x= branches_delta, y=leaves_delta)) +
  geom_point() +
  geom_vline(xintercept=0) +
  geom_hline(yintercept=0) +
  geom_text(x=50, y=400, label="I", size = 20, color="darkgrey", ) +
  geom_text(x=50, y=-400, label="II", size = 20, color="darkgrey") +
  geom_text(x=-50, y=-400, label="III", size = 20, color="darkgrey") +
  geom_text(x=-50, y=400, label="IV", size = 20, color="darkgrey") +
  lims(x=c(-100, 100), y=c(-700, 700))
```



In quadrant I, both the x-distance and y-distance are positive, so their product will be positive. In quadrant II, the x-distance is still positive but the y-distance is negative, so their product will be negative. In quadrant III, both x-distance and y-distance are negative, so their negatives will cancel each other and the product will be positive. Finally, quadrant IV, will have a positive x-distance and negative y-distance and have a negative sign.

```
ggplot(cucumber_cov, aes(x= branches_delta, y=leaves_delta)) +
  geom_point() +
  geom_vline(xintercept=0) +
  geom_hline(yintercept=0) +
  geom_text(x=50, y=400, label="+", size = 25, color="darkgrey", ) +
  geom_text(x=50, y=-355, label="-", size = 25, color="darkgrey") +
  geom_text(x=-50, y=-400, label="+", size = 25, color="darkgrey") +
  geom_text(x=-50, y=445, label="-", size = 25, color="darkgrey") +
  lims(x=c(-100, 100), y=c(-700, 700))
```



Enough already! How does all this determine  $r$ ? It's simple – the stronger the association between  $x$  and  $y$ , the more linear the arrangement of the observations in the plot above. The more linear the arrangement, the more the points will be in diagonal quadrants. In the plot above, any observations that fall in quadrant I or III will contribute to the positive value of  $r$ . Any points that fall in quadrants II or IV will subtract from the value of  $r$ .

In that way, a loose distribution of points around all four quadrants, which would indicate  $x$  and  $y$  are weakly associated, would be penalized with an  $r$  score close to zero. A distribution concentrated in quadrants I and III would have a positive  $r$  value closer to 1, indicating a *positive* association between  $x$  and  $y$ . Conversely, a distribution concentrated in quadrants II and IV would have a negative  $r$  value closer to -1, and a *negative* association between  $x$  and  $y$ .

One last detail. Why is  $r$  always between -1 and 1? To understand that, we look at the complete calculation of  $r$ .

$$r = \frac{S_{xy}}{\sqrt{S_{xx}} \cdot \sqrt{S_{yy}}}$$

You don't need to memorize this equation. Here is what it is doing, in plain English.  $S_{xy}$  is the covariance. It tells us, for each point, how its  $x$  and  $y$  value vary together.  $S_{xx}$  is the sum of squares of  $x$ . It sums the distances of each point from the origin ( $x = 0, y = 0$ ) along the  $x$  axis.  $S_{yy}$  is the sum of squares of  $y$ . It is the total  $y$  distance of points from the origin. By multiplying the

square root of  $S_{xx}$  and  $S_{yy}$ , we calculate the maximum theoretical covariance that the points in our measure could have.

$r$  is, after all this, a proportion. It is the measured covariance of the points, divided by the covariance they would have if they fell in a straight line.

I hope you enjoy this explanation. I don't like to go into great detail about calculations, but one of my greatest hangups with statistics is how little effort is often made to explain where the statistics actually come from. If you look to Wikipedia for an explanation, you usually get a formula that assumes you have an advanced degree in calculus. Why does this have to be so hard?

Statistics is the end, about describing the *shape* of the data. How widely are the observations distributed? How far do they have to be from the mean to be too improbable to be the result of chance? Do the points fall in a line or not? There is beauty in these shapes, as well as awe that persons, decades and even millenia before digital computers, discovered how to describe them with formulas.

Then again, it's not like they had *Tiger King* to watch.

### 10.3 Regression

Regression describes a relationship between an independent variable (usually represented by the letter  $y$ ) and one or more dependent variables ( $x_1, x_2$ , etc). Regression differs from correlation in that the model assumes that the value of  $x$  is substantially determined by the value of  $y$ . Instead of describing the *association* between  $y$  and  $x$ , we now refer to causation – how  $X$  determines the value of  $Y$ .

Regression analysis may be conducted simply to test the hypothesis that a change in one variable drives a change in another, for example, that an increase in herbicide rate causes an increase in weed control. Regression, however, can also be used to predict the value of  $y$  based on intermediate values of  $x$ , that is, values of  $x$  between those used to fit or “train” the regression model.

The prediction of values of  $y$  for intermediate values of  $x$  is called *interpolation*. In the word interpolation we see “inter”, meaning between, and “pole”, meaning end. So interpolation is the prediction of values between actually sampled values. If we try to make predictions for  $y$  outside of the range of values of  $x$  in which the model was trained, this is known as *extrapolation*, and should be approached very cautiously.

If you hear a data scientist discuss a predictive model, it is this very concept to which they refer. To be fair, there are many tools besides regression that are used in predictive models. We will discuss those toward the end of this course. But the regression is commonly used and one of the more intuitive predictive tools in data science.

In this lesson, we will learn about one common kind of regression, simple linear regression. This means we will learn how to fit a cloud of data with a straight line that summarizes the relationship between  $Y$  and  $X$ . This assumes, of course, that it is appropriate to use a straight line to model that relationship, and there are ways for us to test that we will learn at the end of this lesson.

### 10.3.1 Case Study

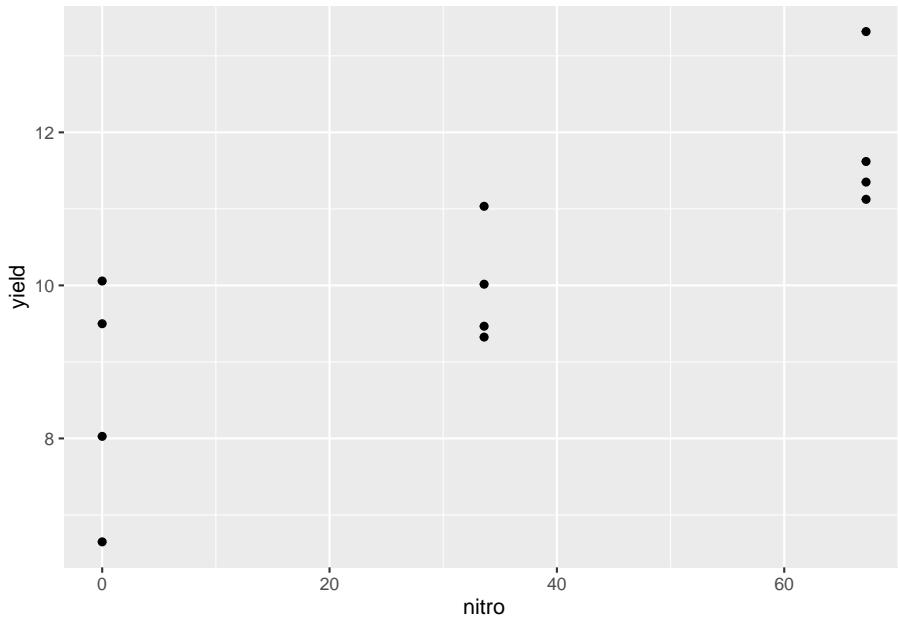
A trial was conducted in Waseca, Minnesota, to model corn response to nitrogen. Yields are per plot, not per acre.

```
nitrogen = read.csv("data-unit-10/nitrogen_waseca.csv")
head(nitrogen)

##   site    loc rep nitro     yield
## 1 S3 Waseca1 R1  0.0  9.49895
## 2 S3 Waseca1 R2  0.0 10.05715
## 3 S3 Waseca1 R3  0.0  8.02693
## 4 S3 Waseca1 R4  0.0  6.64823
## 5 S3 Waseca1 R1 33.6 10.01547
## 6 S3 Waseca1 R2 33.6 11.03366
```

The first thing we should do with any data, but especially data we plan to fit with a linear regression model, is to visually examine the data. Here, we will create a scatterplot with yield on the Y-axis and nitro (the nitrogen rate) on the X-axis.

```
ggplot(data=nitrogen, aes(x=nitro, y=yield)) +
  geom_point()
```



We can see that nitrogen was applied at three rates. It is easy to check these five rates using the `unique` command in R.

```
unique(nitrogen$nitro)
```

```
## [1] 0.0 33.6 67.2
```

We can see that the centers of the distribution appear to fall in a straight line, so we are confident we can model the data with simple linear regression.

### 10.3.2 Linear Equation

Simple linear regression, unlike correlation, fits the data could with an equation. In Unit 5, we revisited middle school, where you learned that a line can be defined by the following equation:

$$y = mx + b$$

Where  $y$  and  $x$  define the coordinate of a point along that line,  $m$  is equal to the slope or “rise” (the change in the value of  $y$  with each unit change in  $x$ , and  $b$  is the  $y$ -intercept (where the line crosses the  $y$ -axis. The  $y$ -intercept can be seen as the “anchor” of the line; the slope describes how the line is pivoted on that anchor.

In statistics we use a slightly different equation – same concept, different annotation

$$\hat{y} = \hat{\alpha} + \hat{\beta}x$$

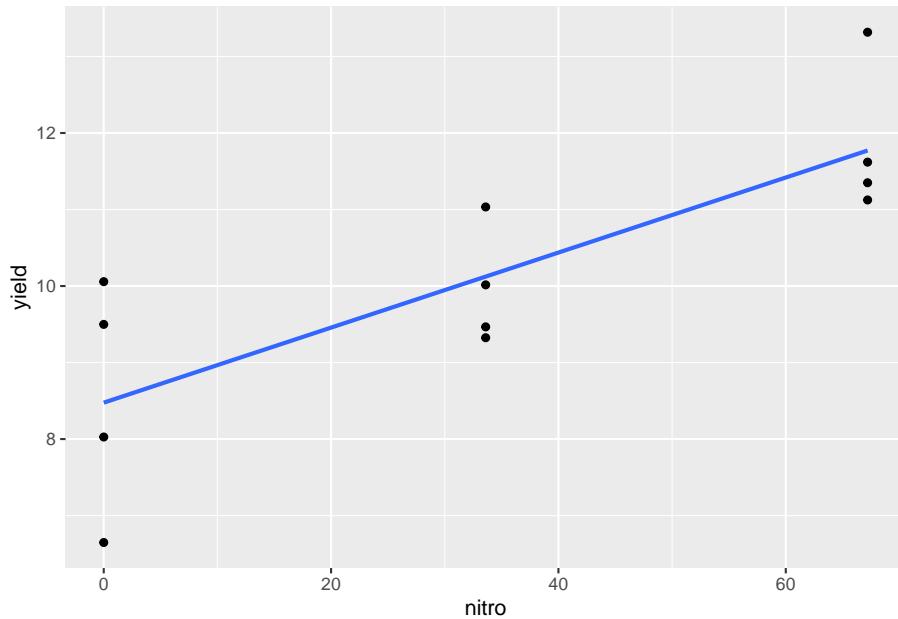
The  $y$  intercept is represented by the greek letter  $\alpha$ , the slope is represented by the letter  $\beta$ .  $y$ ,  $x$ , and  $\alpha$  all have hat-like symbols called carats ( $\hat{\cdot}$ ) above them to signify they are estimates, not known population values. This is because the regression line for the population is being estimated from a sample.  $\hat{y}$  is also an estimate, since it is calculated using the estimated values  $\hat{\alpha}$  and  $\hat{\beta}$ . Only  $x$ , which in experiments is manipulated, is a known value.

### 10.3.3 Calculating the Regression Equation

We can easily add a regression line to our scatter plot.

```
ggplot(data=nitrogen, aes(x=nitro, y=yield)) +
  geom_point() +
  geom_smooth(method=lm, se=FALSE)

## `geom_smooth()` using formula 'y ~ x'
```



The blue line represents the regression model for the relationship between yield and nitro. Of course, it would be nice to see the linear equation as well, which we can estimate using the *lm()* function of R.

```
regression_model = lm(yield~nitro, nitrogen)
regression_model
```

```
##
## Call:
## lm(formula = yield ~ nitro, data = nitrogen)
##
## Coefficients:
## (Intercept)      nitro
##     8.47604    0.04903
```

The coefficients above define our regression model. The number given under “(Intercept)” is the estimate of the y-intercept, or  $\hat{\alpha}$ . The number under nitro is the estimate of the slope, or  $\hat{\beta}$ . Knowing that, we can construct our regression equation:

$$\hat{y} = \hat{\alpha} + \hat{\beta}x$$

$$\hat{y} = 8.476 + 0.04903x$$

This tells us that the yield with 0 units of n is about 8.5, and for each unit of nitrogen yield increases about 0.05 units. If we had 50 units of nitrogen, our yield would be:

$$\hat{y} = 8.476 + 0.04903(50) = 10.9275$$

Since nitrogen was only measured to three significant digits, we will round the predicted value to 10.9.

If we had 15 units of nitrogen, our yield would be:

$$\hat{y} = 8.476 + 0.04903(15) = 9.21145$$

Which rounds to 9.21. So how is the regression line calculated?

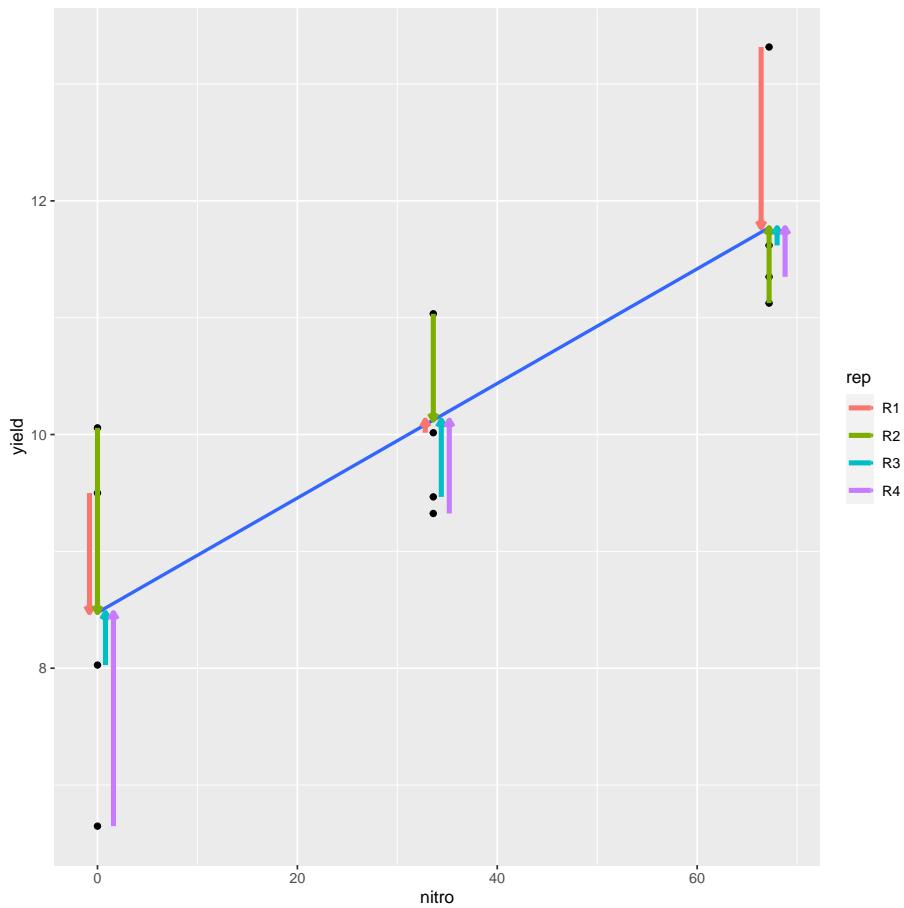
#### 10.3.4 Least-Squares Estimate

The regression line is fit to the data using a method known as the least-squares estimate. To understand this concept we must recognize the goal of a regression equation is to make the most precise estimate for Y as possible. We are not

estimating X, which is already known. Thus, the regression line crosses the data cloud in a way that minimizes the vertical distance (Y-distance) of observations from the regression line. The horizontal distance (X-distance) is not fit by the line.

```
## Sample data

nitrogen$res <- residuals(lm(yield ~ nitro, data=nitrogen))
nitrogen$pred = predict(lm(yield ~ nitro, data=nitrogen))
nitrogen_final = nitrogen %>%
  mutate(rep2 = gsub("R", "", rep)) %>%
  mutate(rep2 = as.numeric(rep2)) %>%
  mutate(xpos = nitro + ((rep2-2)*0.8))
ggplot(data=nitrogen_final, aes(x=nitro, y=yield)) +
  geom_point() +
  geom_smooth(method=lm, se=FALSE) +
  geom_segment(aes(x=xpos, xend=xpos, y=yield, yend=pred, color=rep), size=1.5, arrow = arrow(lem
## `geom_smooth()` using formula 'y ~ x'
```



In the plot above, the distances of each the four points to the regression line are highlighted by arrows. The arrows are staggered (“jittered”, in plot lingo) so you can see them more easily. Note how the line falls closely to the middle of the points at each level of R.

Follow the link below to an app that will allow you to adjust the slope of a regression line and observe the change in the error sum of squares, which measures the sums of the differences between observed values and the value predicted by the regression line.

<https://marin-harbur.shinyapps.io/10-least-squares/>

You should have observed the position of the regression line that minimizes the sum of squares is identical to that fit using the least squares regression technique.

The line is fit using two steps. First the slope is determined, in an approach that is laughably simple – after you understand it.

$$\hat{\beta} = \frac{S_{xy}}{S_{xx}}$$

What is so simple about this? Let's remember what the covariance and sum of squares represents. The covariance is the sum of the manhattan distances of each individual from the "center" of the sample, which is the point located at  $(\bar{x}, \bar{y})$ . For each point, the Manhattan distance is the product of the horizontal distance of an individual from  $\bar{x}$ , multiplied by the sum of the vertical distance of an individual from  $\bar{y}$ .

The sum of squares for  $x$ , ( $S_{xx}$ ) is the sum of the squared distances of each individual from the  $\bar{x}$ .

We can re-write the fraction above as:

$$\hat{\beta} = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})(x_i - \bar{x})}$$

In the equation above, we can cancel out  $(x_i - \bar{x})$  from the numerator and denominator so that we are left with:

$$\hat{\beta} = \frac{\sum (y_i - \bar{y})}{\sum (x_i - \bar{x})}$$

) In other words, the change in  $y$  over the change in  $x$ .

Once we solve for slope ( $\hat{\beta}$ ) we can solve for the  $y$ -intercept ( $\hat{\alpha}$ ). Alpha-hat is equal to the

$$\hat{\alpha} = \hat{y} - \hat{\beta}\bar{x}$$

This equation tells us how much the line descends (or ascends) from the point  $(\bar{x}, \bar{y})$  to where  $x=0$  (in other words, the Y axis).

### 10.3.5 Significance of Coefficients

What else can we learn from our linear model? We can use the *summary()* command in R to get additional information.

```
summary(regression_model)
```

```
##  
## Call:  
## lm(formula = yield ~ nitro, data = nitrogen)
```

```

## 
## Residuals:
##   Min     1Q Median     3Q    Max
## -1.8278 -0.6489 -0.2865  0.9384  1.5811
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 8.47604   0.50016 16.947 1.08e-08 ***
## nitro       0.04903   0.01153  4.252  0.00168 **
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.096 on 10 degrees of freedom
## Multiple R-squared: 0.6439, Adjusted R-squared: 0.6083
## F-statistic: 18.08 on 1 and 10 DF, p-value: 0.001683

```

Of particular interest in this output is the “Coefficients:” section. It shows the coefficient estimates, as we saw before. But it also provides information on the standard error of these estimates and tests whether they are significantly different.

Again, both  $\hat{\beta}$  and the  $\hat{\alpha}$  are estimates. The slope of the least-squares line in the actual population may tilt a more downward or upward than this estimate. Similarly, the line may be higher or lower in the actual population, depending on the actual Y axis. We cannot know the actual slope and y-intercept of the population. But from the sample we can define confidence intervals for both values, and calculate the probability that they differ from hypothetical values by chance.

We forego the discussion how these values are calculated – most computer programs readily provide these – and instead focus on what they represent. The confidence interval for the Y-intercept represents a range of values that is likely (at the level we specify, often 95%) to contain the true Y-intercept for the true regression line through the population.

In some cases, we are interested if the estimated intercept differs from some hypothetical value – in that case we can simply check whether the true population Y-intercept compares to a hypothetical value. If the value falls outside the confidence interval, we conclude the values are significantly different. In other words, there is low probability the true Y-intercept is equal to the hypothetical value.

More often, we are interested in whether the slope is significantly different than zero. This question can be represented by a pair of hypotheses:

$$H_o : \beta = 0$$

$$H_a : \beta \neq 0$$

The null hypothesis,  $H_0$ , is the slope of the true regression line is equal to zero. In other words,  $y$  does not change in a consistent manner with changes in  $x$ . Put more bluntly: there is no significant relationship between  $y$  and  $x$ . The alternative hypothesis,  $H_a$ , is the slope of the true regression line is *not* equal to zero.  $Y$  *does* vary with  $X$  in a consistent way, so there is a significant relationship between  $Y$  and  $X$  in the population.

The significance of the difference of the slope from zero may be tested two ways. First, a t-test will directly test the probability that  $0$ . Second, the significance of the linear model may be tested using an Analysis of Variance, as we learned in Units 5 and 6.

### 10.3.6 Analysis of Variance

Similarly, we can use the `summary.aov()` command in R to generate an analysis of variance of our results.

```
summary.aov(regression_model)

##           Df Sum Sq Mean Sq F value    Pr(>F)
## nitro       1 21.71  21.713  18.08 0.00168 ***
## Residuals   10 12.01   1.201
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This analysis of variance works the same as those you learned previously. The variance described by the relationship between  $Y$  and  $X$  (in this example identified by the “nitro” term) is compared to the random variance among data points. If the model describes substantially more variance than explained by the random location of the data points, the model can be judged to be significant.

For a linear regression model, the degree of freedom associated with the effect of  $x$  on  $y$  is always 1. The concept behind this is that if you know the mean and one of the two endpoints, you can predict the other endpoint. The residuals will have  $n - 1$  degrees of freedom, where  $n$  is the total number of observations.

Notice that the F-value is the square of the calculated t-value for slope in the coefficients table. This is always the case.

### 10.3.7 Measuring Model Fit with R-Square

Let's return to the summary of the regression model.

```
summary(regression_model)

##
## Call:
## lm(formula = yield ~ nitro, data = nitrogen)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -1.8278 -0.6489 -0.2865  0.9384  1.5811
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 8.47604   0.50016 16.947 1.08e-08 ***
## nitro       0.04903   0.01153   4.252  0.00168 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.096 on 10 degrees of freedom
## Multiple R-squared:  0.6439, Adjusted R-squared:  0.6083
## F-statistic: 18.08 on 1 and 10 DF,  p-value: 0.001683
```

At the bottom is another important statistic, *Multiple R-squared*, or  $R^2$ . How well the model fits the data can be measured with the statistic  $R^2$ . Yes, this is the square of the correlation coefficient we learned earlier.  $R^2$  describes the proportion of the total sum of squares described by the regression model: it is calculated by dividing the model sum of squares.

The total sum of squares in the model above is  $21.71 + 12.01 = 33.72$ . We can confirm the  $R^2$  from the model summary by dividing the model sum of squares, 21.71, by this total,  $33.72$ .  $21.71 \div 33.72 = 0.6439$ . This means that 64% of the variation between observed values can be explained by the relationship between yield and nitrogen rate.

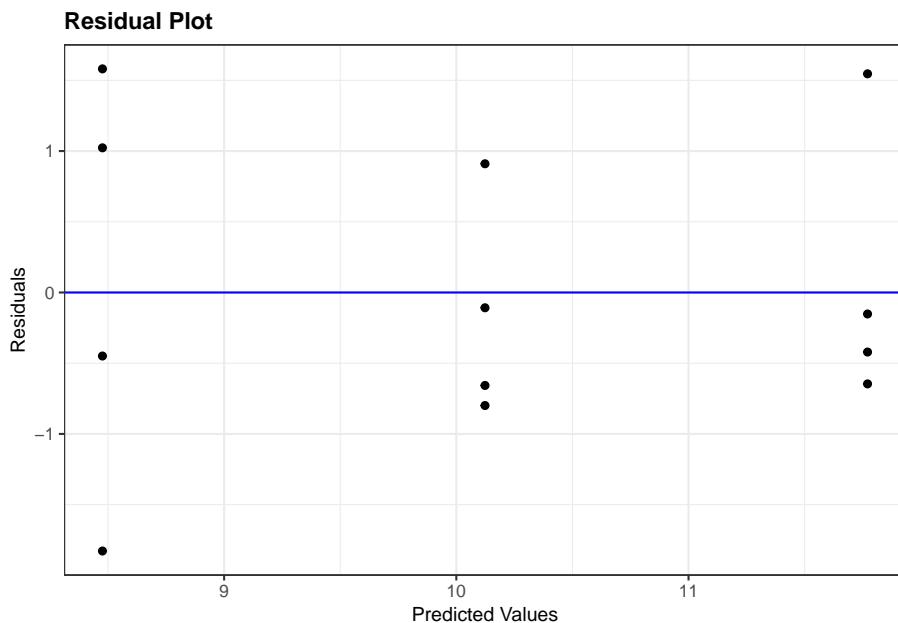
$R^2$  has a minimum possible value of 0 (no relationship at all between y and x) and 1 (perfect linear relationship between y and x). Along with the model coefficients and the analysis of variance, it is the most important measure of model fit.

### 10.3.8 Checking whether the Linear Model is Appropriate

As stated earlier, the simple linear regression model is a predictive model – that is, it is not only useful for establishing a linear relationship between Y and X – it can also be used under the correct circumstances to predict Y given a known value of X. But while a model can be generated in seconds, there are a couple of cautions we must observe.

First, we must be sure it was appropriate to fit our data with a linear model. We can do this by plotting the residuals around the regression line. The *ggResidpanel* package in R allows us to quickly inspect residuals. All we do use run *resid\_panel()* function with two arguments: the name of our model (“regression\_model”) and the plot we want (plots = “resid”).

```
library(ggResidpanel)
resid_panel(regression_model, plots = "resid")
```

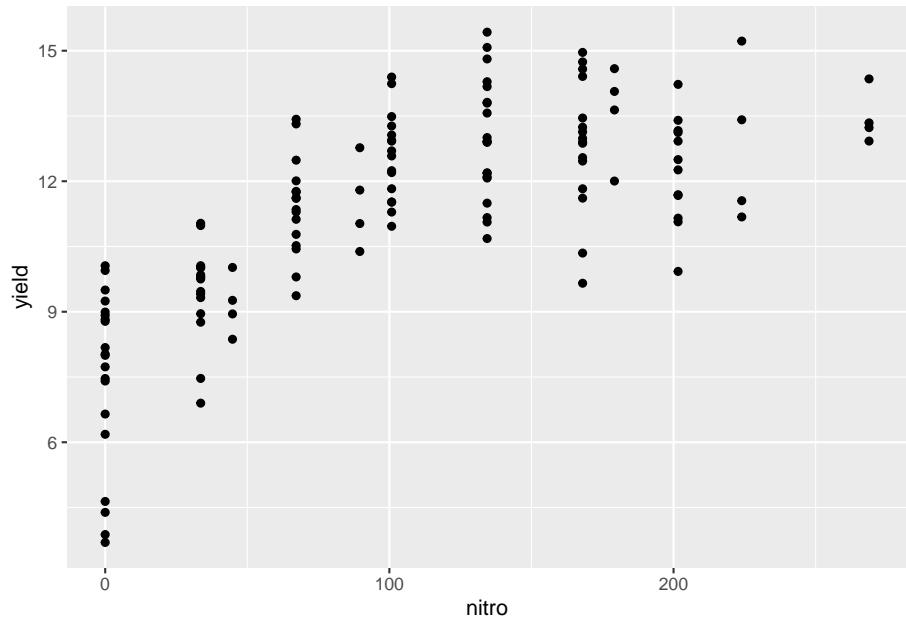


In my mind, the residual plot is roughly equivalent to taking the regression plot above and shifting it so the regression line is horizontal. There are a few more differences, however. The horizontal axis is the y-value predicted by the model for each value of x. The vertical axis is the standardized difference (the actual difference divided by the mean standard deviation across all observations) of each observed value from that predicted for it. The better the regression model fits the observations the closer the points will fall to the blue line.

The key thing we are checking is whether there is any pattern to how the regression line fits the data. Does it tend to overpredict or underpredict the observed values of x? Are the points randomly scattered about the line, or do they seem to form a curve?

In this example, we only modelled a subset of the nitrogen study data. I intentionally left out the higher rates. Why? Let's plot the complete dataset.

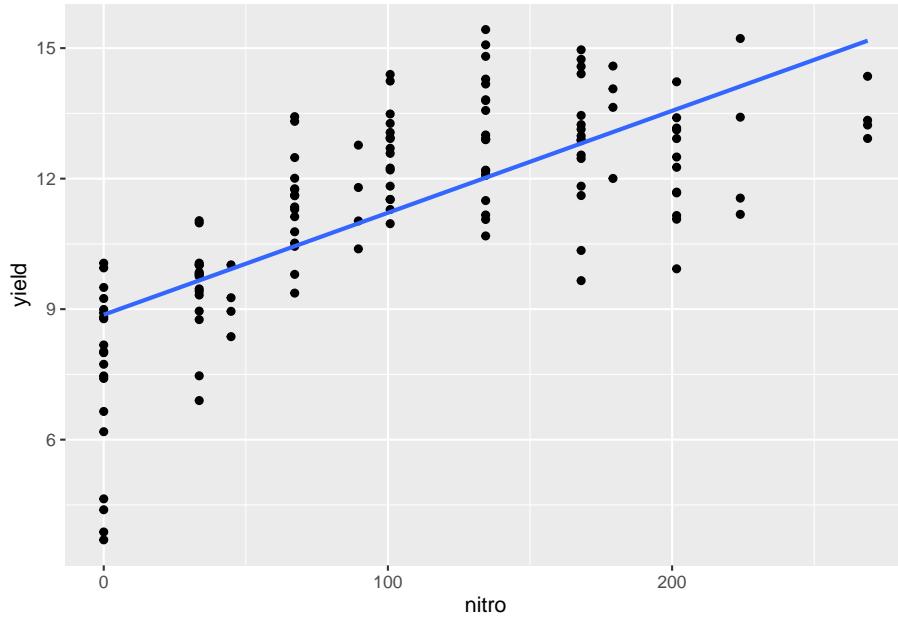
```
library(agridat)
complete_nitrogen = hernandez.nitrogen
ggplot(complete_nitrogen, aes(x=nitro, y=yield)) +
  geom_point()
```



We can see the complete dataset does not follow a linear pattern. What would a regression line, fit to this data, look like?

```
ggplot(complete_nitrogen, aes(x=nitro, y=yield)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



We can see how the regression line appears to overpredict the observed values for at low and high values of nitrogen and underpredict the intermediate values.

What does our regression model look like?

```
bad_model = lm(yield~nitro, data = complete_nitrogen)
summary(bad_model)

##
## Call:
## lm(formula = yield ~ nitro, data = complete_nitrogen)
##
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -5.1752 -0.8803  0.1086  1.1148  3.4049 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 8.872032  0.251752  35.24 <2e-16 ***
## nitro       0.023434  0.001974  11.87 <2e-16 ***
## ---        
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 1.674 on 134 degrees of freedom
```

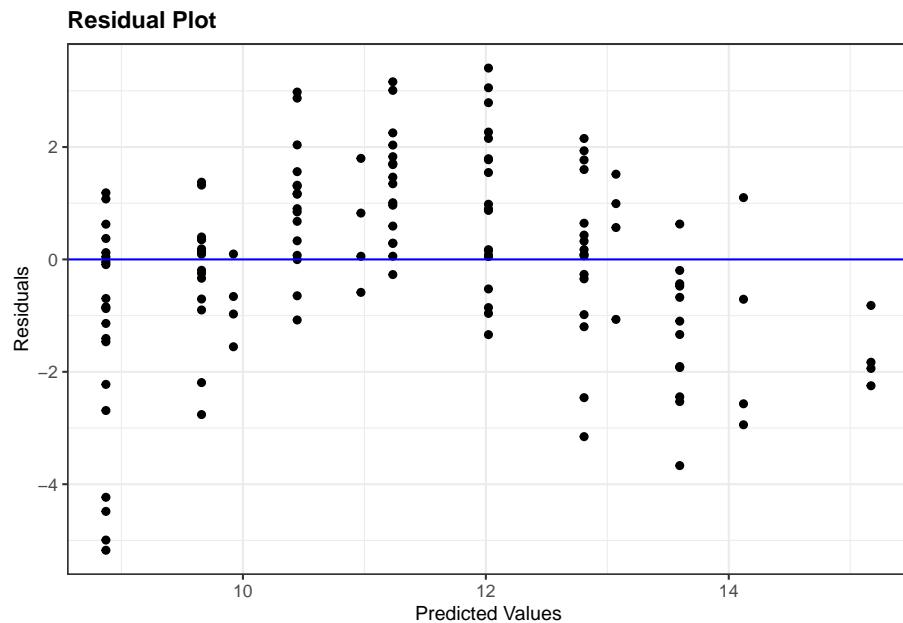
```
## Multiple R-squared:  0.5126, Adjusted R-squared:  0.5089
## F-statistic: 140.9 on 1 and 134 DF,  p-value: < 2.2e-16
```

```
summary.aov(bad_model)
```

```
##          Df Sum Sq Mean Sq F value Pr(>F)
## nitro      1 395.1 395.1 140.9 <2e-16 ***
## Residuals 134 375.7    2.8
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The model is still highly significant, even though it is obvious it doesn't fit the data! Why? Because the simple linear regression model only tests whether the slope is different from zero. Let's look at the residuals:

```
resid_panel(bad_model, plots="resid")
```



As we expect, there is a clear pattern to the data. It curves over the regression line and back down again. If we want to model the complete nitrogen response curve, we will need to use a nonlinear model, which we will learn in the next unit.

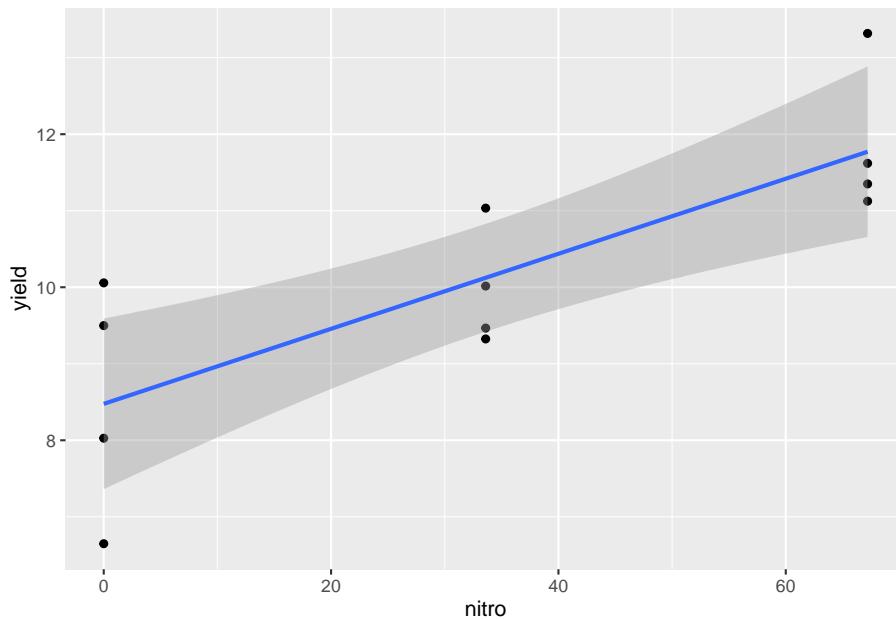
## 10.4 Extrapolation

The above example also illustrates why we should not extrapolate: because we do not know how the relationship between  $x$  and  $y$  may change. In addition, the accuracy of the regression model decreases as one moves away from the middle of the regression line.

Given the uncertainty of the estimated intercept, the entire true regression line may be higher or lower – i.e. every point on the line might be one unit higher or lower than estimated by our estimated regression model. There is also uncertainty in our estimate of slope – the true regression line may have greater or less slope than our estimate. When we combine the two sources of uncertainty, we end up with a plot like this:

```
ggplot(regression_model, aes(x=nitro, y=yield)) +
  geom_point() +
  geom_smooth(method = "lm", se=TRUE)

## `geom_smooth()` using formula 'y ~ x'
```



The dark grey area around the line represents the standard error of the prediction. The least error in our estimated regression line – and the error in any prediction made from it occurs closer at  $\bar{x}$ . As the distance from  $\bar{x}$  increases, so does the uncertainty of the Y-value predicted from it. At first, that increase

in uncertainty is small, but it increases rapidly as we approach the outer data points fit with the model.

We have greater certainty in our predictions when we predict  $y$  for values of  $x$  between the least and greatest  $x$  values used in fitting the model. This method of prediction is called interpolation – we are estimating  $Y$  for  $X$  values within the range of values used to estimate the model.

Estimating  $Y$  for  $X$  values outside the range of values used to estimate the model is called extrapolation, and should be avoided. Not only is our current model less reliable outside the data range used to fit the model – we should not even assume that the relationship between  $Y$  and  $X$  is linear outside the of the range of data we have analyzed. For example, the middle a typical growth curve (often called “sigmoidal”, from sigma, the Greek word for “S”) is linear, but each end curves sharply.

When we make predictions outside of the range of  $x$  values used to fit our model, this is extrapolation. We can now see why it should be avoided.

## 10.5 Exercise: Scatterplots and Regression

This week’s lesson is a marked shift from previous units. Until now, we have worked with independent variables (our  $X$  variables) that were categorical: they had names that represented their value. This week, we begin working with both continuous  $X$  and  $Y$  variables. As we have learned in this lecture, the simplest way to establish a relationship between variables is to examine scatterplots and calculate the correlation coefficients.

### 10.5.1 Case Study: Corn Data

*Allometry* is the study of the different parts of an organism and how their sizes are related. In the word allometry, we can see the root of the word “allocate”. Thus, allometry seeks to understand how the allocation of biomass to one part of an organism may be related to the allocation of biomass to another part. In agronomy, this can provide valuable insight into the relative importance of bigger leaves, taller plants, or larger stalks to grain yield.

Let’s begin by loading our data. This dataset is a simulated based on statistics published by Fred Warren and Robert Fenley in 1970.

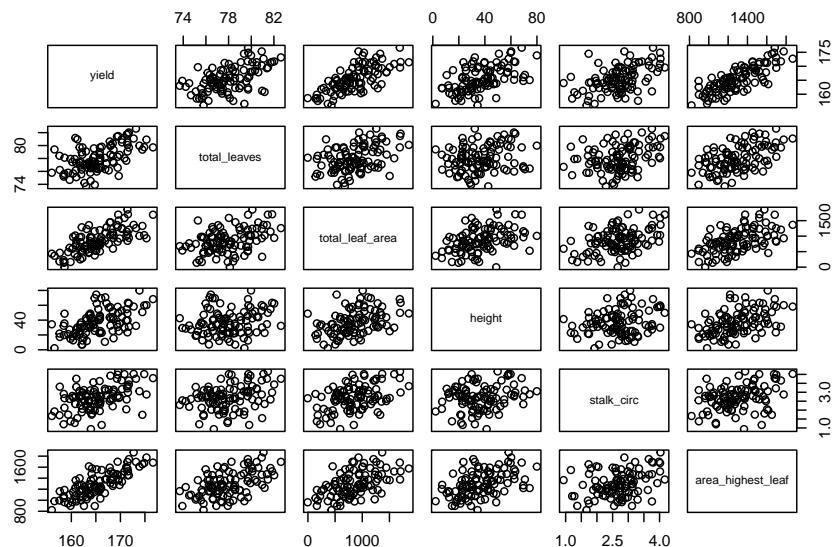
```
library(tidyverse)
allometry = read.csv("data-unit-10/exercise_data/corn_allometry.csv")

head(allometry)
```

```
##   yield total_leaves total_leaf_area   height stalk_circ area_highest_leaf
## 1 166.0838     76.58265      964.9109 69.57771    2.750763      1219.800
## 2 162.2875     79.71361      533.3816 12.48882    2.984688      1051.207
## 3 169.4557     76.56896     1294.9548 30.72335    2.793448      1454.084
## 4 167.9799     78.60217     1448.9345 32.65071    2.631524      1371.558
## 5 173.1781     82.62165     1258.3317 32.27236    3.755847      1696.366
## 6 168.4464     77.50940     1110.5993 25.46963    2.917688      1259.125
```

The first thing we want to do is to examine our data with a scatterplot. The quickest way to do this is with the `plot()` command. This will give us a matrix (a grid) with all possible scatterplots, based on our dataset.

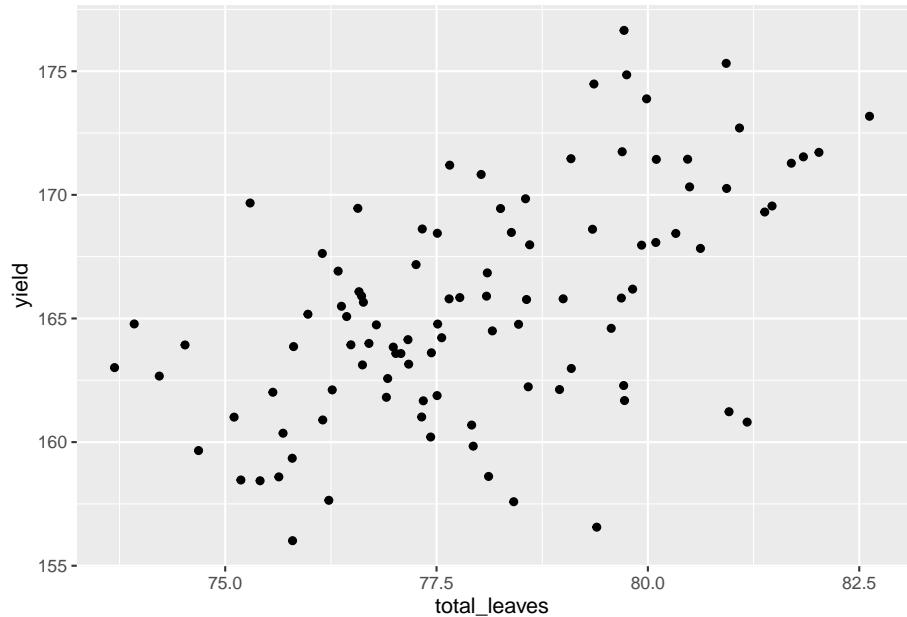
```
plot(allometry)
```



This is called a *scatterplot matrix*. The variable listed in each column defines the position of the points on the X axis for every plot in that column. The variable listed in each row defines the position of the points on the Y axis for every plot in that row. So if we look at the plot directly to the right of yield second plot from left in top column column, its Y-axis is yield and its X axis is total leaves (total leaf biomass).

Of course, this matrix is crowded. If we want to highlight a single scatterplot, we can use `ggplot`.

```
allometry %>%
  ggplot(aes(x=total_leaves, y=yield)) +
  geom_point()                                     # the data frame to feed to ggplot
                                                # tells R where to locate each point
                                                # tells ggplot to plot points instead of
```



What can we see in this plot? Yield and total leaves seem to be correlated. As total\_leaves increased, so did yield.

We can quantify the correlation between yield and total leaves using the `cor.test()` function and the two variables.

```
cor.test(allometry$yield, allometry$total_leaves)
```

```
## 
## Pearson's product-moment correlation
##
## data: allometry$yield and allometry$total_leaves
## t = 6.5193, df = 98, p-value = 3.083e-09
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.3964056 0.6736443
## sample estimates:
## cor
## 0.55
```

The output looks very similar to the t-test we learned earlier. It lists the measured value for t. It includes the probability that t is zero, which is also the probability that the correlation we measured is equal to zero. It also includes, at the bottom, the correlation coefficient (cor), which in this case is 0.55.

What if we have a lot of variables? As we did with the scatter plot matrix, we can build a correlation matrix that shows all the correlations among variables. Again, the code is very simple.

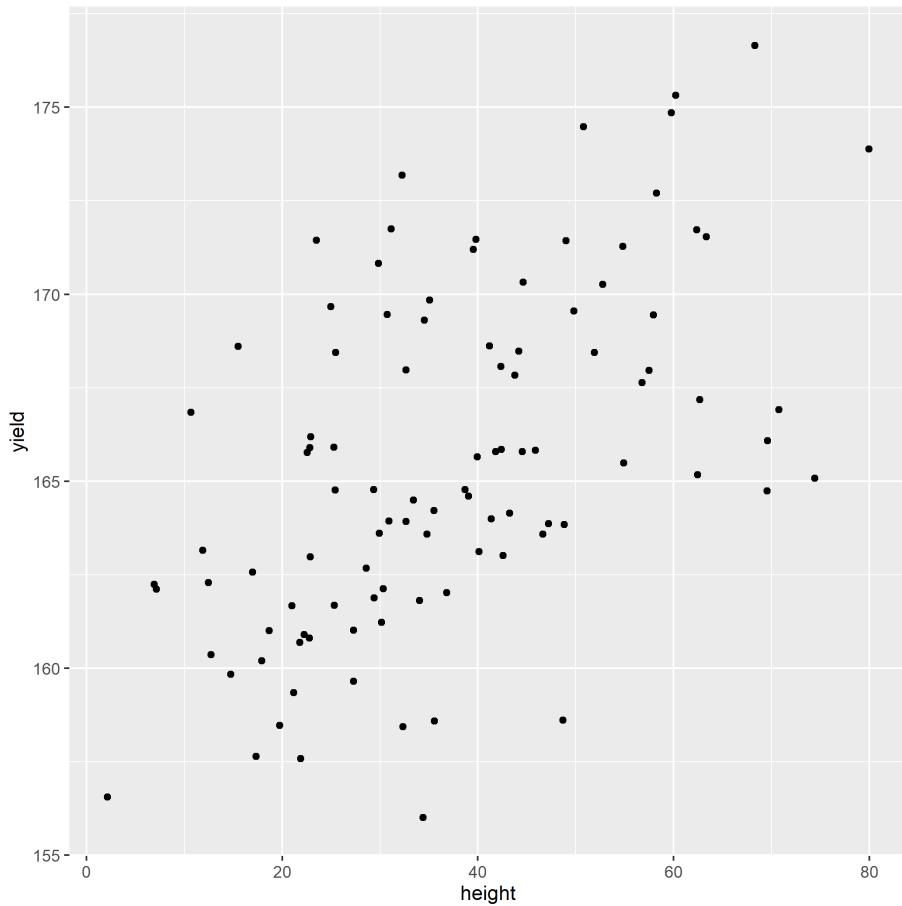
```
cor(allometry)
```

```
##           yield total_leaves total_leaf_area    height stalk_circ
## yield      1.00    0.5500000    0.7100000 0.5500000  0.4900000
## total_leaves 0.55    1.0000000    0.4033716 0.2032565  0.3146973
## total_leaf_area 0.71    0.4033716    1.0000000 0.4174365  0.4297034
## height       0.55    0.2032565    0.4174365 1.0000000  0.2302604
## stalk_circ   0.49    0.3146973    0.4297034 0.2302604  1.0000000
## area_highest_leaf 0.80    0.5332687    0.5409881 0.4691421  0.3397559
##               area_highest_leaf
## yield          0.8000000
## total_leaves   0.5332687
## total_leaf_area 0.5409881
## height          0.4691421
## stalk_circ     0.3397559
## area_highest_leaf 1.0000000
```

If we look along the row corresponding to yield, we see six columns whose names include yield, total\_leaves, total\_leaf\_area, etc. Wherever the yield row intersects these columns, we can read the correlation coefficient. For example, where yield intersects yield, the correlation coefficient is 1.0. This makes sense—yield should be perfectly correlated with itself. If we look at where the yield row intersects the stalk\_circ column, we see a regression coefficient of 0.49.

### 10.5.2 Practice 1

Using the above dataset, calculate a scatter plot for height and corn yield, with corn\_height on the X-axis and yield on the Y-axis. Your plot should look like.



### 10.5.3 Practice 2

Using the above dataset, calculate the correlation between height and yield using the `cor.test()` function. Your estimate for the correlation coefficient (the number at the bottom of the output) should be 0.49.

```
cor.test(allometry$stalk_circ, allometry$yield)
```

```
##  
## Pearson's product-moment correlation  
##  
## data: allometry$stalk_circ and allometry$yield  
## t = 5.5646, df = 98, p-value = 2.287e-07  
## alternative hypothesis: true correlation is not equal to 0
```

```
## 95 percent confidence interval:
## 0.3248467 0.6261540
## sample estimates:
## cor
## 0.49
```

#### 10.5.4 Practice 3

Load the dataset “`data/corn_fertility.csv`”. These data evaluate the response of yield to corn fertility characteristics.

```
fertility = read.csv("data-unit-10/exercise_data/corn_fertility.csv")
head(fertility)
```

	yield	organic_matter	magnesium	potash	phosphorus	pH
## 1	166.0838	2.007710	101.87145	7.214161	105.57985	6.287836
## 2	162.2875	2.001714	122.71640	163.097239	42.50915	7.304132
## 3	169.4557	1.544267	62.12231	26.855054	193.89253	7.494299
## 4	167.9799	1.795988	65.36950	31.239609	84.29568	7.221934
## 5	173.1781	1.988002	227.62357	-30.477244	158.02454	6.573853
## 6	168.4464	2.084489	91.22296	62.028413	100.50722	7.108826

Create a scatterplot matrix for all variables in the `fertility` dataset.

#### 10.5.5 Practice 4

Create a correlation matrix for all the variables in the `fertility` dataset.

## 10.6 Exercise: Simple Linear Regression

In the previous exercise, we learned about scatter plots and correlation. Correlation is used to measure the association between variables, with no assumptions about the direction of causation between variables. We often use correlation with exploratory work, whereas we use regression when our understanding of a topic has grown to incorporate basic knowledge of a science. For example, we know that the rate of photosynthesis of a plant is generally a response to the intensity of photosynthetic radiation (that is, light), instead of the other way around. When we use regression, we fit a model to the data. We then test whether that model explains a significant amount of the variation among our samples, and use it as a linear equation to quantify the relationship between Y and X.

### 10.6.1 Case Study: Corn Allometry

Using our data from the scatterplots and correlation exercise, we will construct simple linear regression models, evaluate their significance, and write the linear model from its coefficients.

```
library(tidyverse)
allometry = read.csv("data-unit-10/exercise_data/corn_allometry.csv")
head(allometry)
```

	yield	total_leaves	total_leaf_area	height	stalk_circ	area_highest_leaf
## 1	166.0838	76.58265	964.9109	69.57771	2.750763	1219.800
## 2	162.2875	79.71361	533.3816	12.48882	2.984688	1051.207
## 3	169.4557	76.56896	1294.9548	30.72335	2.793448	1454.084
## 4	167.9799	78.60217	1448.9345	32.65071	2.631524	1371.558
## 5	173.1781	82.62165	1258.3317	32.27236	3.755847	1696.366
## 6	168.4464	77.50940	1110.5993	25.46963	2.917688	1259.125

### 10.6.2 Fitting the regression model

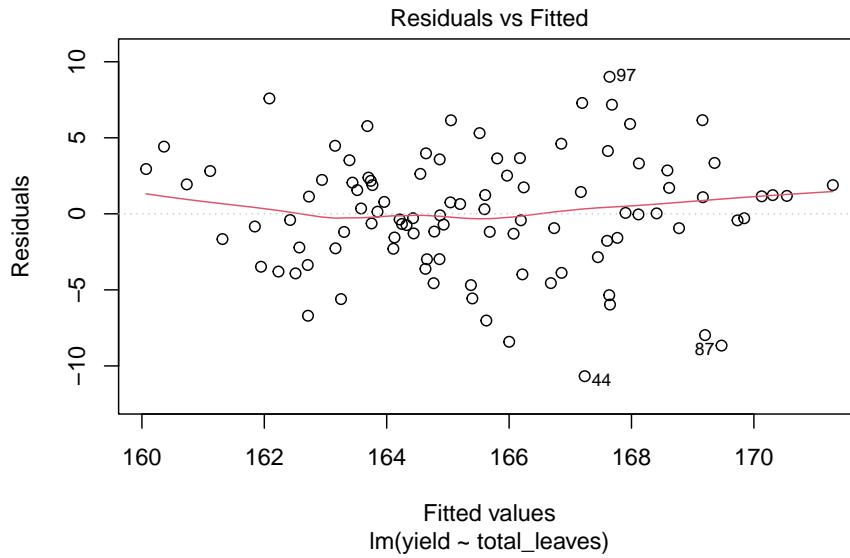
The regression model is fit using a very similar approach to that used for the analysis of variance. In fact, the only thing that is different is that we use `lm()` to define the linear model, instead of `aov()`. We will model yield as a function of total\_leaves (total leaf dry matter).

```
model_yield_total_leaves = lm(yield ~ total_leaves, data=allometry)
```

### 10.6.3 Examining the residuals

An important step that is easy to forget (in fact, I almost did just now) is to plot the residuals. The residuals are the differences between the values predicted by the regression model and the observed values. We observe the residuals to make sure they are randomly distributed around the regression line, represented by the dotted line at Y=0 below. We can use the `plot()` function with our model to get the residual plot.

```
plot(model_yield_total_leaves, which=1) # the "which=1" argument tells R to plot the
```



The red line is not a linear regression line, but a LOESS (locally estimated scatterplot smoothing) – a nonlinear model that can bend to fit the data more closely. If a linear model is appropriate to fit the relationship between Y and X, the red line will be close to (but rarely exactly) linear. The above plot suggests linear regression is appropriate. What we want to beware of is a red line that rises way above zero on either end and dips way below zero in the middle, or vice versa. Those scenarios suggest the relationship between Y and X may be curved instead of linear.

#### 10.6.4 Viewing the Model Coefficients

A lot of information is included in the linear model output. Rather than wade through it in one big chunk, we will use a package called *broom* to break it into more digestible pieces. We can use the *tidy()* format to summarise the coefficients.

```
library(broom)

## Warning: package 'broom' was built under R version 4.1.3

tidy(model_yield_total_leaves)

## # A tibble: 2 x 5
```

```

##   term      estimate std.error statistic    p.value
##   <chr>     <dbl>     <dbl>     <dbl>     <dbl>
## 1 (Intercept) 67.5      15.0      4.49  0.0000197
## 2 total_leaves  1.26      0.193     6.52  0.000000000308

```

Remember, our regression line is defined as:

$Y = \alpha + \Beta * x$ , where  $\alpha$  is the y-intercept and  $\Beta$  is the slope. In the output, the (Intercept) row provides information on  $\alpha$ . The estimate column contains its estimated value, 67.47. R automatically tests whether the estimated value of the intercept is significantly different from zero. That measure is given in the p.value column. We see that  $\alpha$  is significantly different from zero. Much of the time, this insight isn't particularly interesting – there are many relationships between  $Y$  and  $X$  where the y-intercept is not zero. For example, if we studied the relationship between yield and nitrogen, we would likely observe a yield above zero, even if our nitrogen rate was zero.

The second row provides information on  $\Beta$ . Its estimated value is 1.26. We see from its p.value that  $\Beta$  is also significantly different from zero. Unlike  $\alpha$ , this p.value is very interesting to us. If our p.value is significantly different from zero, then we can reject the hypothesis that there is no relationship between yield and total\_leaves. Our model explains a significant amount of the observed variation among our samples.

### 10.6.5 More measures of model fit

The `anova()` function will generate an ANOVA table for our results. We interpret this model the same as we have done previously.

```

anova(model_yield_total_leaves)

## Analysis of Variance Table
##
## Response: yield
##              Df  Sum Sq Mean Sq F value    Pr(>F)
## total_leaves  1  625.28  625.28  42.502 3.083e-09 ***
## Residuals    98 1441.75   14.71
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

The F-value is the ratio of the variance explained by our model to the variance explained by random variation among individuals. In this case, total\_leaves is our independent variable. Since our model is based on a regression line, it only has one degree of freedom. This is because if we know the slope and any one point on that line, we can predict its y-value of any point as long as we know

its x value. Don't get hung up on this – just know that the linear model is supposed to have 1 degree of freedom.

Our F-value is large and the probability of encountering that an F-value that large by chance is given by  $\text{Pr}(>F)$ . That value is very low, so we conclude our model explains a significant amount of the observed variation. You may also note that the p.value for the slope of the regression model and the  $\text{Pr}(>F)$  above are identical – 3.083-09. This is because the significance of the model is based entirely on the significance of the slope.

One other important statistic can be gotten from the `glance()` function. We are interested in the statistic all the way to the left: r.squared.

```
glance(model_yield_total_leaves)
```

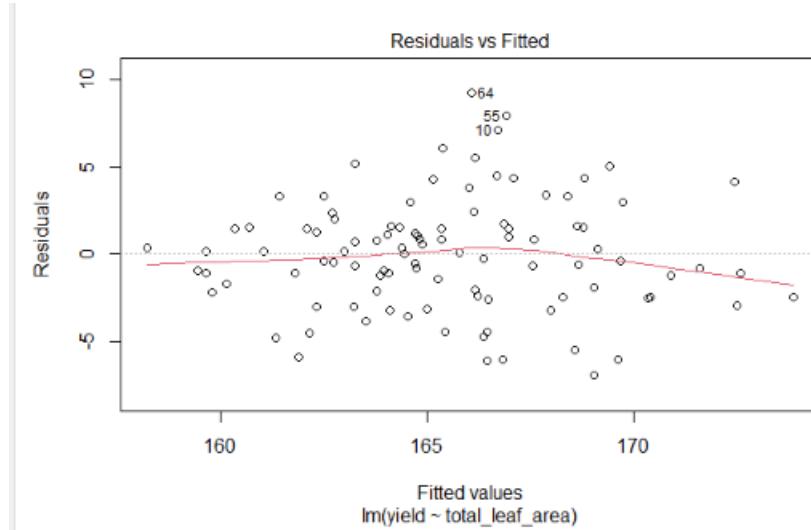
```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic      p.value      df logLik     AIC     BIC
##       <dbl>         <dbl> <dbl>      <dbl>        <dbl>     <dbl> <dbl> <dbl> <dbl>
## 1     0.302         0.295  3.84     42.5 0.00000000308     1   -275.  557.  564.
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

`r.squared` is the percentage of the total variation explained by the regression model. In this case, 30.25% of the variation is explained by our regression model. `r.squared` can be as great as 1, in which case the observed values and predicted values are identical, and as little as zero, in which case there is no relationship between the dependent variable (Y) and the independent variable (X). Our `r.squared` value of 0.3025 does not suggest a strong relationship between yield and total number of leaves. That said, biological systems are very complicated – studying them outdoors makes them more so. Given this complexity, an `r.squared` of 0.30 is worth attention.

### 10.6.6 Practice 1

Lets now model yield as a function of the `total_leaf_area`. First, lets fit the regression model.

Next, plot the residuals. Your results should look like:



Next, let's view the model coefficients

What do you notice about the relationship between yield and total leaf area?  
Is it significant? Does yield increase or decrease with total leaf area?

Using the coefficients above, write the regression model.

Use the `glance()` function to develop additional model fit statistics.

You should see an `r.squared` of 0.50.

### 10.6.7 Practice 2

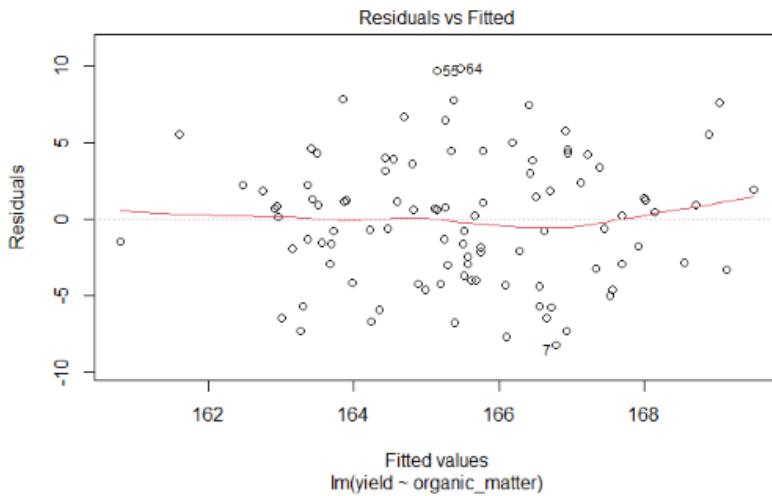
Now lets work with our corn fertility data.

```
fertility = read.csv("data-unit-10/exercise_data/corn_fertility.csv")
head(fertility)
```

```
##      yield organic_matter magnesium      potash phosphorus      pH
## 1 166.0838     2.007710 101.87145    7.214161 105.57985 6.287836
## 2 162.2875     2.001714 122.71640   163.097239   42.50915 7.304132
## 3 169.4557     1.544267  62.12231   26.855054 193.89253 7.494299
## 4 167.9799     1.795988  65.36950   31.239609   84.29568 7.221934
## 5 173.1781     1.988002 227.62357  -30.477244  158.02454 6.573853
## 6 168.4464     2.084489  91.22296   62.028413 100.50722 7.108826
```

Create a simple linear regression model for yield as a function of organic matter.

Next, plot the residuals. Your results should look like:



What do you notice about the residuals? Is there strong indication that a linear model might be inappropriate?

Next, let's view the model coefficients

```
library(broom)
```

What do you notice about the relationship between yield and organic matter? Is it significant? Does yield increase or decrease with organic matter?

Using the coefficients above, write the regression model.

Use the `glance()` function to develop additional model fit statistics.

You should see an `r.squared` of 0.15.

## 10.7 Exercise: Making Predictions from Regression Models.

Now that we have created our linear models and ascertained that they explain a significant amount of the variance in our data, we want to use them to visualize the relationship between our Y and X variables. Furthermore, we may want to use them to make predictions for specific values of X.

### 10.7.1 Case Study

We will continue to work with our corn allometry dataset.

```
library(tidyverse)
allometry = read.csv("data-unit-10/exercise_data/corn_allometry.csv")
head(allometry)

##      yield total_leaves total_leaf_area    height stalk_circ area_highest_leaf
## 1 166.0838     76.58265      964.9109 69.57771   2.750763      1219.800
## 2 162.2875     79.71361      533.3816 12.48882   2.984688      1051.207
## 3 169.4557     76.56896      1294.9548 30.72335   2.793448      1454.084
## 4 167.9799     78.60217      1448.9345 32.65071   2.631524      1371.558
## 5 173.1781     82.62165      1258.3317 32.27236   3.755847      1696.366
## 6 168.4464     77.50940      1110.5993 25.46963   2.917688      1259.125
```

### 10.7.2 Creating and Testing our Model

What effect does stalk circumference have on yield? Let's find out.

```
model_circumference = lm(yield ~ stalk_circ, data=allometry)
```

We will again us tools from the *broom()* package to present clear results. First, we examine our coefficients.

```
library(broom)
tidy(model_circumference)
```

```
## # A tibble: 2 x 5
##   term       estimate std.error statistic p.value
##   <chr>     <dbl>     <dbl>     <dbl>     <dbl>
## 1 (Intercept) 157.      1.60      97.9  1.52e-99
## 2 stalk_circ    3.20     0.575     5.56  2.29e- 7
```

We see the slope of our model is significantly different than zero. There is a significant relationship between stalk circumference and yield. How strong is this relationship?

```
glance(model_circumference)
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic  p.value    df logLik   AIC   BIC
##   <dbl>        <dbl> <dbl>     <dbl>     <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 0.240        0.232  4.00     31.0  0.000000229     1 -280.  565.  573.
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

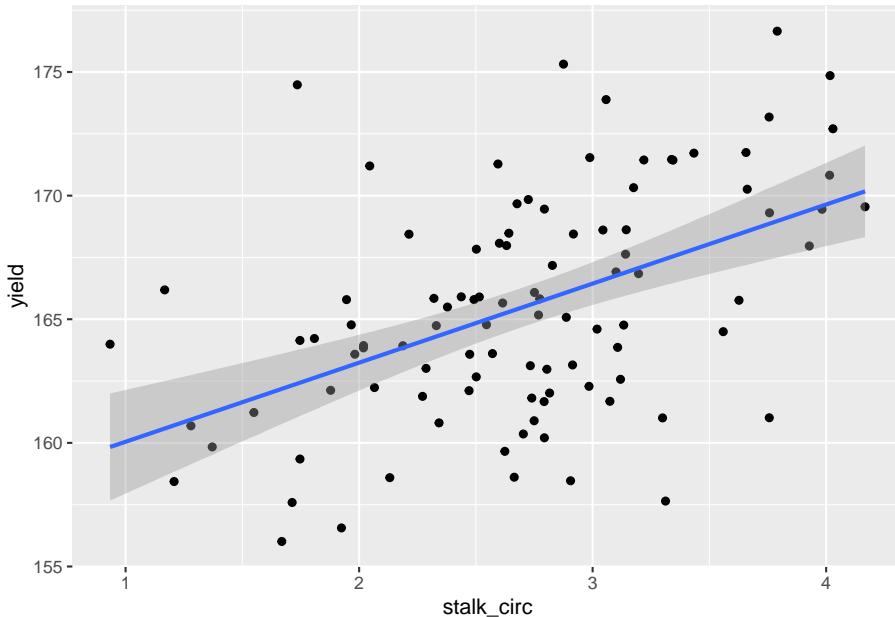
We see the r.squared is about 0.24 . Not particularly strong, but meaningful just the same.

### 10.7.3 Visualizing the Regression Model

Let's visualize this relationship using `ggplot()`.

```
model_circumference %>%
  ggplot(aes(x=stalk_circ, y=yield)) +
  geom_point() +
  geom_smooth(method = "lm", se=TRUE)

## `geom_smooth()` using formula 'y ~ x'
```



Above, we first created a scatter plot, based on the stalk circumference and yield associated with each observation. The first two lines of plot creation, `ggplot()` and `geom_point()` shoud be increasingly familiar to you.

The next line, uses the `geom_smooth()` function to add our regression model. The argument `method="lm"` tells R to fit a linear model to the data. This model is identical to that we used in our regression analysis. The second argument, `se=TRUE`, tells R to shade the confidence interval around the regression line.

Remember, our regression line is an estimate, based on samples. Our model parameters, including our slope, are estimates, complete with standard errors. The confidence interval is based on the slope of our line. Because a change in the slope of the line would be more pronounced at the ends of our regression line than the middle (think of it like a propeller), our confidence interval has a

convex (or hourglass) shape to it. We are 95% confident the actual relationship between Y and X is included in this confidence interval.

#### 10.7.4 Making Predictions with the Regression Model

Once we have defined a simple linear regression model, it is easy to make predictions with it.

First, we need to create a new data frame with the values of stalk circumference for which we want to make predictions. We use the *data.frame()* function to create the data.frame. Next, we define the first column of the data.frame, *stalk\_circ*. We define it as having 5 values, from 1.5 to 3.5

```
new_stalk_data = data.frame(
  stalk_circ=c(1.5,
              2.0,
              2.5,
              3.0,
              3.5))

new_stalk_data
```

```
##   stalk_circ
## 1      1.5
## 2      2.0
## 3      2.5
## 4      3.0
## 5      3.5
```

That was probably the hardest part of the prediction. Now, all we need to do is run the *predict()* function. This function takes two arguments: 1) the name of the linear model we have developed, and 2) the name of the data.frame with the new values of *stalk\_circ*.

```
predict(model_circumference, new_stalk_data)
```

```
##       1       2       3       4       5
## 161.6444 163.2436 164.8429 166.4422 168.0415
```

We can also assign these values to a new column in our *new\_stalk\_data* dataframe.

```

new_stalk_data$yield = predict(model_circumference, new_stalk_data)

new_stalk_data

##   stalk_circ     yield
## 1      1.5 161.6444
## 2      2.0 163.2436
## 3      2.5 164.8429
## 4      3.0 166.4422
## 5      3.5 168.0415

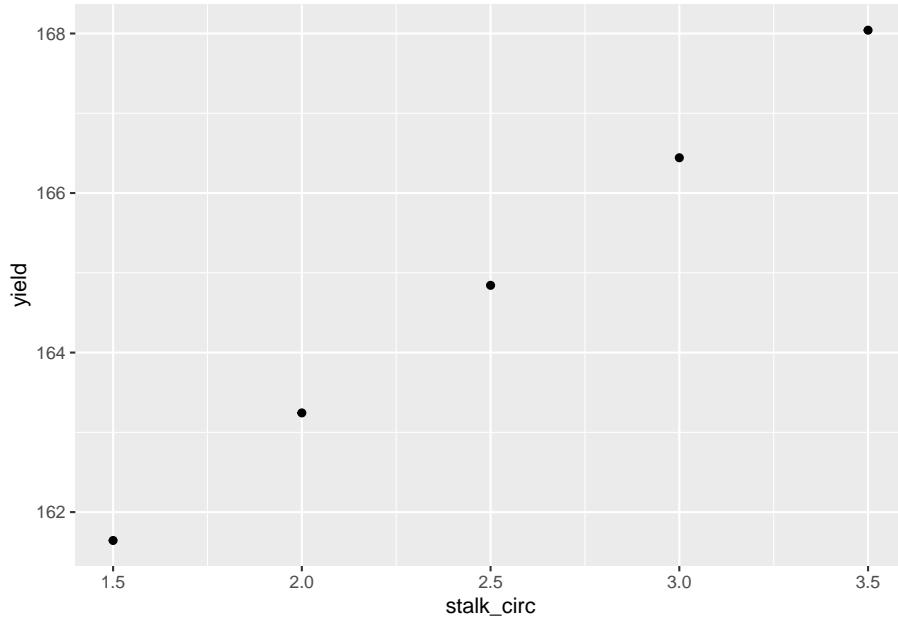
```

Finally, we can plot these new values using `ggplot()`

```

new_stalk_data %>%
  ggplot(aes(x=stalk_circ, y=yield)) +
  geom_point()

```

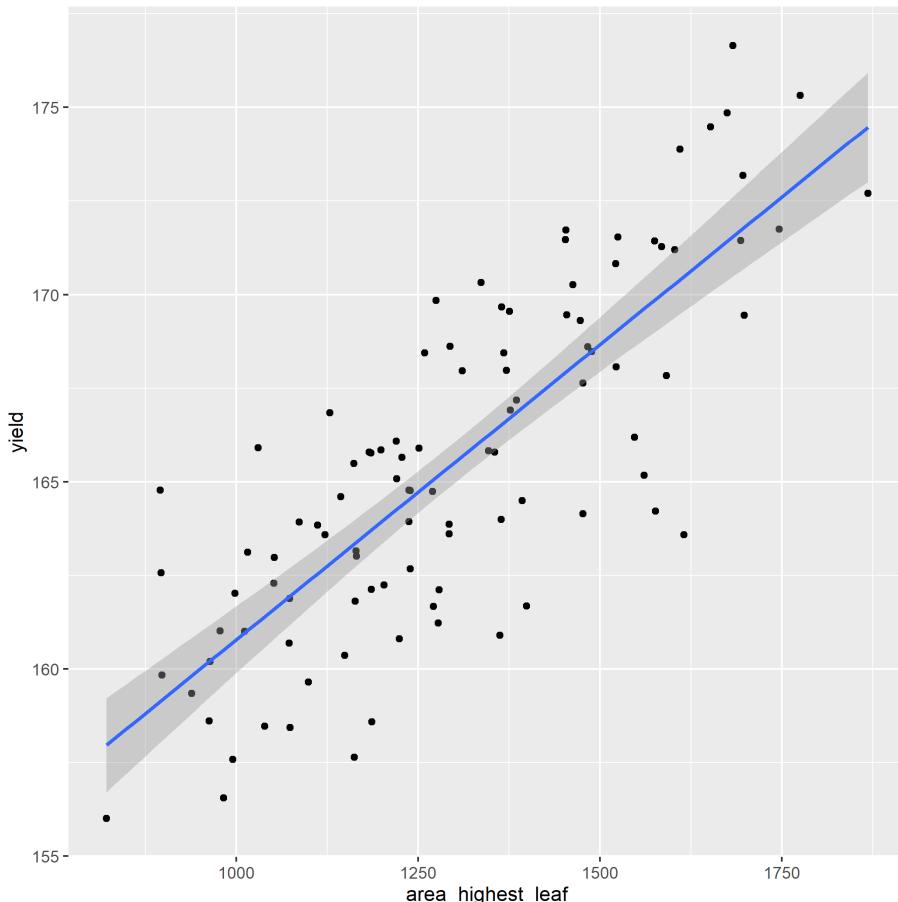


### 10.7.5 Practice 1

What effect does the area of the highest leaf have on yield? Let's find out.

- 1) Build the linear model

- 2) Examine the model coefficients. The estimate for area\_highest\_leaf should be about 0.016.
- 3) How strong is the relationship between highest leaf area and yield? Pretty strong: your results should show an r.squared of 0.64.
- 4) Create a plot of the regression model using ggplot(). Your plot should look like:



- 4) Make predictions for the five new leaf area values in the new\_leaf\_area data.frame

```
new_leaf_area = data.frame(
  area_highest_leaf = c(900,
    1000,
    1100,
```

```

    1200,
    1300
)
)

# Your new table should look like:
#
# area_highest_leaf yield
# 900   159.2116
# 1000  160.7872
# 1100  162.3629
# 1200  163.9385
# 1300  165.5141

```

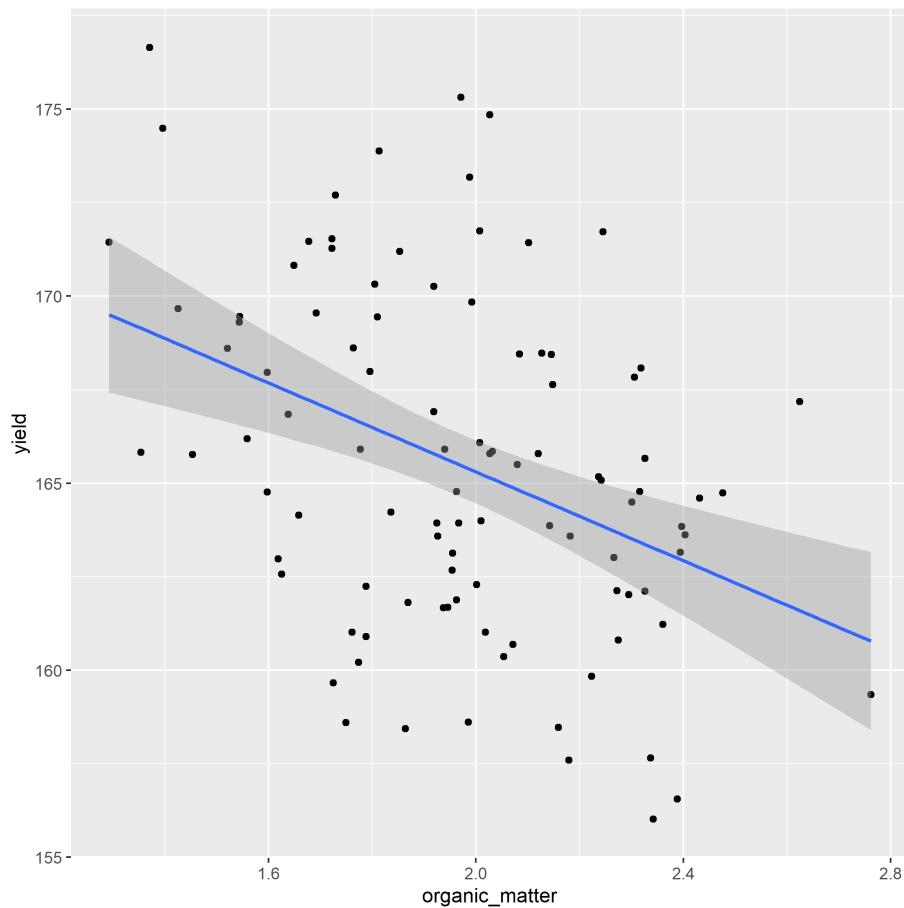
### 10.7.6 Practice 2

We will work with the corn fertility dataset once more.

```
fertility = read.csv("data-unit-10/exercise_data/corn_fertility.csv")
head(fertility)

##      yield organic_matter magnesium      potash phosphorus      pH
## 1 166.0838     2.007710 101.87145    7.214161 105.57985 6.287836
## 2 162.2875     2.001714 122.71640   163.097239   42.50915 7.304132
## 3 169.4557     1.544267  62.12231   26.855054 193.89253 7.494299
## 4 167.9799     1.795988  65.36950   31.239609   84.29568 7.221934
## 5 173.1781     1.988002 227.62357  -30.477244  158.02454 6.573853
## 6 168.4464     2.084489  91.22296   62.028413 100.50722 7.108826
```

- 1) Create the linear model for yield as a function of phosphorus.
- 2) Examine the model coefficients. The estimate for phosphorus should be about -5.94.
- 3) How strong is the relationship between highest leaf area and yield? Not very strong: your results should show an r.squared of 0.15.
- 4) Create a plot of the regression model using ggplot(). Your plot should look like:



- 4) Make predictions for the five new organic matter values in the new\_om data.frame

```
new_om = data.frame(  
  organic_matter = c(1.5,  
                     1.8,  
                     2.1,  
                     2.4,  
                     2.7  
  ))  
  
# Your new table should look like:
```

10.7. EXERCISE: MAKING PREDICTIONS FROM REGRESSION MODELS.371

```
#  
# organic_matter      yield  
# 1.5    168.2745  
# 1.8    166.4925  
# 2.1    164.7104  
# 2.4    162.9283  
# 2.7    161.1463  
#
```



## Chapter 11

# Nonlinear Relationships and Multiple Linear Regression

In the last unit, we learned how to describe linear relationships between two variables,  $X$  and  $Y$ . *Correlation* was used when we wanted to measure the *association* between the two variables. This was appropriate when when, based on our “domain knowledge” (agronomy or our other specialty), we did not have insight into whether one variable affected the other, or whether a third, unknown variable affected the value of both. *Simple Linear Regression* was used when we had insight into the *causation* that linked two variables: we knew or hypothesized that a single response variable,  $Y$ , was affected (and could be predicted) by a single predictor variable,  $X$ .

Simple linear regression, however, has its limitations. First, simple linear regression is often too inadequate for modelling more complex systems. For example, a simple linear regression model might fit the effect of rainfall on corn yield. But we all realize that a prediction of corn yield based on rainfall alone will not be particularly accurate.

If we included additional predictor variables, such as a measure of heat (cumulative growing degree days) or soil texture (water holding capacity or clay content), we would likely predict yield more accurately. *Multiple Linear Regression* allows us to build model the relationship between a response variable,  $Y$ , and multiple predictor variables,  $X_1$ ,  $X_2$ ,  $X_3$ , etc.

Second, a linear model assumes the relationship between  $Y$  and  $X$  can be fit with a straight line. If you have taken a soil science course, however, you learned about *Leibig’s Law of the Minimum* and the *Mitscherlich Equation* that describe the relationship between nutrient availability and plant biomass. These

relationships are curved and need to be fit with *Nonlinear Regression* models.

In this unit, we will learn how to use multiple linear regression to model yield responses to environment. We will also learn how to model nonlinear relationships, such as fertilizer response and plant growth

## 11.1 Multiple Linear Regression

In multiple linear regression, response variable Y is modeled as a function of multiple X variables:

$$Y = \mu + X_1 + X_2 + X_3 \dots X_n$$

### 11.1.1 Case Study: Modelling Yield by County

We are going to work with a county-level dataset from Purdue university that includes soil characteristics, precipitation, corn, and soybean yield. I've become best-friends with this dataset during the past few years and have combined it with county-level weather data to build complex yield models. The authors used land use maps to exclude any acres not in crop production. Click the following link to access a little app with which to appreciate this dataset.

Here is the top of the data.frame:

```
library(tidyverse)

## -- Attaching packages ----- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr    0.3.4
## v tibble  3.1.6     v dplyr    1.0.8
## v tidyr   1.2.0     v stringr  1.4.0
## v readr   2.1.2     vforcats  0.5.1

## Warning: package 'ggplot2' was built under R version 4.1.3

## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()   masks stats::lag()

counties = read.csv("data-unit-11/county_environments.csv")

head(counties)
```

```

##      county state stco    ppt     whc     sand     silt     clay      om
## 1 Abbeville   SC 45001 562.2991 22.94693 39.26054 21.38839 39.351063 36.92471
## 2 Acadia     LA 22001 751.1478 37.83115 12.82896 54.19231 32.978734 105.79133
## 3 Accomack   VA 51001 584.6807 18.56290 74.82042 16.82072 8.358863 103.11425
## 4 Ada        ID 16001 124.9502 15.54825 44.49350 40.48000 15.026510 77.61499
## 5 Adair       IA 19001 598.4478 28.91851 17.82011 48.44056 33.739326 259.53255
## 6 Adair       KY 21001 698.6234 20.87349 15.66781 48.16733 36.164866 73.05356
##      kwfactor kffactor    sph    slope tfactor    corn soybean cotton
## 1 0.2045719 0.2044325 5.413008 42.59177 4.666834 57.75000 16.75385 525.500
## 2 0.4432803 0.4431914 6.370533 34.92266 4.855202 92.42500 28.72857     NA
## 3 0.2044544 0.2044544 5.166446          NA 4.011007 116.99714 30.44857 575.125
## 4 0.4106704 0.4198545 7.648190          NA 2.853957 145.74138     NA     NA
## 5 0.3541369 0.3541369 6.215139 55.24201 4.691089 133.02571 41.70857     NA
## 6 0.3166089 0.3725781 5.351832 37.65075 4.119289 95.76765 37.50000     NA
##      wheat
## 1 30.17500
## 2 38.11000
## 3 56.07407
## 4 97.26296
## 5 34.04000
## 6 37.42105

```

How does corn yield respond to soil properties and precipitation in Minnesota and Wisconsin? Let's say we want to model corn yield as a function of precipitation (ppt), percent sand (sand), percent clay (clay), percent organic matter (om) and soil pH (sph)? Our linear additive model would be:

$$Y = \alpha + \beta_1 \text{ppt} + \beta_2 \text{sand} + \beta_3 \text{clay} + \beta_4 \text{om} + \beta_5 \text{sph} + \epsilon$$

First, we need to filter the dataset to MN and WI

```

counties_mn_wi = counties %>%
  filter(state %in% c("MN", "WI"))

```

To fit our model, we use the linear model `lm()` function of R. For multiple regression, we don't list any interactions between the terms.

```

model_mn_wi = lm(corn ~ ppt + sand + clay + om + sph, data=counties_mn_wi)
summary(model_mn_wi)

```

```

##
## Call:
## lm(formula = corn ~ ppt + sand + clay + om + sph, data = counties_mn_wi)
## 
```

```

## Residuals:
##      Min     1Q Median     3Q    Max
## -42.432 -4.342  1.108  7.035 23.278
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -189.02125   22.76727 -8.302 5.52e-14 ***
## ppt          0.37140    0.02165  17.159 < 2e-16 ***
## sand         -0.33075   0.10746 -3.078  0.00248 **
## clay         -1.16506   0.24579 -4.740 4.92e-06 ***
## om           -0.01679   0.00307 -5.469 1.85e-07 ***
## sph          24.21303   2.02991 11.928 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.61 on 150 degrees of freedom
## (3 observations deleted due to missingness)
## Multiple R-squared:  0.7651, Adjusted R-squared:  0.7573
## F-statistic: 97.71 on 5 and 150 DF,  p-value: < 2.2e-16

```

Let's go through the results. The first output item, the Call, is the linear model we gave to R.

The second item, residuals, describes the distribution of residuals around the regression model. We can see the minimum residual is 42 bu/acre below the predicted value. The maximum residual is 23 bu/acre above the predicted value. The middle 50% (between 1Q and 3Q) were between 4.3 below and 7.0 above the predicted values.

The Coefficients table is the highlight of our output. The table shows the estimated slopes  $\beta_1, \beta_2, \beta_3, \dots, \beta_n$  associated with each environmental factor, plus the individual t-tests of their significance. We can see that each factor of our model is significant.

In the bottom, we see that three observations were deleted due to missingness (they didn't have a value for each factor). Two  $R^2$ 's are presented. The multiple  $R^2$  is the proportion of the total variance in the model explained by our regression model. This is the same concept as for simple linear regression. Our value is 0.77, which is pretty good for an environmental model like this, especially because we did not include any temperature data.

### 11.1.2 Beware of Bloated Models

Multiple Linear Regression is a powerful tool, but it generally pays to be conservative in how many factors you include in a model. The more terms you include, the more you are likely to encounter problems with overfitting, multicollinearity, and heteroscedasticity.

### 11.1.2.1 Overfitting

Every model contains fixed terms (which the model is meant to measure and predict) and random terms, such as error, which are beyond the scope of the model to predict. Not only that: since the error effect is random, it would be wrong for a model to try to predict it. If we add enough factors to a model, however, it will do exactly that. It will *overfit* the data. The problem with overfitting is that the model contorts itself to fit every nook and cranny of one dataset – but fails to accurately predict values for additional datasets that, randomly, have different error structures.

Here is an analogy. You let your spouse or best friend borrow your car. When they return it, the seating settings are completely messed up. The lumbar support is either poking you in your but or about to break your neck. You either can't reach the pedals and the steering wheel, or the seat is up so far you cannot even get in the car. In addition, the climate control is too hot or too cold. And, seeing what they left on the radio, you start to rethink the whole relationship.

This is exactly the problem with overfitting. The more we perfect the fit of a model one dataset, the more unlikely it is to make accurate predictions for another dataset.

Adding factors to a model will always increase the  $R^2$  value, even if the new factor has nothing to do with what the model is predicting. For fun, lets create a column that randomly assigns an average number of Grateful Dead concerts attended per person, from 1 to 40, to each county.

```
set.seed(092220)
counties_mn_wi_gd = counties_mn_wi %>%
  mutate(gd_concerts = sample(c(1:40), 159, replace = TRUE))
head(counties_mn_wi_gd)

##   county state stco     ppt     whc     sand     silt     clay      om
## 1    Adams    WI 55001 548.2443 15.67212 78.38254 11.71815 9.899312 438.3402
## 2   Aitkin    MN 27001 485.5737 28.30823 47.94610 38.02355 14.030353 1760.8355
## 3   Anoka    MN 27003 551.9312 24.75840 75.16051 18.69882 6.140666 1787.7790
## 4  Ashland    WI 55003 495.9158 21.11778 43.10370 32.20112 24.695183 274.1879
## 5   Barron    WI 55005 528.5029 22.30522 58.10407 31.03742 10.858506 231.5082
## 6 Bayfield    WI 55007 490.7059 21.75064 34.08119 29.80771 36.111094 137.1594
##   kwfactor kffactor     sph     slope tfactor      corn soybean cotton
## 1 0.1170651 0.1179408 6.068121 65.21191 4.521995 112.78000 34.08857     NA
## 2 0.3114794 0.3126979 6.405352 25.42843 3.700648 83.21111 27.98462     NA
## 3 0.1505643 0.1513100 6.248463 28.34023 4.432518 101.93429 27.70000     NA
## 4 0.2977456 0.3111864 6.638459 62.67817 4.265288 88.78421 24.00000     NA
## 5 0.2886149 0.2924627 5.710398 48.65562 3.943001 111.98000 33.79429     NA
## 6 0.2927169 0.2933493 7.166786 63.60773 4.636175 85.72857 22.40000     NA
```

```
##      wheat gd_concerts
## 1 40.32500      18
## 2 30.64000      27
## 3 35.05000      34
## 4 32.82857      37
## 5 44.49630      33
## 6 31.21429       3
```

And then let's check out our model:

```
model_mn_wi_gd = lm(corn ~ ppt + sand + clay + om + sph + gd_concerts, data=counties_mn_wi_gd)
summary(model_mn_wi_gd)
```

```
##
## Call:
## lm(formula = corn ~ ppt + sand + clay + om + sph + gd_concerts,
##     data = counties_mn_wi_gd)
##
## Residuals:
##   Min     1Q Median     3Q    Max
## -41.881 -4.347  1.033  6.632 23.862
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.897e+02  2.284e+01 -8.304 5.66e-14 ***
## ppt          3.707e-01  2.173e-02 17.059 < 2e-16 ***
## sand         -3.334e-01  1.078e-01 -3.093  0.00236 **
## clay         -1.169e+00  2.464e-01 -4.745 4.84e-06 ***
## om           -1.690e-02  3.082e-03 -5.483 1.75e-07 ***
## sph          2.426e+01  2.036e+00 11.917 < 2e-16 ***
## gd_concerts  4.393e-02  7.361e-02  0.597  0.55155
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.64 on 149 degrees of freedom
##   (3 observations deleted due to missingness)
## Multiple R-squared:  0.7656, Adjusted R-squared:  0.7562
## F-statistic: 81.13 on 6 and 149 DF,  p-value: < 2.2e-16
```

See? Our Multiple  $R^2$  increased slightly from 0.765 to 0.766. While this example is absurd, it points to a temptation for statisticians and data scientists: keep adding terms until the *MultipleR<sup>2</sup>* is an accepted value. Less nefariously, the researcher may just think the more variables the researcher can add, the better. We can now see this is wrong.

### 11.1.2.2 Multicollinearity

Another problem that can occur as we add factors to our model is *multicollinearity*. Our linear model assumes that each factor has an *independent* effect on the response variable. We know, however, that is not always the case. For example, all sorts of weather and soil factors in our model can be related. Cloudy days associated with precipitation may decrease temperature. Warm temperatures may reduce soil moisture. Areas with greater precipitation will tend to have more woody vegetation, leading to lower soil pH and soil organic matter. Coarser soils, which are better drained, will tend to have lower soil organic matter.

Here is a hypothetical scenario: suppose we look at the regression of yield on precipitation and pH and conclude yield is significantly affected by both of them. Do we know that soil pH caused the change in yield? No, it's possible that precipitation affected both pH and yield so that they appeared to change together. That's not to say we can't include all of these factors in our model. But we need to make certain that they are directly causing changes *to* the response variable, rather than responding *with* the response variable to a third variable in the model.

### 11.1.2.3 Heteroscedasticity

The third problem we can have with a multiple linear regression model is that it is *heteroscedastic*. This intimidating term refers to unequal variances in our model. That is, the variance of observed yields varies markedly with the value being predicted. Multiple linear regression, like other linear models, assumes that the variance will be the same along all levels of the factors in the model. When heteroscedasticity – unequal variances – occurs, it jeopardizes our ability to fit the that factor. Our least squared estimate for that factor will be more influenced by factor levels with greater variances and less influences by levels with lesser variances.

One of the causes of heteroscedasticity is having many predictors – each with their own scales of measure and magnitudes – in a model. Since linear regression relies on least squares – that is, minimizing the differences between observed and predicted values – it will be more influenced by factors with greater variances than factors with small variances, regardless of how strongly each factor is correlated with yield.

The end result is that the regression model will fit the data more poorly. Its error will be greater and, thus, its F-value and p-value will be reduced. This may lead us to conclude a factor or the overall model does not explain a significant proportion of the variation in data, when in fact it does. In other words, we may commit at Type II error.

### 11.1.3 Methods for Avoiding Bloated Models

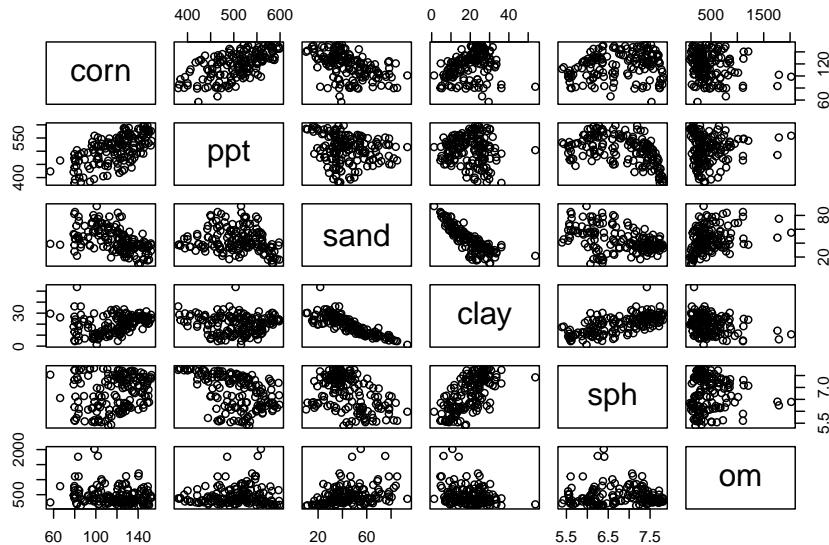
There are multiple approaches, and surely books written, about how to avoid overfitting and multicollinearity in models. After all that, also, model tuning (selecting the factors to include) seems to be art as much as science. This section provides an overview of methods used, which you might consider if presented with someone else's results, or if you are trying to construct a model with a few factors on your own.

#### 11.1.3.1 Create a Covariance Matrix

A matrix (not “the matrix”) is a mathematical term that describes what you and I would call a table. So a covariance matrix is a table composed of correlation plots that we can use to inspect the covariance (or relationship) between each possible pair of variables in our dataset.

```
yield_predictors = counties_mn_wi %>%
  dplyr::select(corn, ppt, sand, clay, sph, om) %>%
  na.omit() %>%
  as.data.frame()

plot(yield_predictors)
```



To use the matrix, find the intersection between a column and a row containing two variables whose relationship you want to inspect. For example, in the fourth

column, clay is plotted in relationship to the four other predictor variables, plus the response corn, in our model. In each plot, clay is on the X-axis, and the intersecting variable is on the Y-axis. Looking at the matrix, we notice the relationship of clay with sand and soil pH (sph) is visible. We may wonder, then, if the addition of clay to our model is improving our prediction.

### 11.1.3.2 Partial Correlation

Another thing we can do to look for multicollinearity is to calculate the partial correlations. This can also be done with a simple line of code. Partial correlation shows the individual correlations between variables, with all other variables being held constant. What this does is allow us to quantify the correlation between two variables without worrying that both may be affected by a third variable. For example, we can look at the correlation between soil pH and soil organic matter without worrying that precipitation might be driving changes in both we could mistake for a relationship.

```
library(psych)

## 
## Attaching package: 'psych'

## The following objects are masked from 'package:ggplot2':
## 
##     %+%, alpha

partial.r(yield_predictors)

##          corn      ppt      sand      clay      sph      om
## corn  1.0000000  0.81392756 -0.24374133 -0.3609330  0.6977065 -0.4077095
## ppt   0.8139276  1.00000000 -0.03904489  0.2401821 -0.7798037  0.4618512
## sand  -0.2437413 -0.03904489  1.00000000 -0.7743363  0.1767851  0.1169185
## clay  -0.3609330  0.24018214 -0.77433626  1.0000000  0.5772645 -0.1893945
## sph   0.6977065 -0.77980370  0.17678512  0.5772645  1.0000000  0.4490715
## om   -0.4077095  0.46185116  0.11691848 -0.1893945  0.4490715  1.0000000
```

The output of partial correlation is a matrix, which cross-tabulates the correlations among every predictor variables and reports their values in a table. The output above tells us that sand and clay are both negatively correlated with corn. That's odd – we would expect that as sand decreases, clay would increase, or vice versa – unless, both are being impacted by a third value, silt, which is not in our model.

### 11.1.3.3 Cross Validation

A third way to evaluate the performance of a model and to avoid mistaking over-prediction for true model performance is to use cross-validation. I'll confess to graduating from Iowa State and making it another 15 years in academia and industry without having a clue about how important this is. If you plan to use your model not only to test hypotheses about variable significance, but to make predictions, the cross-validation is critical.

In cross-validation, the initial data are divided into *training* and *testing* groups. The model parameters (coefficients for slopes) are solved for using the training dataset. In general, all models will better fit the data used to train them. Therefore, the predictive performance of the model is measured using the testing dataset. In this way, the true performance of the model can be measured. In addition, the effect of adding or removing factors may be measured.

We will use a technique that sounds vaguely like an adult-alternative music group – Repeated 10-Fold Cross Validation. While the name sounds scary as all get-out, how it works is (pretty) straight forward:

1. Divide the data into 10 groups.
2. Randomly select 9 groups and fit the regression model to them. These groups are called the “training” dataset
3. Predict the responses for each observation in the tenth dataset, using the model fit to the other 9 datasets. This 10th dataset is called the “testing” dataset.
4. Use linear regression to determine the strength of the relationship between the predicted value and the actual values. Review summary statistics in comparing models.

Here is the cross validation of our first yield model for Illinois and Ohio:

```
library(caret)

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##      lift
```

```

# Define training control
set.seed(123)
train.control <- trainControl(method = "repeatedcv",
                               number = 10, repeats = 3)
# Train the model
model <- train(corn ~ ppt + sand + clay + om + sph, data=yield_predictors, method = "lm",
                trControl = train.control)
# Summarize the results
print(model)

## Linear Regression
##
## 156 samples
##   5 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 140, 140, 141, 140, 140, 140, ...
## Resampling results:
##
##   RMSE      Rsquared     MAE
##   10.73952  0.7632216  8.203451
##
## Tuning parameter 'intercept' was held constant at a value of TRUE

```

We can see in the top of the output confirmation we are working with the original 5 predictors. To measure model performance, let's look particularly at the MRSE and Rsquared statistics at the bottom. RMSE is the Root Mean Square Error, which you earlier learned is the square root of the Mean Square Error, and equivalent to the standard deviation of our data. This is expressed in our regression output above as residual standard error. An RMSE of 10.7 means the distribution of residuals (observed values) around our model predictions has a standard deviation of about 10.7. Thus 95% of our observed values would be expected to be within 21.4 bushels ( $2 * 10.7$ ) of the predicted value.

The Rsquared statistic is the same as we have seen previously, and describes the amount of variation in the observed values explained by the predicted values.

#### 11.1.4 Tunning and Comparing Models

Here is our original model for reference. Note that sand has a negative effect (-0.33075) on yield and that it is highly significant ( $p=0.00248$ ).

```
full_model = lm(corn ~ ppt + sand + clay + om + sph, data=counties_mn_wi)
summary(full_model)

##
## Call:
## lm(formula = corn ~ ppt + sand + clay + om + sph, data = counties_mn_wi)
##
## Residuals:
##   Min     1Q Median     3Q    Max
## -42.432 -4.342  1.108  7.035 23.278
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -189.02125  22.76727 -8.302 5.52e-14 ***
## ppt          0.37140   0.02165 17.159 < 2e-16 ***
## sand         -0.33075  0.10746 -3.078 0.00248 **
## clay        -1.16506  0.24579 -4.740 4.92e-06 ***
## om           -0.01679  0.00307 -5.469 1.85e-07 ***
## sph          24.21303  2.02991 11.928 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.61 on 150 degrees of freedom
##   (3 observations deleted due to missingness)
## Multiple R-squared:  0.7651, Adjusted R-squared:  0.7573
## F-statistic: 97.71 on 5 and 150 DF,  p-value: < 2.2e-16
```

We saw above that clay was strongly correlated with both sand and soil pH. Let's drop clay from our model and see what happens:

```
model_no_clay = lm(corn ~ ppt + sand + om + sph, data=counties_mn_wi)
summary(model_no_clay)

##
## Call:
## lm(formula = corn ~ ppt + sand + om + sph, data = counties_mn_wi)
##
## Residuals:
##   Min     1Q Median     3Q    Max
## -50.342 -5.438  1.410  7.135 22.276
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.031e+02  2.412e+01 -8.421 2.69e-14 ***
```

```

## ppt      3.815e-01  2.302e-02  16.575 < 2e-16 ***
## sand     5.577e-02  7.479e-02   0.746   0.457
## om      -1.607e-02  3.277e-03  -4.903  2.41e-06 ***
## sph      1.953e+01  1.895e+00  10.306 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.34 on 151 degrees of freedom
##   (3 observations deleted due to missingness)
## Multiple R-squared:  0.7299, Adjusted R-squared:  0.7227
## F-statistic:  102 on 4 and 151 DF,  p-value: < 2.2e-16

```

First, let's look at the model fit. The Multiple R-squared decreased from 0.7651 to 0.7299. The residual standard error increased from 10.61 to 11.34. These would suggest the fit of the model has decreased, although we also notice the F-statistic has increased from 97.7 to 102, which suggests the model itself is more strongly detecting the combined effects of the factors on yield,

An additional statistic that we want to watch is the Adjusted R-Squared. This statistic not only takes into effect the percentage of the variation explained by the model, but how many factors were used to explain that variance. The model is penalized for according to the number of factors used: of two models that explained the same amount of variation, the one that used more factors would have a lower Adjusted R-square. We see the adjusted R-square decreased from our first model to the second.

Now let's go back to sand. In the first model, it had a negative effect of -0.33075 and was highly significant. Now it has a positive effect of 5.577e-02 and an insignificant effect on corn yield. Remember, each factor in a linear regression model should be independent of the other factors. If we compare the other four factors, we will see their coefficients have changed slightly, but they have remained highly significant. This suggests that clay was affecting both sand content (after all, if you have more clay, you are likely to have less sand) and yield.

Let's cross-validate the new model.

```

library(caret)

# Define training control
set.seed(123)
train.control <- trainControl(method = "repeatedcv",
                               number = 10, repeats = 3)

# Train the model
model <- train(corn ~ ppt + sand + om + sph, data=yield_predictors, method = "lm",
                trControl = train.control)

```

```
# Summarize the results
print(model)

## Linear Regression
##
## 156 samples
##   4 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 140, 140, 141, 140, 140, ...
## Resampling results:
##
##   RMSE      Rsquared     MAE
##   11.18499  0.7413634  8.347023
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

We see the Rsquared and RMSE (root mean square error) are statistics are slightly lower than the original model in the cross-validation, too.

Since sand was insignificant in the second model, let's remove it and rerun our model.

```
model_no_clay_sand = lm(corn ~ ppt + om + sph, data=counties_mn_wi)
summary(model_no_clay_sand)
```

```
##
## Call:
## lm(formula = corn ~ ppt + om + sph, data = counties_mn_wi)
##
## Residuals:
##   Min     1Q Median     3Q    Max
## -50.786 -5.803  1.263  6.637 22.088
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.904e+02  1.702e+01 -11.187 < 2e-16 ***
## ppt         3.724e-01  1.943e-02 19.167 < 2e-16 ***
## om          -1.501e-02  2.949e-03 -5.089 1.05e-06 ***
## sph         1.863e+01  1.465e+00 12.720 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```

## Residual standard error: 11.32 on 152 degrees of freedom
##   (3 observations deleted due to missingness)
## Multiple R-squared:  0.7289, Adjusted R-squared:  0.7236
## F-statistic: 136.2 on 3 and 152 DF,  p-value: < 2.2e-16

```

We see little change in the Multiple R-squared or Adjusted R-squared, but the F-statistic has again increased. Let's cross-validate the model.

```

library(caret)

# Define training control
set.seed(123)
train.control <- trainControl(method = "repeatedcv",
                               number = 10, repeats = 3)
# Train the model
model <- train(corn ~ ppt + om + sph, data=yield_predictors, method = "lm",
                trControl = train.control)
# Summarize the results
print(model)

## Linear Regression
##
## 156 samples
##   3 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 140, 140, 141, 140, 140, 140, ...
## Resampling results:
##
##   RMSE      Rsquared     MAE
##   11.15776  0.7424514  8.284529
##
## Tuning parameter 'intercept' was held constant at a value of TRUE

```

We see the new model fits slightly better.

We could go on and on with the process. We might try adding silt into the model to replace the sand and clay we removed. Water holding capacity was a factor in the original dataset – we might try using that as a proxy, too. But the interations we have gone through show us that bigger models are not necessarily better (not by much, in any case). While we can build complex models with multiple linear regression, it is better not to when possible.

## 11.2 Nonlinear Relationships

As mentioned in the introduction, there are many relationships between variables that are nonlinear – that is, cannot be modelled with a straight line. In reality, few relationships in agronomy are perfectly linear, so by nonlinear we mean relationships where a linear model would systematically over-predict or underpredict the response variable. In the plot below, a disease population (infected plants per plot) is modeled as a function of days since infection.

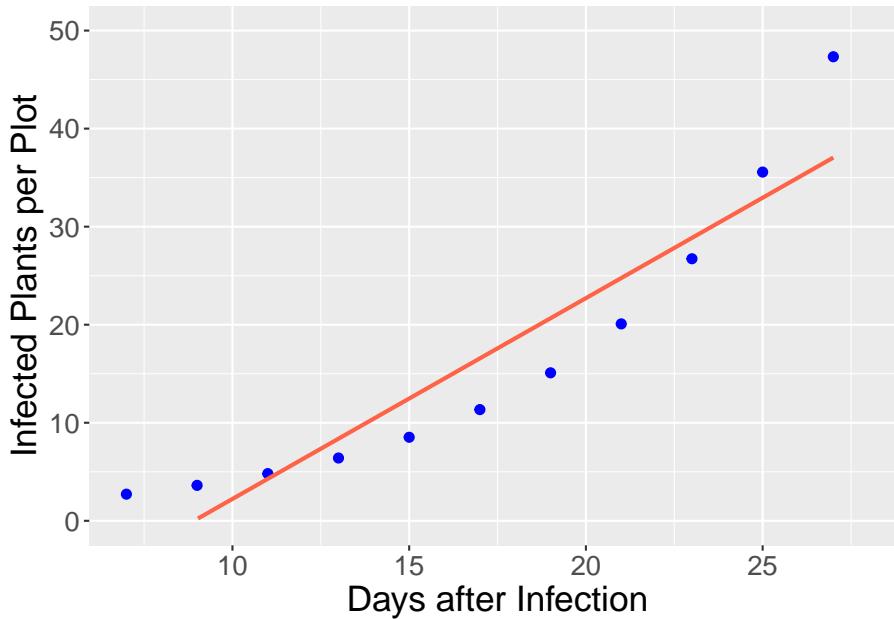
As with many pest populations, the number of infections increases exponentially with time. We can see how a linear model would underpredict the number of plants from 7 to 9 days after infection and again from 25 to 27 days after infection, while overpredicting the number of infected plants from 15 to 21 days after infection . Systematic overpredictions or underpredictions are called *bias*.

```
library(tidyverse)
exponential = data.frame(x = seq(from=7, to=28, by=2))
exponential_final = exponential %>%
  mutate(y = exp(x/7)) %>%
  mutate(log_y = log(y))

exponential_final %>%
  ggplot(aes(x=x, y=y)) +
  geom_point(size=2, color="blue") +
  geom_smooth(method='lm', color = "tomato", se=FALSE) +
  labs(x = "Days after Infection", y = "Infected Plants per Plot") +
  lims(x = c(7,28), y=c(0,50)) +
  theme(axis.title = element_text(size=18),
        axis.text = element_text(size=14))

## `geom_smooth()` using formula 'y ~ x'

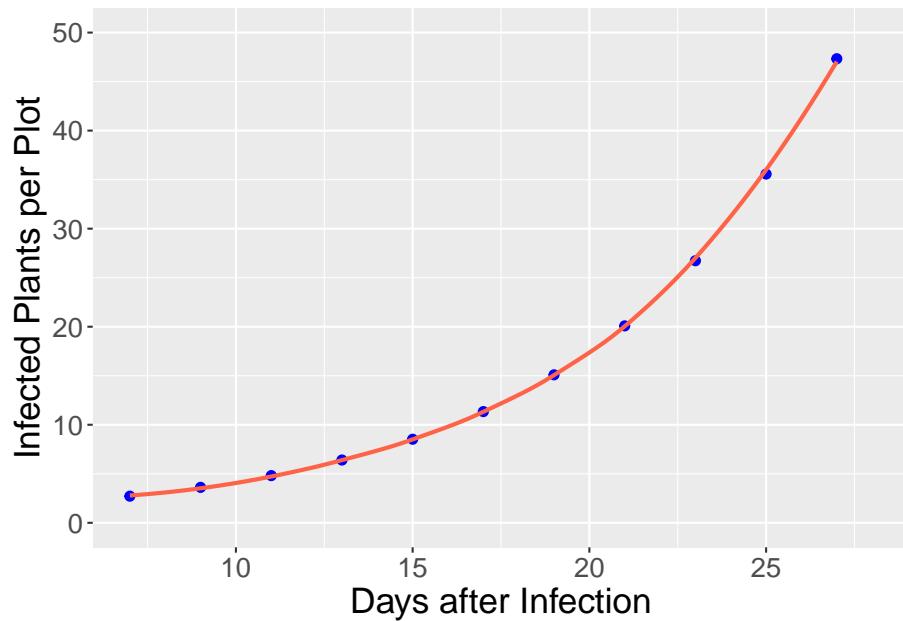
## Warning: Removed 8 rows containing missing values (geom_smooth).
```



Instead, we can fit the data with an exponential model that reduces the bias and increases the precision of our model:

```
exponential_final %>%
  ggplot(aes(x=x, y=y)) +
  geom_point(size=2, color="blue") +
  geom_smooth(method='loess', color = "tomato", se=FALSE) +
  labs(x = "Days after Infection", y = "Infected Plants per Plot") +
  lims(x = c(7,28), y=c(0,50)) +
  theme(axis.title = element_text(size=18),
        axis.text = element_text(size=14))

## `geom_smooth()` using formula 'y ~ x'
```



### 11.2.1 Fitting Nonlinear Responses with Linear Regression

Fitting a relationship with a simple linear regression model is simpler than fitting it with a nonlinear model, which we will soon see. Exponential relationships, like the one above can be fit by *transforming* the data to a new scale. This is the same concept as we used in an earlier unit to work with messy data.

#### 11.2.1.1 Exponential Model

For example, the data above (and many exponential relationships in biology and other disciplines can be transformed using the natural log:

$$y = \log(x)$$

If our original dataset looks like:

```
exponential_final %>%
  dplyr::select(x, y)
```

```
##      x          y
## 1    7  2.718282
```

```
## 2   9  3.617251
## 3 11  4.813520
## 4 13  6.405409
## 5 15  8.523756
## 6 17 11.342667
## 7 19 15.093825
## 8 21 20.085537
## 9 23 26.728069
## 10 25 35.567367
## 11 27 47.329930
```

Where x is the days since infection and y is the mean number of infections per plot. We can create a new column, log\_y with the natural log of infected plants.

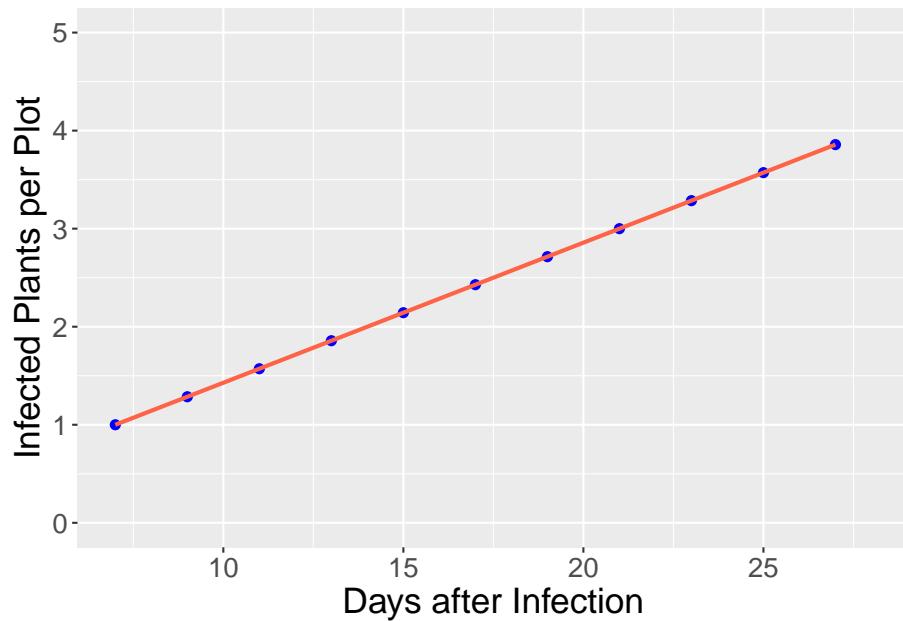
```
exponential_final %>%
  dplyr::select(x, y) %>%
  mutate(log_y = log(y))
```

```
##      x      y    log_y
## 1    7 2.718282 1.000000
## 2    9 3.617251 1.285714
## 3   11 4.813520 1.571429
## 4   13 6.405409 1.857143
## 5   15 8.523756 2.142857
## 6   17 11.342667 2.428571
## 7   19 15.093825 2.714286
## 8   21 20.085537 3.000000
## 9   23 26.728069 3.285714
## 10  25 35.567367 3.571429
## 11  27 47.329930 3.857143
```

When we fit the log of the infected plants, we see the relationship between the number of infected plants and days since infection is now linear.

```
exponential_final %>%
  ggplot(aes(x=x, y=log_y)) +
  geom_point(size=2, color="blue") +
  geom_smooth(method='loess', color = "tomato", se=FALSE) +
  labs(x = "Days after Infection", y = "Infected Plants per Plot") +
  lims(x = c(7,28), y=c(0,5)) +
  theme(axis.title = element_text(size=18),
        axis.text = element_text(size=14))

## `geom_smooth()` using formula 'y ~ x'
```



We can now fit a linear model to the relationship between infected plants and days after infection.

```

names

## function (x)  .Primitive("names")

infection_linear_model = lm(log_y ~ x, exponential_final)
summary(infection_linear_model)

## Warning in summary.lm(infection_linear_model): essentially perfect fit: summary
## may be unreliable

##
## Call:
## lm(formula = log_y ~ x, data = exponential_final)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -4.375e-16 -3.117e-16 -3.760e-18  2.469e-16  5.395e-16
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2.142e-15  3.108e-16 6.893e+00 7.12e-05 ***

```

```

## x           1.429e-01 1.713e-17 8.337e+15 < 2e-16 ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.594e-16 on 9 degrees of freedom
## Multiple R-squared:      1, Adjusted R-squared:     1
## F-statistic: 6.951e+31 on 1 and 9 DF, p-value: < 2.2e-16

```

Our linear model, from the equation above, is:

$$\log_y = 0 + 0.1429 \times x$$

Just to test the model, let's predict the number of infected plants when  $x = 15$

```

paste("log_y =", 0 + 0.1429*15)

## [1] "log_y = 2.1435"

```

We can see this value is approximately the number of infected plants 15 days after infection in the table above. Any predicted value can be transformed from the logarithm back to the original scale using the `exp()` function. Compare this with the original count,  $y$ , in the table above.

```

paste("y =", exp(2.1435))

## [1] "y = 8.52923778043145"

```

### 11.2.1.2 Parabolic

In nutrient availability and plant density models, we sometimes encounter data that are parabolic – the relationship between  $Y$  and  $X$  resembles a  $\cap$  or  $\cup$  shape. These data can be fit with a *quadratic model*. Remember that beast from eighth grade algebra. Don't worry – we don't have decompose it!

Let's say we have data from a plant density study in corn:

```

plant_density_data_pre = data.frame(pop = seq(28, 44, 4),
                                    yield = c(170, 200, 205, 200, 170)) %>%
  mutate(pop2 = pop^2)

pop_model = lm(yield ~ pop + pop2, data = plant_density_data_pre)

plant_density_data = data.frame(pop = seq(28, 44, 2)) %>%

```

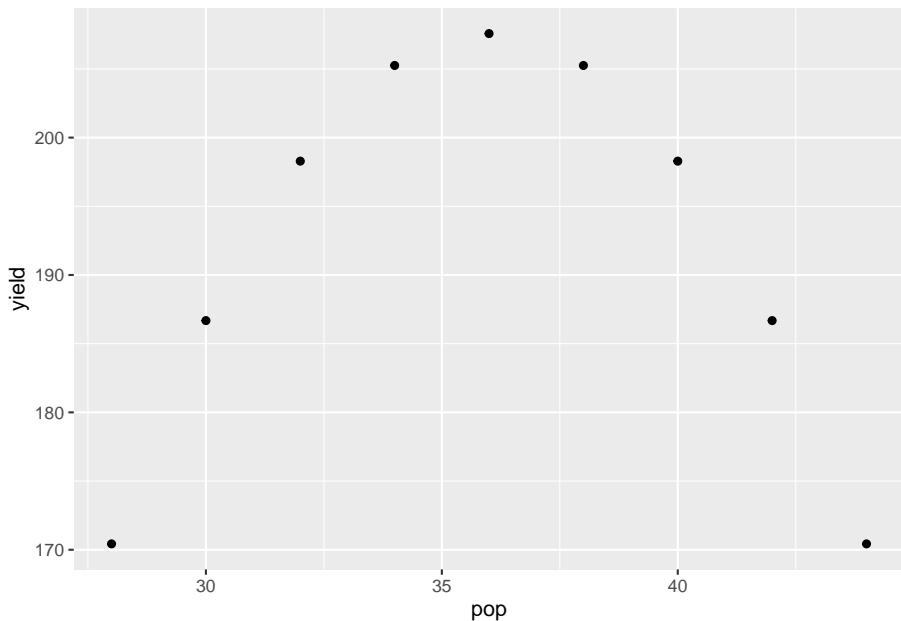
```

mutate(pop2 = pop^2)

plant_density_data$yield = predict(pop_model, plant_density_data)

plant_density_data %>%
  ggplot(aes(x = pop, y = yield)) +
  geom_point()

```



The quadratic model is:

$$Y = \alpha + \beta X + \gamma X^2$$

Where  $\alpha$  is the Y-intercept, and  $\beta$  and  $\gamma$  are the coefficients associated with  $X$  and  $X^2$ . We can run this model the same as we would a simple regression.

```

pop_model = lm(yield ~ pop + pop2, data = plant_density_data)
summary(pop_model)

```

```

## Warning in summary.lm(pop_model): essentially perfect fit: summary may be
## unreliable

##
## Call:
## lm(formula = yield ~ pop + pop2, data = plant_density_data)

```

```

## 
## Residuals:
##      Min       1Q   Median      3Q     Max 
## -8.565e-14 -3.115e-14 -7.869e-15  4.033e-14  7.242e-14 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -5.446e+02  1.043e-12 -5.223e+14 <2e-16 ***
## pop          4.179e+01  5.877e-14  7.111e+14 <2e-16 ***  
## pop2         -5.804e-01  8.146e-16 -7.125e+14 <2e-16 ***  
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
## 
## Residual standard error: 5.718e-14 on 6 degrees of freedom
## Multiple R-squared:      1, Adjusted R-squared:      1 
## F-statistic: 2.538e+29 on 2 and 6 DF, p-value: < 2.2e-16

```

This output should look very similar to the simple linear regression output. The only difference is that there are now three coefficients returned: the intercept ( $\alpha$  above), the coefficient for `pop` ( $\beta$  above), and the coefficient for `pop2` ( $\gamma$  above).

### 11.2.2 Fitting Nonlinear Responses with Nonlinear Regression

Other nonlinear relationships must be fit with nonlinear regression. Nonlinear regression differs from linear regression in a couple of ways. First, a nonlinear regression model may include multiple coefficients, but only X as a predictor variable. Second, models are not fit to nonlinear data using the same approach (using least square means to solve for slope) as with linear data. Models are often fit to nonlinear data often do so using a “trial and error” approach, comparing multiple models before converging on the model that fits best. To help this process along, data scientists must often “guesstimate” the initial values of model parameters and include that in the code.

#### 11.2.2.1 Monomolecular

In the Monomolecular growth model, the response variable Y initially increases rapidly with increases in X. Then the rate of increase slows, until Y plateaus and does not increase further with X. In the example below, the response of corn yield to nitrogen fertilization rate is modelled with the monomolecular (asymptotic) function.

First, we load and plot the data.

```

corn_n_mono = read.csv("data-unit-11/corn_nrate_mono.csv")
p = corn_n_mono %>%
  ggplot(aes(x=n_rate, y=yield)) +
  geom_point(color="blue")

```

Then we fit our nonlinear model.

```

corn_n_mono_asym = stats::nls(yield ~ SSasymp(n_rate, init, m, plateau), data=corn_n_mono)
summary(corn_n_mono_asym)

```

```

##
## Formula: yield ~ SSasymp(n_rate, init, m, plateau)
##
## Parameters:
##             Estimate Std. Error t value Pr(>|t|)
## init      160.6633    5.7363  28.01 5.24e-13 ***
## m         96.6297    4.3741  22.09 1.08e-11 ***
## plateau -4.2634    0.3019 -14.12 2.90e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.76 on 13 degrees of freedom
##
## Number of iterations to convergence: 0
## Achieved convergence tolerance: 6.133e-08

```

The most important part of this output is the bottom line, “Achieved convergence tolerance”. That means our model successfully fit the data.

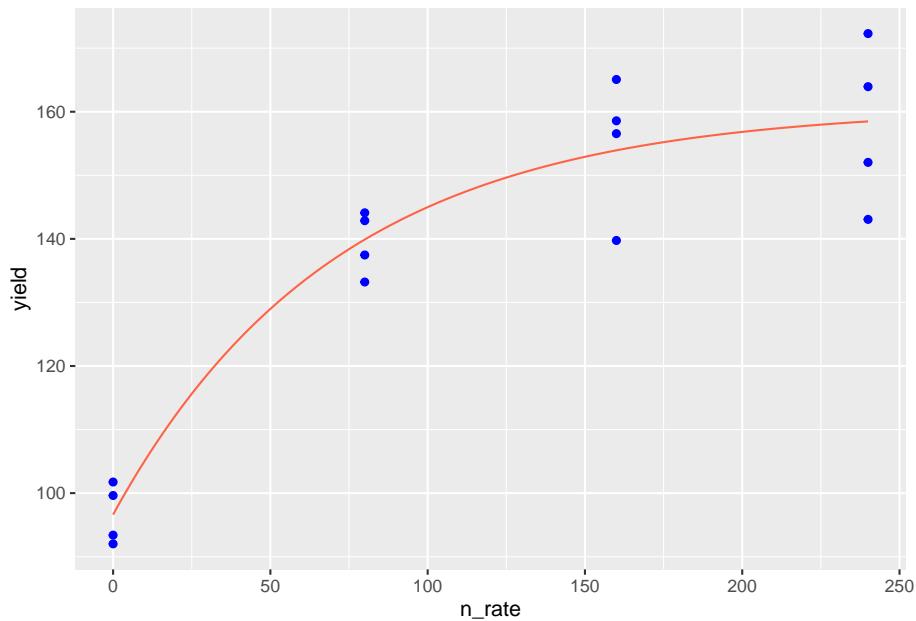
Finally, we can add our modelled curve to our initial plot:

```

test_data = data.frame(n_rate = seq(0, 240, 1))
test_data$pred = predict(corn_n_mono_asym, test_data)

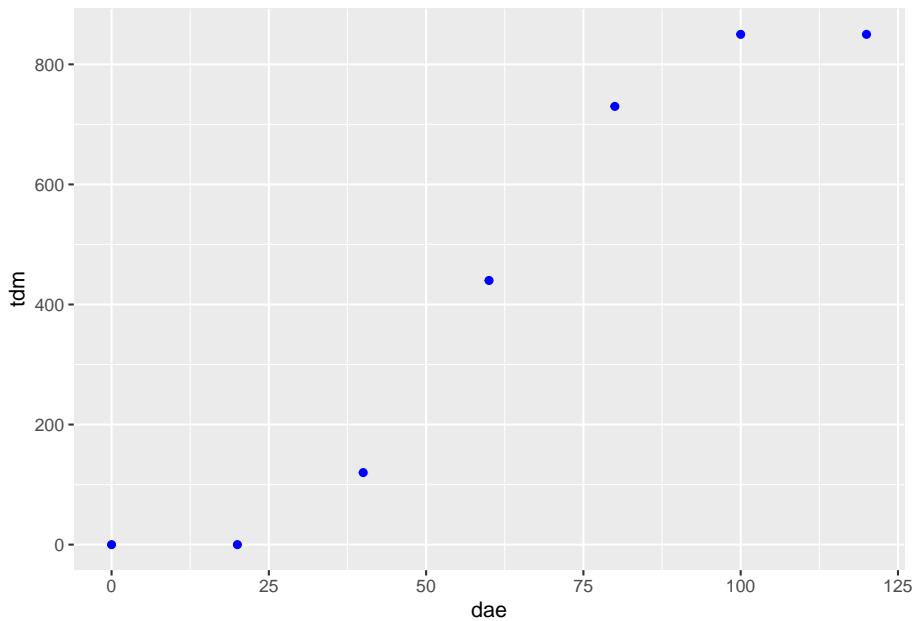
p + geom_line(data = test_data, aes(x=n_rate, y=pred), color="tomato")

```



```
geom_point(color="blue")
```

```
p_soy
```



The “S” shape of the data is very pronounced. Next, we fit a logistic growth curve to the data:

```
soybean_tdm_model = stats::nls(tdm ~ SSlogis(dae, Asym, xmid, scal), data=soybean_tdm)
summary(soybean_tdm_model)
```

```
##
## Formula: tdm ~ SSlogis(dae, Asym, xmid, scal)
##
## Parameters:
##             Estimate Std. Error t value Pr(>|t|)
## Asym     857.5043   11.7558  72.94 2.12e-07 ***
## xmid     59.7606    0.7525  79.42 1.51e-07 ***
## scal     10.8261    0.6574  16.47 7.96e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 14.64 on 4 degrees of freedom
##
## Number of iterations to convergence: 0
```

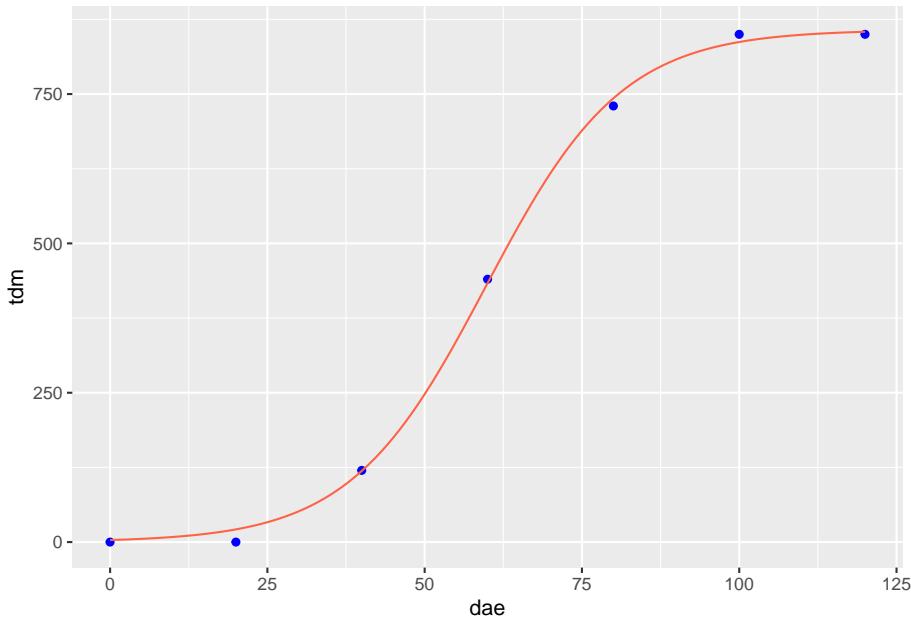
```
## Achieved convergence tolerance: 3.029e-06
```

We see again in the output that the model achieved convergence tolerance. Another thing to note is the “Number of iterations to convergence”. It took this model 9 steps to fit the data. The algorithm will typically quit after 50 unsuccessful attempts. When that occurs, it may be an indication the data should be fit with a different model.

Here is our data with the fitted prediction model:

```
test_data_soy = data.frame(dae=seq(0,120,1))
test_data_soy$pred = predict(soybean_tdm_model, test_data_soy)

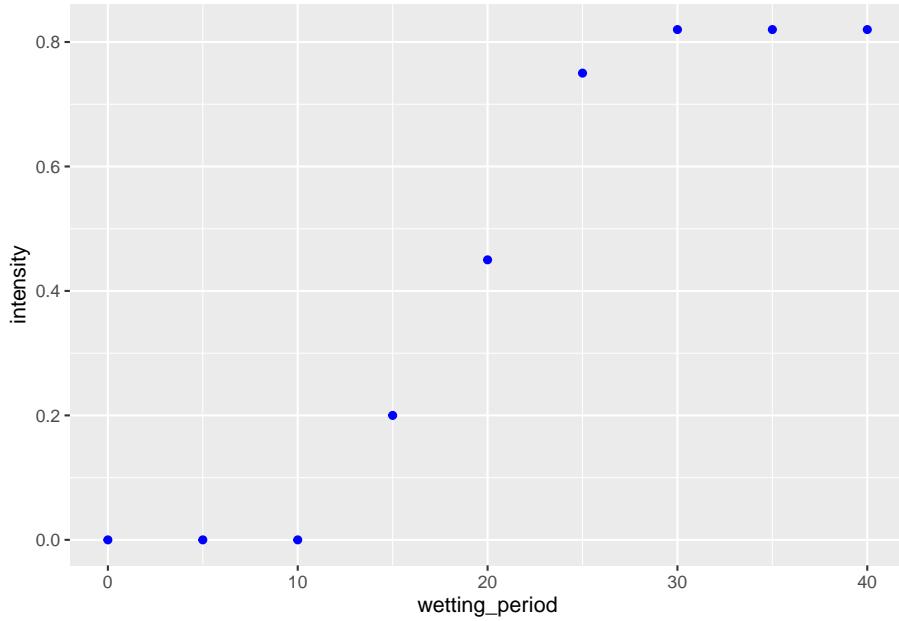
p_soy + geom_line(data = test_data_soy, aes(x=dae, y=pred), color="tomato")
```



We can see this again illustrated in the next plot, which illustrates the effect of wetting period on the intensity of wheat blast.

```
wheat_blast = read.csv("data-unit-11/wheat_blast_wetting_intensity.csv")
p_wheat = wheat_blast %>%
  ggplot(aes(x=wetting_period, y=intensity)) +
  geom_point(color="blue")

p_wheat
```



We fit the data with the logistic model.

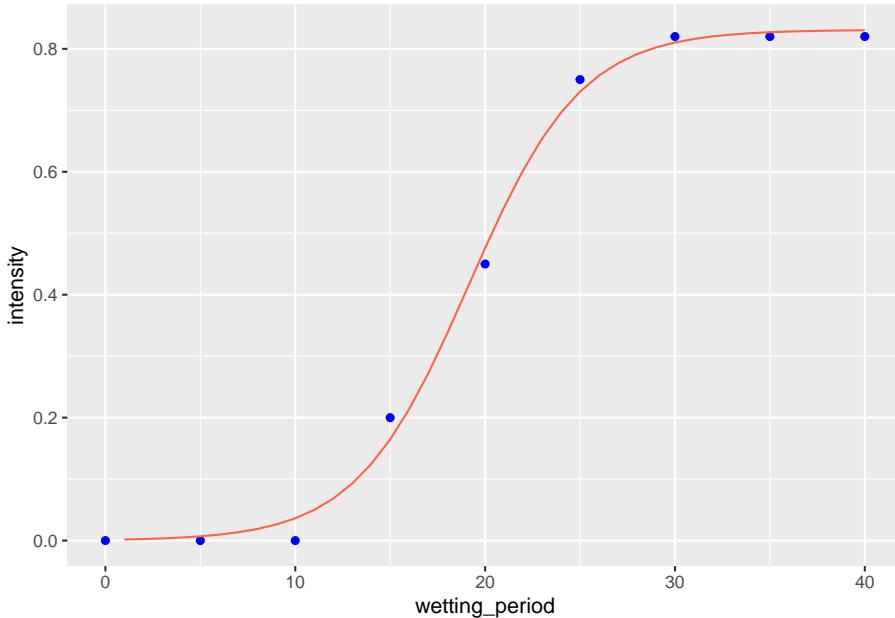
```
wheat_blast_model_logist = stats::nls(intensity ~ SSlogis(wetting_period, Asym, xmid, scal))
summary(wheat_blast_model_logist)
```

```
##
## Formula: intensity ~ SSlogis(wetting_period, Asym, xmid, scal)
##
## Parameters:
##   Estimate Std. Error t value Pr(>|t|)
##   Asym  0.83092   0.01606  51.74 3.50e-09 ***
##   xmid 19.13425   0.33505  57.11 1.94e-09 ***
##   scal  2.95761   0.28825  10.26 5.00e-05 ***
##   ---
##   Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02552 on 6 degrees of freedom
##
## Number of iterations to convergence: 1
## Achieved convergence tolerance: 4.024e-06
```

Convergence criteria was met. Here is the data plotted with the logistic model.

```
test_data_wheat_log = data.frame(wetting_period = seq(1,40,1))
test_data_wheat_log$pred = predict(wheat_blast_model_logist, test_data_wheat_log)

p_wheat + geom_line(data=test_data_wheat_log, aes(x=wetting_period, y=pred), color="tomato")
```



### 11.2.2.3 Starting Variable Values

As mentioned earlier, nonlinear regression algorithms can be difficult to use in that they don't fit a regression model using the least squares approach. Instead, nonlinear regression "nudges" the different coefficients in its equation until it zeros in on coefficient values that best fit the data. Often, traditionally, algorithms have not been able to solve for these coefficients from scratch. The user would have to provide initial coefficient values and hope they were close enough to true values that the nonlinear algorithm could then fit the curve.

What a pain in the butt. While some coefficients could be easily guessed, others required more complex calculations. With R, some packages have introduced "self-starting" non-linear models that do not require the user to enter initial values for coefficients. I have used those to create the examples above and we will cover them in the exercises. Just be aware, were you to get into more intensive nonlinear modelling, that you may need to specify those variables.

## 11.3 Summary

Multivariate models (those that model yield or another response to multiple variables) are very powerful tools for unlocking the mysteries of how changes in environment drive what we observe in our research. However, they require skill to use. Remember that more variables do not automatically make a better model. Too many variables can cause our model to overfit our original data, but be less accurate or unbiased in fitting future datasets. Ask yourself whether it makes sense to include each variable in a model. Covariance matrixes and partial correlations can help us identify predictor variables that may be correlated with each other instead of the response variable.

Cross validation is also an important tool in assessing how a model will work with future data. In this unit, we learned a common practice, 10-fold cross validation, in which the data were divided into 10 groups. Each group took turns being part of the datasets used to train the model, and the dataset used to test it.

Finally, nonlinear regression is used to fit variables that have a nonlinear relationship. Unlike multiple linear regression, there is usually only one predictor variable in non-linear regression. In addition, nonlinear regression is often based on a theoretical relationship between the predictor and response variable. In this unit, we focused on two models: the monomolecular model for relationships between yield and soil fertility, and the logistic model for predicting growth of plants and other organisms.

## 11.4 Exercise: Multiple Linear Regression

In this exercise, we will learn how to fit a multiple linear regression model to data.

### 11.4.1 Case Study: Country Corn Yields

We will use the same dataset as in the lesson. This time, we will model corn yield as a function of precipitation (ppt), water holding capacity (whc), sand, silt, clay, and organic matter (om) in Iowa and Illinois.

```
library(tidyverse)
county_corn_data = read.csv("data-unit-11/exercise_data/county_environments.csv")
head(county_corn_data)

##      county state stco      ppt      whc      sand      silt      clay          om
## 1 Abbeville    SC 45001 562.2991 22.94693 39.26054 21.38839 39.351063 36.92471
```

```

## 2 Acadia LA 22001 751.1478 37.83115 12.82896 54.19231 32.978734 105.79133
## 3 Accomack VA 51001 584.6807 18.56290 74.82042 16.82072 8.358863 103.11425
## 4 Ada ID 16001 124.9502 15.54825 44.49350 40.48000 15.026510 77.61499
## 5 Adair IA 19001 598.4478 28.91851 17.82011 48.44056 33.739326 259.53255
## 6 Adair KY 21001 698.6234 20.87349 15.66781 48.16733 36.164866 73.05356
##   kwfactor kffactor sph slope tfactor corn soybean cotton
## 1 0.2045719 0.2044325 5.413008 42.59177 4.666834 57.75000 16.75385 525.500
## 2 0.4432803 0.4431914 6.370533 34.92266 4.855202 92.42500 28.72857 NA
## 3 0.2044544 0.2044544 5.166446 NA 4.011007 116.99714 30.44857 575.125
## 4 0.4106704 0.4198545 7.648190 NA 2.853957 145.74138 NA NA
## 5 0.3541369 0.3541369 6.215139 55.24201 4.691089 133.02571 41.70857 NA
## 6 0.3166089 0.3725781 5.351832 37.65075 4.119289 95.76765 37.50000 NA
##   wheat
## 1 30.17500
## 2 38.11000
## 3 56.07407
## 4 97.26296
## 5 34.04000
## 6 37.42105

ia_and_il_corn = county_corn_data %>%
  filter(state %in% c("IL", "IA"))

head(ia_and_il_corn)

##   county state stco    ppt     whc     sand     silt     clay      om
## 1 Adair   IA 19001 598.4478 28.91851 17.820112 48.44056 33.73933 259.5326
## 2 Adams   IA 19003 609.1156 33.81156 11.794021 53.87736 34.32862 228.8244
## 3 Adams   IL 17001 600.4242 34.81655 9.794088 62.05820 28.14772 153.8438
## 4 Alexander IL 17003 640.1377 31.64141 24.959399 47.95812 27.08248 174.4598
## 5 Allamakee IA 19005 599.4642 28.65935 10.737119 60.69751 28.56537 125.8014
## 6 Appanoose IA 19007 659.3385 28.76599 15.346745 47.48802 37.16523 153.5939
##   kwfactor kffactor sph slope tfactor corn soybean cotton
## 1 0.3541369 0.3541369 6.215139 55.24201 4.691089 133.0257 41.70857 NA
## 2 0.3763315 0.3763315 6.460359 55.83899 4.653991 127.6057 40.22571 NA
## 3 0.4200895 0.4206594 6.286546 100.57306 4.704058 133.3571 40.19143 NA
## 4 0.3526438 0.3544891 6.503335 47.04578 4.543438 122.2438 33.19032 NA
## 5 0.4408712 0.4474461 5.993133 67.87513 4.400579 142.5800 43.90286 NA
## 6 0.3553925 0.3553940 6.132728 59.00196 4.025806 112.4057 36.14857 NA
##   wheat
## 1 34.04000
## 2 32.03333
## 3 52.77778
## 4 45.55556
## 5 36.50000
## 6 34.88333

```

### 11.4.2 Fitting the Model

Our model statement is the same as that used for single linear regression, except with multiple predictor variables.

```
ia_and_illinois_model = lm(corn ~ ppt + whc + sand + silt + clay + om, data = ia_and_i
```

### 11.4.3 Inspecting the Model

We will again use the *broom()* package and the *tidy()* function to inspect the model coefficients.

```
library(broom)
```

```
## Warning: package 'broom' was built under R version 4.1.3
tidy(ia_and_illinois_model)
```

```
## # A tibble: 7 x 5
##   term      estimate std.error statistic p.value
##   <chr>     <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept) 29262498.    1.95e+7     1.50  1.36e- 1
## 2 ppt        -0.258     2.71e-2    -9.49  8.69e-18
## 3 whc         0.581     2.30e-1     2.53  1.23e- 2
## 4 sand        -292622.    1.95e+5    -1.50  1.36e- 1
## 5 silt        -292622.    1.95e+5    -1.50  1.36e- 1
## 6 clay        -292623.    1.95e+5    -1.50  1.36e- 1
## 7 om          0.0205    9.73e-3     2.11  3.65e- 2
```

We notice that three of our predictor variables (sand, silt, clay) have p-values greater than 0.05. What is the overall performance of the model? Let's use the *glance()* function to find out.

```
glance(ia_and_illinois_model)
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic p.value    df logLik    AIC    BIC
##   <dbl>        <dbl>     <dbl>     <dbl>    <dbl> <dbl> <dbl> <dbl>
## 1 0.517       0.503    10.4     34.7  2.67e-28     6  -753. 1521. 1548.
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

The model has an r.squared of about 0.52 and is highly significant. In the next exercise. we will look more closely at our predictor variables and whether all should be included in our final model.

### 11.4.4 Practice: Thompson Corn Data

For the first practice, we will use a multi-year study of corn yield in several midwestern states. The study was conducted almost a century ago, so don't be surprised by the yields! Older data aside, this dataset is a good candidate for multiple linear regression.

To prevent this exercise from getting out of hand, I have reduced the predictor variables to four. In this practice, you will build the initial model and evaluate its coefficients and fit.

The predictor variables are total June precipitation in inches (rain6), total July precipitation (rain7), mean june temperature (temp6), and mean July temperature (June 7).

Build the initial model of yield as a function of rain6, rain7, temp6, and temp7.

```
thompson_corn = read.csv("data-unit-11/exercise_data/thompson_corn.csv")
head(thompson_corn)

##   yield rain6 rain7 temp6 temp7
## 1 26.5  3.36  1.01  71.9  79.1
## 2 37.0  3.19  2.97  75.5  79.3
## 3 43.0  4.08  3.32  74.0  77.6
## 4 27.0  1.48  2.41  78.2  78.1
## 5 21.5  3.01  3.24  79.0  81.7
## 6 38.5  5.99  3.33  68.2  79.0

thompsom_model = lm(yield ~ temp6 + rain6 + rain7 + temp7, thompson_corn)
```

Use the tidy() function to review the coefficient values. Your estimates for rain7 and temp7 should be about 4.00 and 2.31. Which coefficients are significant? Insignificant?

```
library(broom)

tidy(thompsom_model)

## # A tibble: 5 x 5
##   term      estimate std.error statistic p.value
##   <chr>      <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept) 235.       25.7      9.14  2.62e-16
## 2 temp6      -0.281     0.303     -0.928 3.55e- 1
## 3 rain6       -1.39      0.501     -2.78  6.05e- 3
## 4 rain7        4.00      0.588      6.81  1.88e-10
## 5 temp7      -2.31      0.316     -7.29  1.32e-11
```

Examine the model fit using the `glance()` function. You should observe an `r.squared` of 0.55. How well does the model fit this data?

```
glance(thompson_model)
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic p.value    df logLik    AIC    BIC
##       <dbl>         <dbl> <dbl>     <dbl>    <dbl> <dbl> <dbl> <dbl> <dbl>
## 1     0.547        0.535  9.92     48.2  1.47e-26     4 -610. 1232. 1251.
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

## 11.5 Exercise: Avoiding Bloated Models

In the last exercise, we learned how to create a multiple linear regression model with R. It was a relatively simple process: construct the model using the `lm()` function, then review the coefficients and fit using the `tidy()` and `glance()` functions from the `broom` package.

It is easy to create models. Refining them can take a little soul-searching. In this exercise, we will learn how to construct a covariance matrix to visually inspect for relationships among predictor variables that may weaken our model, calculate partial correlations to quantify those relationships, and how to cross-validate our model to evaluate whether we might be over-fitting our data.

### 11.5.1 Case Study

We will continue with the model we generated in the previous exercise.

```
library(tidyverse)
county_corn_data = read.csv("data-unit-11/exercise_data/county_environments.csv")

ia_and_il_corn = county_corn_data %>%
  filter(state %in% c("IL", "IA"))

ia_and_illinois_model = lm(corn ~ ppt + whc + sand + silt + clay + om, data = ia_and_il_corn)
```

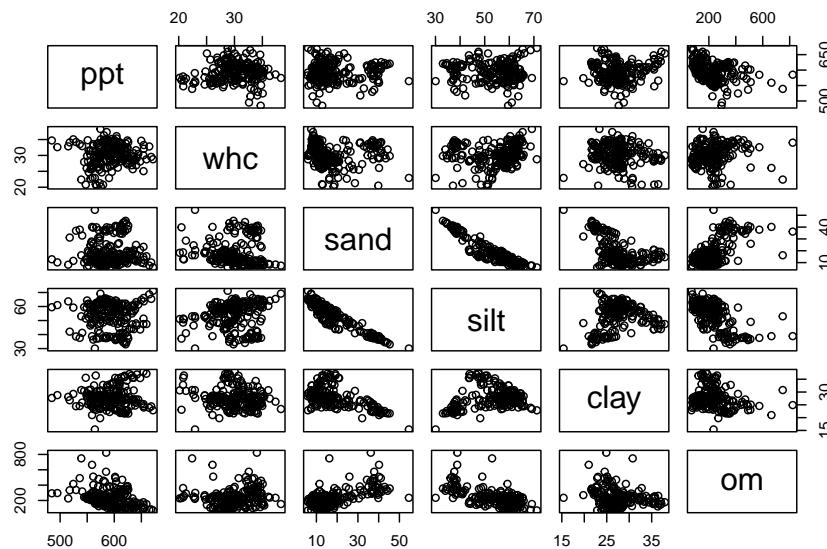
### 11.5.2 Covariance Matrix

In the last unit, we were introduced to the covariariance matrix, sometimes also called the correlation matrix. We will generate that again, using the `plot()` function. First, however, we will want to narrow down the predictor variables to those of interest. We can use the `select()` function to drop the other variables.

```
variables_of_interest = ia_and_il_corn %>%
  dplyr::select(ppt, whc, sand, silt, clay, om)
```

We can now plot our covariance matrix.

```
plot(variables_of_interest)
```



Right away, we notice a strong relationship between sand and silt. Sand might also be associated with organic matter. Other potential relationships include precipitation and organic matter, or clay and sand. Next, we will use partial correlations to quantify the strength of correlations between predictor variables.

### 11.5.3 Partial Correlation

Partial correlation measure the correlation between two variables while holding the other variables in the dataset constant. In other words, partial correlation measures the true relationship between the two variables, separate from any effect a third variable may be having on them.

We can generate a matrix with partial correlations using the *psych* package and the *partial.r()* function.

```

library(psych)

partial.r(variables_of_interest)

## In smc, smcs < 0 were set to .0

##          ppt      whc      sand      silt      clay       om
## ppt    1.0000000  0.2274461  0.24731123 -0.3552027  0.15604023 -0.5020622
## whc    0.2274461  1.0000000 -0.17263293  0.2887891 -0.20460027  0.3015432
## sand   0.2473112 -0.1726329  1.00000000  0.5458297 -0.06206651  0.4737700
## silt   -0.3552027  0.2887891  0.54582965  1.0000000  0.49931202 -0.6094931
## clay   0.1560402 -0.2046003 -0.06206651  0.4993120  1.00000000  0.1326898
## om     -0.5020622  0.3015432  0.47376997 -0.6094931  0.13268983  1.0000000

```

The first two tables, \$estimate and \$p.value, are the most interesting. Above we noticed a strong relationship between sand and silt and clay. Looking at the \$estimate table, we see their partial correlations are each 1.0. If we look at the \$p.value table, we see those correlations are also highly significant. All three are also weakly correlated with precipitation and water holding capacity. We will need to look at each of these three variables more closely when we go to tune our model.

#### 11.5.4 Cross-Validation

In cross validation, we divide our data into two groups. Most of the data are used to fit or “train” the model. The remainder of the data are used to “test” how well the model performs by comparing its predictions to the actual observed response of the dependent variable (Y).

We use the *caret* package to cross-validate our data. Specifically, we use a process called k-fold cross validation. In k-fold cross-validation, the data are divided into 10 groups. 9 groups are used to train the model, and the remaining group used for the comparison between predicted and observed values. In the code below, we use 10-fold cross-validation.

The code below is basically plug-and-play. Everything from library(caret) through the train.control statement can be left as-is. The one line that needs to be modified is the train() function. Replace the terms in brackets below with your desired name for the output, your linear model, and your data frame.

```
[your model name] = train([your linear model], data=[your data frame], method = "lm", trControl = train.control)
```

```

library(caret)

# Leave this as-is -- tells R to conduct 10-fold cross validation
set.seed(123)
train.control <- trainControl(method = "repeatedcv",
                               number = 10, repeats = 3)

# Modify this with your model and dataset
model <- train(corn ~ ppt + whc + sand + silt + clay + om, data=ia_and_il_corn, method = "lm", tr
# Summarize the results
print(model)

## Linear Regression
##
## 201 samples
##   6 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 181, 181, 181, 181, 181, 180, ...
## Resampling results:
##
##   RMSE     Rsquared    MAE
##   10.57434  0.5295991  8.253716
##
## Tuning parameter 'intercept' was held constant at a value of TRUE

```

Our model has an Rsquared of about 0.53 and a root mean-square error of 10.6. In our next model, we will tune our model by subtracting variables to see whether model performance might be improved with fewer predictor variables.

### 11.5.5 Practice: Thompson Corn Data

In this practice, let's take our corn model from the last exercise and inspect it for evidence of collinearity.

```

thompson_corn = read.csv("data-unit-11/exercise_data/thompson_corn.csv")

thompson_model = lm(yield ~ rain6 + rain7 + temp6 + temp7, thompson_corn)

```

Plot the covariance matrix. Do you notice any predictor variables that appear correlated?

Next, plot the partial correlation matrix. Are any of the predictor variables significantly correlated with each other?

Finally, cross-validate the model by finishing the code below. Your cross-validated Rsquared should be about 0.537.

```
library(caret)

# Define training control
set.seed(123)
train.control <- trainControl(method = "repeatedcv",
                               number = 10, repeats = 3)
```

## 11.6 Exercise: Tuning the Model

At this point, we have built the multiple regression model and examined the significance of each predictor variable. We have also identified correlations between individuals. We have used cross-validation to measure the model's performance on original data, in order to guard against overfitting the data.

Our final phase of model development is to tune the model. In tuning the model, we test the effect of dropping out variables we think are collinear – that is, correlated with each other instead of being independent predictors of yield.

### 11.6.1 Case Study

We continue to work with the model we built in the other exercises. Lets start by calling up some of our data from the last exercise.

First, our model:

```
library(tidyverse)
county_corn_data = read.csv("data-unit-11/exercise_data/county_environments.csv")

ia_and_il_corn = county_corn_data %>%
  filter(state %in% c("IL", "IA"))

ia_and_illinois_model = lm(corn ~ ppt + whc + sand + silt + clay + om, data = ia_and_il_corn)
```

Then we define our variables of interest.

```
variables_of_interest = ia_and_il_corn %>%
  dplyr::select(ppt, whc, sand, silt, clay, om)
```

Next, the partial correlation matrix:

```
library(psych)
partial.r(variables_of_interest)

## In smc, smcs < 0 were set to .0

##      ppt      whc      sand      silt      clay      om
## ppt  1.000000  0.2274461  0.24731123 -0.3552027  0.15604023 -0.5020622
## whc  0.2274461  1.0000000 -0.17263293  0.2887891 -0.20460027  0.3015432
## sand 0.2473112 -0.1726329  1.00000000  0.5458297 -0.06206651  0.4737700
## silt -0.3552027  0.2887891  0.54582965  1.0000000  0.49931202 -0.6094931
## clay  0.1560402 -0.2046003 -0.06206651  0.4993120  1.00000000  0.1326898
## om   -0.5020622  0.3015432  0.47376997 -0.6094931  0.13268983  1.0000000
```

And, finally, for reference, the cross-validation:

```
library(caret)

# Define training control
set.seed(123)
train.control <- trainControl(method = "repeatedcv",
                               number = 10, repeats = 3)
# Train the model
model <- train(corn ~ ppt + whc + sand + silt + clay + om, data=ia_and_il_corn, method = "lm",
                trControl = train.control)
# Summarize the results
print(model)

## Linear Regression
##
## 201 samples
##   6 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 181, 181, 181, 181, 181, 180, ...
## Resampling results:
##
##   RMSE      Rsquared     MAE
##   10.57434  0.5295991  8.253716
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

### 11.6.2 Tuning the Model

In this example, choosing the first term to drop from our model is difficult, given that sand, silt, and clay are similarly correlated with each other. After reviewing the covariance matrix from the last exercise, however, let's go ahead and drop sand from the sample:

```
model_wo_sand = lm(corn ~ ppt + whc + silt + clay + om, data = ia_and_il_corn)
```

How are the remaining coefficients affected?

```
library(broom)
```

```
tidy(model_wo_sand)
```

term	estimate	std.error	statistic	p.value
(Intercept)	290.	20.4	14.2	5.08e-32
ppt	-0.254	0.0271	-9.37	1.88e-17
whc	0.614	0.230	2.67	8.24e- 3
silt	-0.135	0.114	-1.18	2.39e- 1
clay	-0.699	0.201	-3.47	6.35e- 4
om	0.0225	0.00968	2.32	2.12e- 2

We can see the slope (estimate) for clay is now significant. Let's check our fit.

```
glance(model_wo_sand)
```

	r.squared	adj.r.squared	sigma	statistic	p.value	df	logLik	AIC	BIC
1	0.512	0.499	10.4	40.9	1.18e-28	5	-754.	1522.	1545.

Our r.squared and adj.r.squared have decreased slightly. Now lets cross-validate our model.

```
# Define training control
set.seed(123)
train.control <- trainControl(method = "repeatedcv",
                               number = 10, repeats = 3)
# Train the model
```

```

model <- train(corn ~ ppt + whc + silt + clay + om, data=ia_and_il_corn, method = "lm",
                trControl = train.control)
# Summarize the results
print(model)

## Linear Regression
##
## 201 samples
##   5 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 181, 181, 181, 181, 181, 180, ...
## Resampling results:
##
##   RMSE      Rsquared     MAE
##   10.63189  0.5282656  8.294276
##
## Tuning parameter 'intercept' was held constant at a value of TRUE

```

Our Rsquared has decreased ever so slightly from our first model.

Now, let's repeat this process, dropping both sand and silt from the model.

```
model_wo_sand_silt = lm(corn ~ ppt + whc + clay + om, data = ia_and_il_corn)
```

How are the remaining coefficients affected?

```
tidy(model_wo_sand_silt)
```

```

## # A tibble: 5 x 5
##   term      estimate std.error statistic p.value
##   <chr>      <dbl>     <dbl>      <dbl>    <dbl>
## 1 (Intercept) 277.      17.2       16.1  7.85e-38
## 2 ppt        -0.242     0.0250     -9.66 2.59e-18
## 3 whc         0.532      0.219      2.42  1.63e- 2
## 4 clay        -0.738     0.199     -3.71 2.65e- 4
## 5 om          0.0305    0.00693     4.40  1.81e- 5

```

All of the remaining slopes are now significant. Next, our model fit:

```
glance(model_wo_sand_silt)
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic p.value    df logLik    AIC    BIC
##       <dbl>        <dbl> <dbl>     <dbl> <dbl>    <dbl> <dbl> <dbl>
## 1     0.508      0.498  10.5     50.7 3.07e-29      4 -754. 1521. 1541.
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

Our r.squared and adj.r.squared have again decreased slightly. Now lets cross-validate our model.

```
# Define training control
set.seed(123)
train.control <- trainControl(method = "repeatedcv",
                               number = 10, repeats = 3)
# Train the model
model <- train(corn ~ ppt + whc + clay + om, data=ia_and_il_corn, method = "lm", trControl = train.control)
# Summarize the results
print(model)

## Linear Regression
##
## 201 samples
##   4 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 181, 181, 181, 181, 181, 180, ...
## Resampling results:
##
##   RMSE    Rsquared    MAE
##   10.5296  0.5377286  8.256467
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

Yes! We see that our Rsquared is greater than the original model. Our RMSE (Root Mean Square Error) is also the lowest among the models.

We could continue this process, perhaps testing what would happen if water-holding capacity (whc) were removed from the model. But at this point we can see how tuning can reduce the bloat and complexity of our model. While it may always seem the more data the better, we can now see that is not the case. From a practical point of view, also, we now have fewer variables to measure and less data to store – two considerations that can become substantial with larger projects.

### 11.6.3 Practice

```

thompson_corn = read.csv("data-unit-11/exercise_data/thompson_corn.csv")
thompsom_model = lm(yield ~ rain6 + rain7 + temp6 + temp7, thompson_corn)

tidy(thompsom_model)

## # A tibble: 5 x 5
##   term      estimate std.error statistic p.value
##   <chr>     <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept) 235.      25.7      9.14  2.62e-16
## 2 rain6       -1.39     0.501    -2.78  6.05e- 3
## 3 rain7        4.00     0.588     6.81  1.88e-10
## 4 temp6       -0.281    0.303    -0.928 3.55e- 1
## 5 temp7       -2.31     0.316    -7.29  1.32e-11

glance(thompsom_model)

## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic p.value    df logLik    AIC    BIC
##   <dbl>        <dbl>     <dbl>     <dbl>    <dbl> <dbl> <dbl> <dbl>
## 1 0.547        0.535    9.92     48.2  1.47e-26     4 -610. 1232. 1251.
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>

library(caret)

# Define training control
set.seed(123)
train.control <- trainControl(method = "repeatedcv",
                               number = 10, repeats = 3)

# Train the model
model <- train(yield ~ rain6 + rain7 + temp6 + temp7, data=thompson_corn, method = "lm", trControl = train.control)

# Summarize the results
print(model)

## Linear Regression
##
## 165 samples
##   4 predictor
##
## No pre-processing

```

## 416 CHAPTER 11. NONLINEAR RELATIONSHIPS AND MULTIPLE LINEAR REGRESSION

```

## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 148, 149, 147, 149, 148, 149, ...
## Resampling results:
##
##   RMSE      Rsquared     MAE
##   9.981899  0.5370066  8.258314
##
## Tuning parameter 'intercept' was held constant at a value of TRUE

thompson_model_wo_temp6 = lm(yield ~ rain6 + rain7 + temp7, thompson_corn)

tidy(thompson_model_wo_temp6)

## # A tibble: 4 x 5
##   term       estimate std.error statistic p.value
##   <chr>     <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept) 224.      22.8      9.83 3.70e-18
## 2 rain6       -1.23     0.470     -2.62 9.53e- 3
## 3 rain7        4.04     0.586      6.89 1.17e-10
## 4 temp7       -2.44     0.282     -8.65 4.93e-15

glance(thompson_model_wo_temp6)

## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic p.value    df logLik    AIC    BIC
##   <dbl>        <dbl> <dbl>     <dbl>    <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 0.544        0.536  9.92     64.1 2.57e-27     3 -611. 1231. 1247.
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>

library(caret)

# Define training control
set.seed(123)
train.control <- trainControl(method = "repeatedcv",
                               number = 10, repeats = 3)
# Train the model
model <- train(yield ~ rain6 + rain7 + temp7, data=thompson_corn, method = "lm", trControl = train.control)
# Summarize the results
print(model)

## Linear Regression
##
## 165 samples

```

```

## 3 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 148, 149, 147, 149, 148, 149, ...
## Resampling results:
##
##   RMSE      Rsquared    MAE
##   9.942276  0.538738  8.273144
##
## Tuning parameter 'intercept' was held constant at a value of TRUE

```

#### 11.6.4 Practice

Let's synthesize what you found in the first two exercises.

First, looking at the original model output, we see one of the predictor variables did not have a coefficient whose value was significantly different from zero: temp6.

Second, looking at the covariance matrix and partial correlation matrix, we note two strong correlations between the predictor variables: temp6 with temp7, and temp6 with rain6.

Both of these suggest that temp6 may not belong in our final model.

With this in mind, create a new model of yield as a function of rain6, rain7, and temp7.

```
thompson_model_wo_temp6 = lm(yield ~ rain6 + rain7 + temp7, thompson_corn)
```

Lets see the effect on our coefficients.

```
tidy(thompson_model_wo_temp6)
```

```

## # A tibble: 4 x 5
##   term       estimate std.error statistic p.value
##   <chr>     <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept) 224.      22.8      9.83 3.70e-18
## 2 rain6      -1.23     0.470     -2.62 9.53e- 3
## 3 rain7       4.04     0.586      6.89 1.17e-10
## 4 temp7      -2.44     0.282     -8.65 4.93e-15

```

You should find all the remaining predictors have slopes significantly different than zero.

What about the effect on model fit?

```
glance(thompson_model_wo_temp6)
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic p.value    df logLik    AIC    BIC
##       <dbl>         <dbl> <dbl>     <dbl>    <dbl> <dbl> <dbl> <dbl>
## 1     0.544        0.536  9.92     64.1 2.57e-27     3 -611. 1231. 1247.
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

You should see an r.squared of 0.54. That is a slight decrease from our original model.

But what about the cross-validated model? What is the effect on its Rsquared?

```
library(caret)

# Define training control
set.seed(123)
train.control <- trainControl(method = "repeatedcv",
                               number = 10, repeats = 3)
# Train the model
model <- train(yield ~ rain6 + rain7 + temp7, data=thompson_corn, method = "lm", trControl = train.control)
# Summarize the results
print(model)
```

```
## Linear Regression
##
## 165 samples
## 3 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 148, 149, 147, 149, 148, 149, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   9.942276  0.538738  8.273144
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

You should see an Rsquared of 0.538. This suggests that we were correct in removing temp6 from our model.

## 11.7 Exercise: Nonlinear Regression

In the other exercises, we focused on multiple linear regression, i.e. using a linear model with multiple predictor variables to explain a complex response. In this exercise, we will learn to fit data with a nonlinear model. In our nonlinear model, there is only one predictor variable, but the shape of the model is determined by multiple coefficients, which are fit using nonlinear regression

### 11.7.1 Case Study 1: Monomolecular Data

In this example, we will look at the response of corn silage yield to plant population. We will fit this with a monomolecular, or asymptotic curve. In this type of curve, the data rise quickly as X increases, but then slope decreases, approaching zero, and the curve plateaus. This model is often used to fit responses to fertilizer (provided excess rates of the fertilizer are not toxic). It is also used to describe the yield responses to plant population of many crops, including silage, where crops are grown more for biomass than grain.

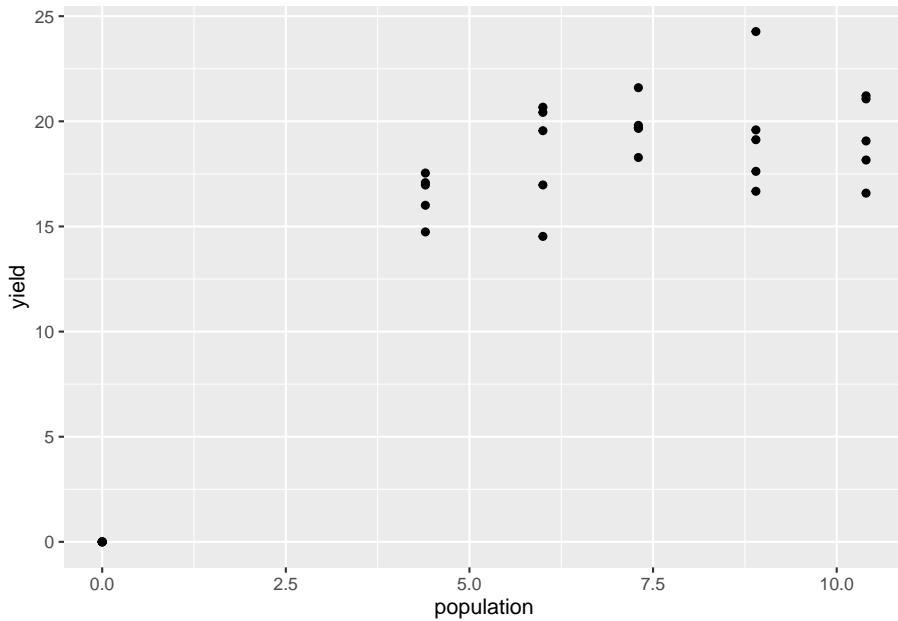
```
library(tidyverse)
silage = read.csv("data-unit-11/exercise_data/corn_silage_pop_mono.csv")
head(silage)

##   block population     yield
## 1      1       0.0  0.00000
## 2      1       4.4 16.96924
## 3      1       6.0 16.97407
## 4      1       7.3 21.59875
## 5      1       8.9 24.26829
## 6      1      10.4 21.21362
```

First, let's plot our data. We are going to take the unusual step of assigning our plot to an r object, p. This stores all the instructions for making the plot, kind of like we store data in a data.frame. That way, we can easily add our nonlinear regression line to this plot later.

```
p = silage %>%
  ggplot(aes(x=population, y=yield)) +
  geom_point()

p
```



Now, let's fit our nonlinear model. The next line of code is particularly ugly, but you can largely plug-and-play with it.

```
[your object name] = stats::nls([your response variable] ~ SSasymp([your predictor variable],init,m,plateau), data=[your data frame])
```

Change these items [your object name]: change to whatever object name you would like. [your response variable]: change to whatever variable you are measuring the response in. In this case study, it is yield. [your predictor variable]: change to whatever variable you are using to predict the response. In this case study, it is population. [your data frame]: wherever your data ist stored. In this case, the silage data frame.

```
silage_mono = stats::nls(yield ~ SSasymp(population,init,m,plateau), data=silage)

summary(silage_mono)

##
## Formula: yield ~ SSasymp(population, init, m, plateau)
##
## Parameters:
##             Estimate Std. Error t value Pr(>|t|)
## init      19.98096   0.76441  26.139 < 2e-16 ***
## m        -0.01673   0.82940  -0.020 0.984059
## plateau -0.87295   0.20178  -4.326 0.000186 ***
## ---
##
```

```

## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.855 on 27 degrees of freedom
##
## Number of iterations to convergence: 0
## Achieved convergence tolerance: 2.818e-07

```

Most of the information in the mode summary can be, for our purposes, ignored. The important part of this output is the bottom line, “Achieved convergence tolerance”. That means our model successfully fit the data.

We can now create a new data frame with predicted yields for populations from 0 to 10.4. We are going to start by building a data frame with populations from 0 to 10.4. We will use the `seq()` command to build our sequence of values for population.

```

silage_predicted = data.frame(
  population =
    seq(from=0,to=10.4,by=0.1)
)

silage_predicted

```

## population	# this tells R to create a new data frame called # this tells R to create a new column named "n_r # this creates a sequence of numbers from 0 to 10.4
## 1 0.0	
## 2 0.1	
## 3 0.2	
## 4 0.3	
## 5 0.4	
## 6 0.5	
## 7 0.6	
## 8 0.7	
## 9 0.8	
## 10 0.9	
## 11 1.0	
## 12 1.1	
## 13 1.2	
## 14 1.3	
## 15 1.4	
## 16 1.5	
## 17 1.6	
## 18 1.7	
## 19 1.8	
## 20 1.9	
## 21 2.0	

## 422CHAPTER 11. NONLINEAR RELATIONSHIPS AND MULTIPLE LINEAR REGRESSION

```
## 22      2.1
## 23      2.2
## 24      2.3
## 25      2.4
## 26      2.5
## 27      2.6
## 28      2.7
## 29      2.8
## 30      2.9
## 31      3.0
## 32      3.1
## 33      3.2
## 34      3.3
## 35      3.4
## 36      3.5
## 37      3.6
## 38      3.7
## 39      3.8
## 40      3.9
## 41      4.0
## 42      4.1
## 43      4.2
## 44      4.3
## 45      4.4
## 46      4.5
## 47      4.6
## 48      4.7
## 49      4.8
## 50      4.9
## 51      5.0
## 52      5.1
## 53      5.2
## 54      5.3
## 55      5.4
## 56      5.5
## 57      5.6
## 58      5.7
## 59      5.8
## 60      5.9
## 61      6.0
## 62      6.1
## 63      6.2
## 64      6.3
## 65      6.4
## 66      6.5
## 67      6.6
```

```
## 68      6.7
## 69      6.8
## 70      6.9
## 71      7.0
## 72      7.1
## 73      7.2
## 74      7.3
## 75      7.4
## 76      7.5
## 77      7.6
## 78      7.7
## 79      7.8
## 80      7.9
## 81      8.0
## 82      8.1
## 83      8.2
## 84      8.3
## 85      8.4
## 86      8.5
## 87      8.6
## 88      8.7
## 89      8.8
## 90      8.9
## 91      9.0
## 92      9.1
## 93      9.2
## 94      9.3
## 95      9.4
## 96      9.5
## 97      9.6
## 98      9.7
## 99      9.8
## 100     9.9
## 101    10.0
## 102    10.1
## 103    10.2
## 104    10.3
## 105    10.4
```

We see the first 10 rows of our new data frame above. Our next step is to use the `predict()` to create a new column in our data frame. This column will have the predicted yield for each value of population.

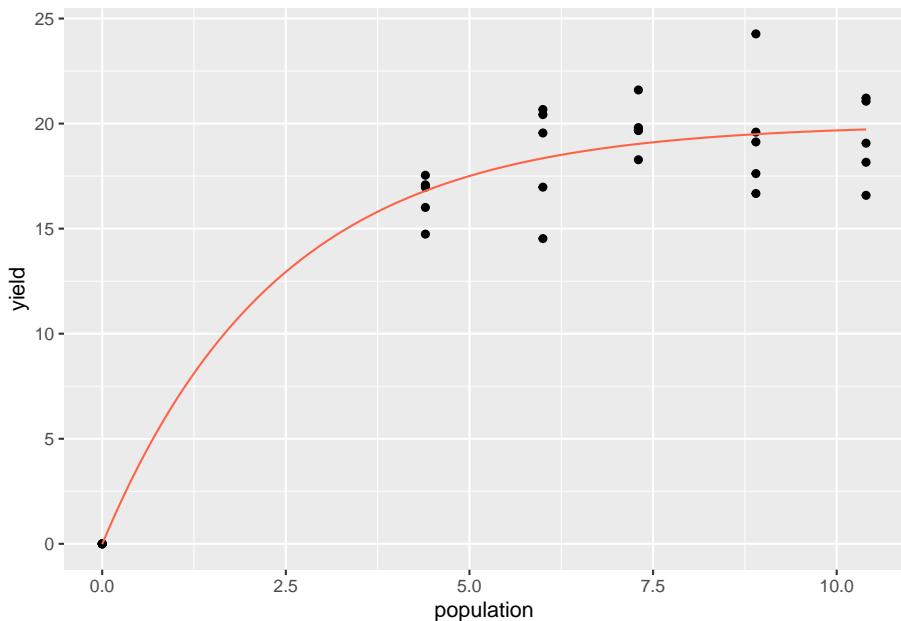
```
silage_predicted$yield = predict(silage_mono, silage_predicted)
```

Finally, we can add our modelled curve to our initial plot. This is where we

reuse our plot earlier, which we saved as `p`. We can now add a new geom to `p` just like we would to a plot if we were creating it the first time around.

`geom_line()` takes three arguments: the name of the data frame, and `aes()` argument with the `x` and `y` coordinates of the predicted values and, finally, a color argument so that our predicted values are easily distinguished from our observed values.

```
p +
  geom_line(data = silage_predicted, aes(x=population, y=yield), color="tomato")
```



### 11.7.2 Case Study 2: Logistic Data

The logistic curve is often used in agronomy to model the absolute growth of plants over time. A seedling starts out small – even if it grows rapidly relative to its initial size, the increments of growth will be small on an absolute scale. It is the same concept as compound interest. At first, biomass accumulation seems small. After a certain amount of time, however, the seedling establishes a leaf area and root mass that supports rapid, linear growth. This rapid vegetative growth, however, is followed by a transition to seed production, in which growth slows, and ultimately plateaus as seed fill completes.

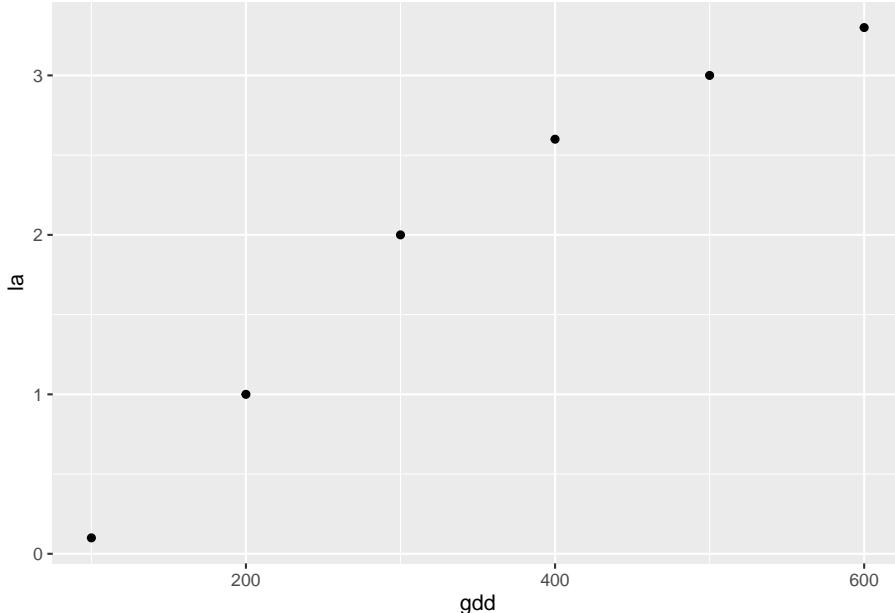
```
velvetleaf = read.csv("data-unit-11/exercise_data/velveleaf_gdd_lai_gomp.csv")
head(velvetleaf)
```

```
##   X gdd  la
## 1 1 100 0.1
## 2 2 200 1.0
## 3 3 300 2.0
## 4 4 400 2.6
## 5 5 500 3.0
## 6 6 600 3.3
```

First, let's plot the original data.

```
pV = velvetleaf %>%
  ggplot(aes(x=gdd, y=la)) +
  geom_point()

pV
```



Next, we need to fit our nonlinear function to the data. Similar to the monomolecular function, this is an ugly model, but all you need to know is how to switch out variables so it can fit your dataset.

```
[your object name] = stats::nls([your response variable] ~ SSlogis([your predictor variable], Asym, xmid, scal), data=[your data frame])
```

Change these items [your object name]: change to whatever object name you would like. [your response variable]: change to whatever variable you are measuring the response in. In this case study, it is yield. [your predictor variable]:

change to whatever variable you are using to predict the response. In this case study, it is population. [your data frame]: wherever your data ist stored. In this case, the silage data frame.

Below, we will plug in our velvetleaf data

```
velvetleaf_log = stats::nls(la ~ SSlogis(gdd, Asym, xmid, scal), data=velvetleaf)

summary(velvetleaf_log)

##
## Formula: la ~ SSlogis(gdd, Asym, xmid, scal)
##
## Parameters:
##             Estimate Std. Error t value Pr(>|t|)
## Asym     3.2130    0.1626 19.756 0.000283 ***
## xmid   269.6640    16.1602 16.687 0.000469 ***
## scal    75.3642    13.7009  5.501 0.011826 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.176 on 3 degrees of freedom
##
## Number of iterations to convergence: 0
## Achieved convergence tolerance: 7.224e-06
```

To construct our curve, we again need a dataset with an appropriate sequence of values for gdd. In this case, we create a range of values from 100 to 600, in increments of 10.

```
velvetleaf_predicted = data.frame(
  gdd =
    seq(from=100,to=600,by=10) # this tells R to create a new data frame
) # this tells R to create a new column
# this creates a sequence of numbers from 100 to 600 in increments of 10

velvetleaf_predicted

##      gdd
## 1    100
## 2    110
## 3    120
## 4    130
## 5    140
## 6    150
## 7    160
```

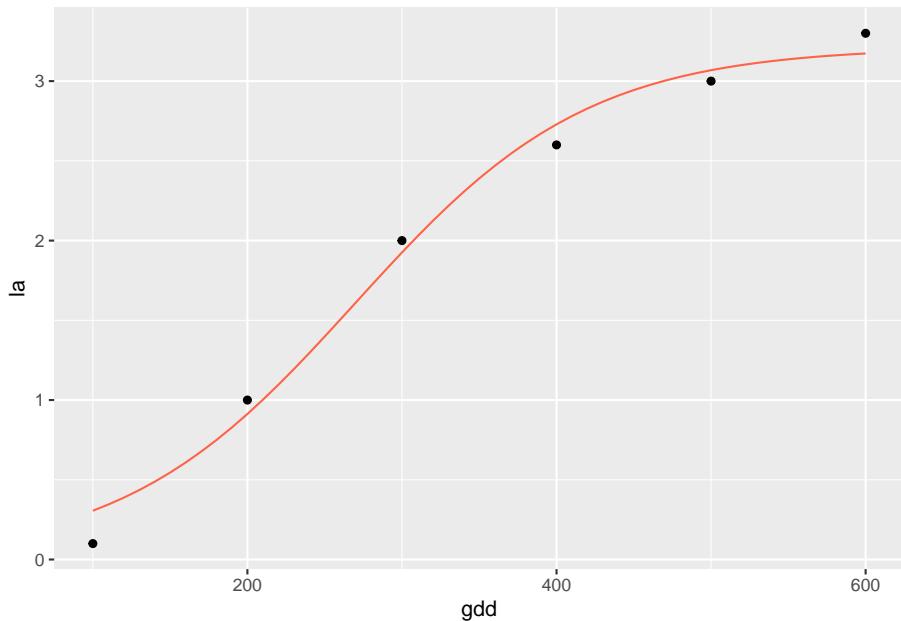
```
## 8 170
## 9 180
## 10 190
## 11 200
## 12 210
## 13 220
## 14 230
## 15 240
## 16 250
## 17 260
## 18 270
## 19 280
## 20 290
## 21 300
## 22 310
## 23 320
## 24 330
## 25 340
## 26 350
## 27 360
## 28 370
## 29 380
## 30 390
## 31 400
## 32 410
## 33 420
## 34 430
## 35 440
## 36 450
## 37 460
## 38 470
## 39 480
## 40 490
## 41 500
## 42 510
## 43 520
## 44 530
## 45 540
## 46 550
## 47 560
## 48 570
## 49 580
## 50 590
## 51 600
```

We use the *predict()* function the same as we did above.

```
velvetleaf_predicted$la = predict(velvetleaf_log, velvetleaf_predicted)
```

Finally, we can add our modelled curve to our initial plot, pV

```
pV +
  geom_line(data = velvetleaf_predicted, aes(x=gdd, y=la), color="tomato")
```



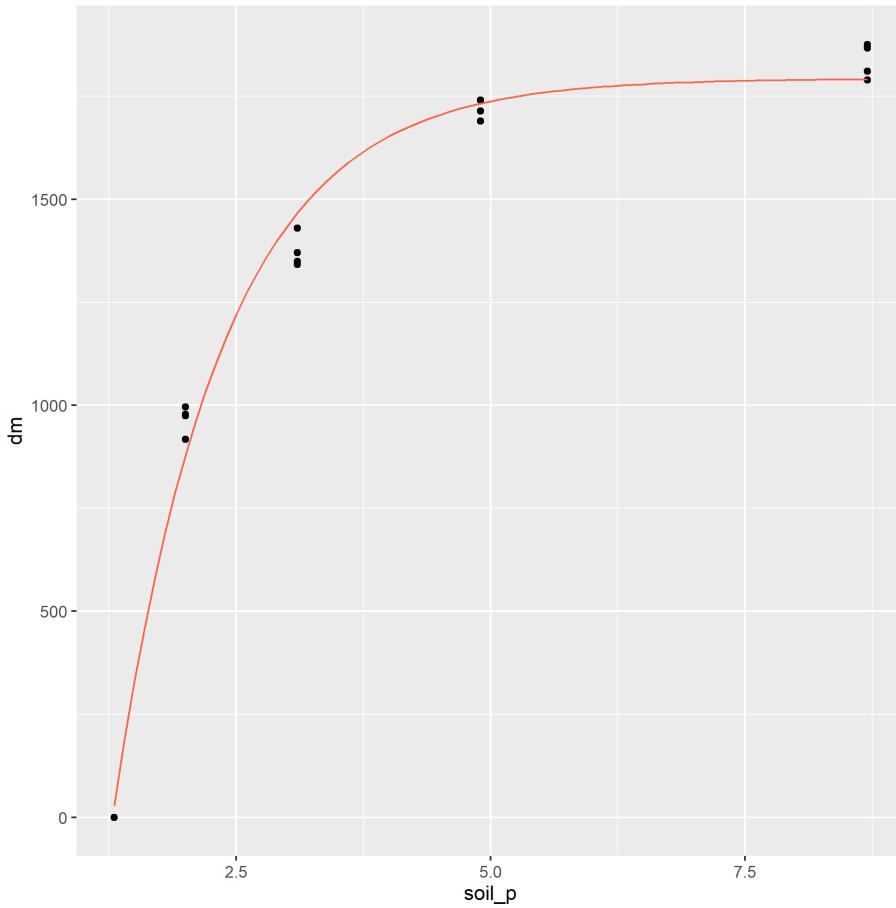
### 11.7.3 Practice 1

In the following study, we will model ryegrass dry matter (dm) response to soil phosphorus test levels.

```
ryegrass = read.csv("data-unit-11/exercise_data/ryegrass_soil_p_mono.csv")
head(ryegrass)
```

```
##   block soil_p      dm
## 1     1    1.3  0.0000
## 2     1    2.0 975.0045
## 3     1    3.1 1370.8090
## 4     1    4.9 1714.5594
## 5     1    8.7 1789.6946
## 6     2    1.3  0.0000
```

1. Create a scatterplot of the initial ryegrass data. Assign it to an object (for example, “ryegrass\_plot”) so you can add the regression line to it later.
2. Now fit the monomolecular model to the data. Print the summary. You should see an archived convergence tolerance of 2.535e-06.
3. Next, create a new data frame with soil\_p values from 1.3 to 8.7, by 0.1.
4. Use the *predict()* function to create a new column with the predicted dm for each value of soil\_p.
5. Add the modelled curve to your initial plot. Your final plot should look like:



### 11.7.4 Practice 2

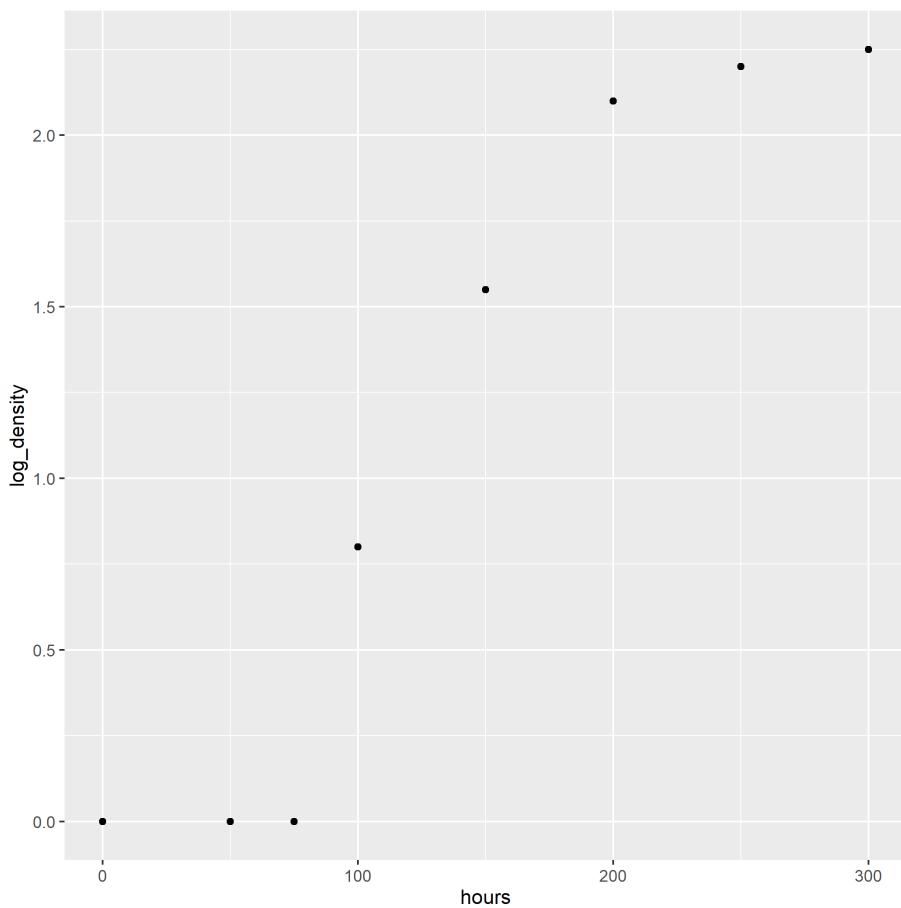
In this practice, we will fit a logistic curve to the growth of aspergillus, a soil bacteria.

```
aspergillus = read.csv("data-unit-11/exercise_data/aspergillus_hours_density_logist.csv")
head(aspergillus)
```

```
##   hours log_density
## 1     0      0.00
## 2    50      0.00
## 3    75      0.00
## 4   100      0.80
## 5   150      1.55
## 6   200      2.10
```

1. Plot the original data.
2. Fit the logistic model to the data.
3. Create a new dataset with hours from 0 to 300 in increments of 10.
4. Use the predict() function to predict the values of the logistic growth curve.
5. Finally, add the modelled curve to the initial plot.

Your plot should look like:





# Chapter 12

## Spatial Statistics

One of the most powerful ways I use and present data is to explain spatial patterns in our data. How does a product perform in Ohio versus Iowa? What might be the underlying weather or soil causes of these data? How do they vary with geography?

Quantitative data and hard numbers are fantastic. But as we have already seen with plots, visualizations can be much more engaging. Our minds are evolved to recognize patterns – in fact, we are so focused on looking for patterns that we need tools like statistics to keep us honest. So a statistics-based plot or map is a very powerful way to convey information to your audience or customers.

This is one of two brand-new units in Agronomy 513. You might not think a statistics software like R might be equipped to work with spatial data, especially after spending the first 11 weeks working with some ugly code. But R can readily work with shape files and rasters (think of a fertilizer application map), both creating and analyzing them. We will learn how to overlay polygons to relate soil to yield, and how to create a application map based on gridded soil tests.

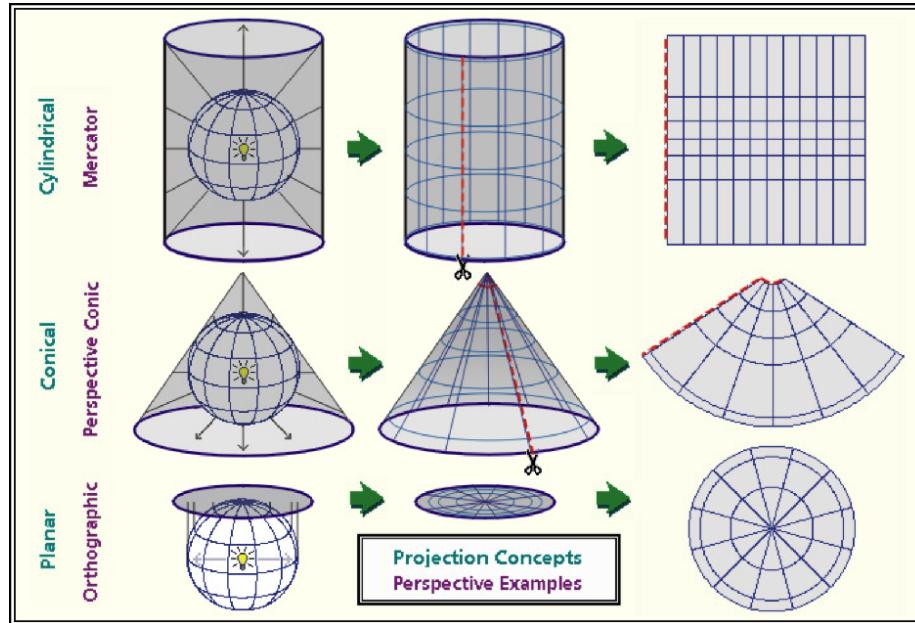
This unit will be light on calculations (yea!) and focus more on three key areas. First, what is projection and why do we need to worry about it if all we want to do is draw a map or average some data?

Second, what is a shapefile? How do we make sure it's projection is correct for our analyses and for joining with other shapefiles? How can layer the shapefile data with soil survey data readily accessible through R? How can we create an attractive map with our shapefile data?

Finally, we will study rasters. Rasters organize data in grids of cells that are of equal dimensions. Using point data from a shapefile, we can use tools like kriging to interpolate (predict) the value of each cell in the raster, creating another kind of map we can use to understand spatial data trends.

## 12.1 Projection (General)

One of the most challenging concepts for me when I began working with spatial data was *projection*. To be honest, it is still a challenging concept for me! Projection describes how we represent points on the surface of the earth, which is spheroidal, using maps, which are flat.



As we can see in the figure above, projection differ in how they are positioned relative to the earth's surface. Some are positioned relative to the equator, others might be centered between the equator and the poles, while yet others may be positioned at the poles. Each map will represent the center of it's geography better than the edges.

Each map is a compromise between the representation of boundaries (positions on the earth's surface) and the areas within those boundaries. Maps that pursue the accurate representation of boundaries on the earth's surface are going to end up distorting the area of geographies outside the focal point of the map. Maps that accurately represent areas are going to distort the position of geographic boundaries on the earth's surface. Thus, there are hundreds of different projection systems, focused on different areas of the earth, and using different units to describe the position of borders.

“Whoa, Marin”, you may be thinking. “I’m not trying to represent the world, the United States, Minnesota, or even my county! It’s just a freaking yield map!” And you would be absolutely correct: none of these projection systems are going to vary much in how they represent the location or area of a single section of land.



Figure 12.1: from <https://datacarpentry.org/r-raster-vector-geospatial/09-vector-when-data-dont-line-up-crs/>

But, in working with spatial data from that field, you will encounter differences among systems in how they locate your field on the face of the earth. Therefore, it is important we look at a few examples so you understand how to process those data.

We will start in the lower corner with WGS 84. This is the geographic system with which most of you are probably familiar. It is also how I roll with most of my analyses. It's simplistic, but it works just fine for point geographies – that is, single points on the earth's surface.

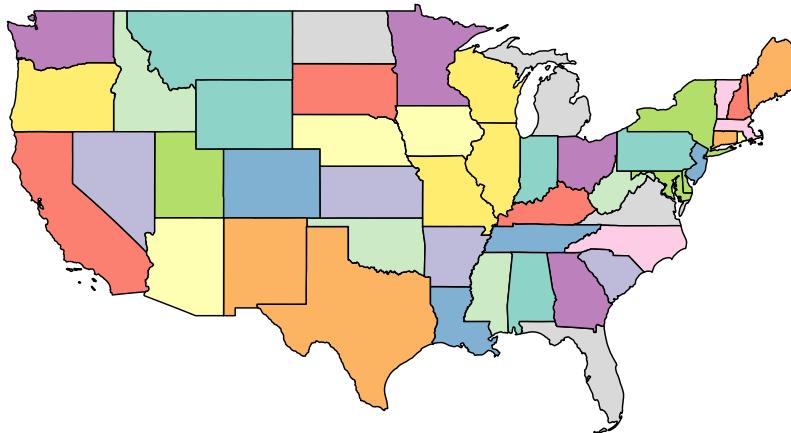
### 12.1.1 WGS 84 (EPSG: 4236)

WGS 84 refers to “World Geodetic System”; 84 refers to 1984, the latest (!) revision of this system. WGS 84 uses the earth’s center as its *origin*. An origin is the reference point for any map – each location is then geo-referenced according to its position relative to the origin. In WGS 84, the position of each location is described by its angle, relative to the origin. We usually refer to these angles as degrees latitude and longitude.

EPSG (EPSG Geodetic Parameter Dataset) is a set of many, many systems used to describe the coordinates of points on the Earth’s surface. and how they are projected onto flat maps. The EPSG stands for “European Petroleum Survey Group” – presumably, for the purpose of locating oil fields. 4326 is the code that EPGS uses to represent the WGS 84 system.

We can map the continental United states using

**state\_name**

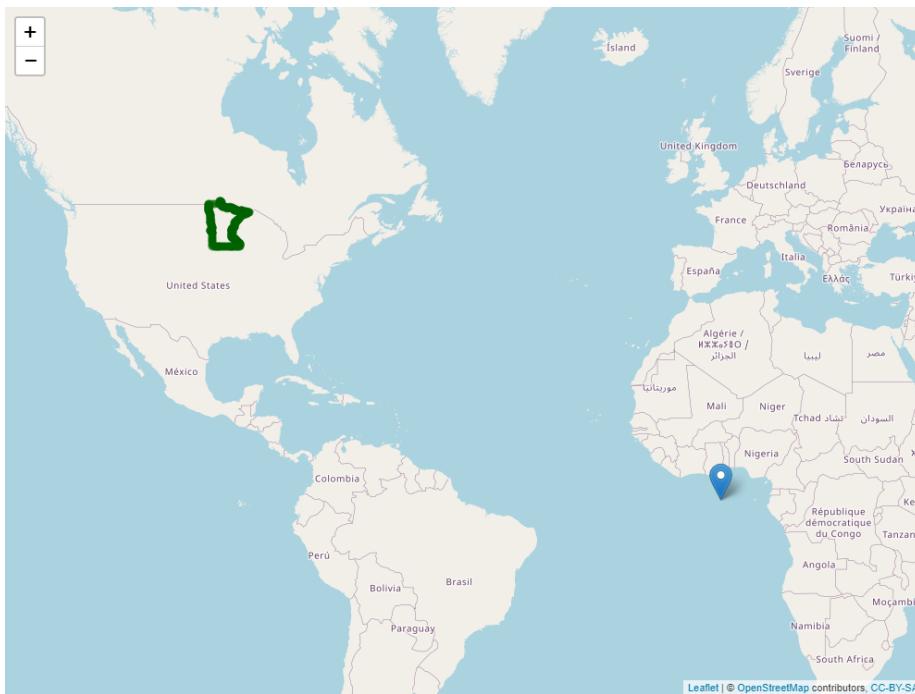


This is the flat map with which most of us are familiar. Latitude and longitude are drawn as parallel lines on this map. The map data are in a shapefile, a format we encountered at the beginning of this course. Let's look at the top few rows of this shapefile.

```
## Simple feature collection with 6 features and 1 field
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -124.4096 ymin: 32.53416 xmax: -86.80587 ymax: 49.38436
## Geodetic CRS:  WGS 84
##   state_name           geometry
## 1 California MULTIPOLYGON (((-118.594 33...
## 2 Wisconsin MULTIPOLYGON (((-86.93428 4...
## 3 Idaho MULTIPOLYGON (((-117.243 44...
## 4 Minnesota MULTIPOLYGON (((-97.22904 4...
## 5 Iowa MULTIPOLYGON (((-96.62187 4...
## 6 Missouri MULTIPOLYGON (((-95.76564 4...
```

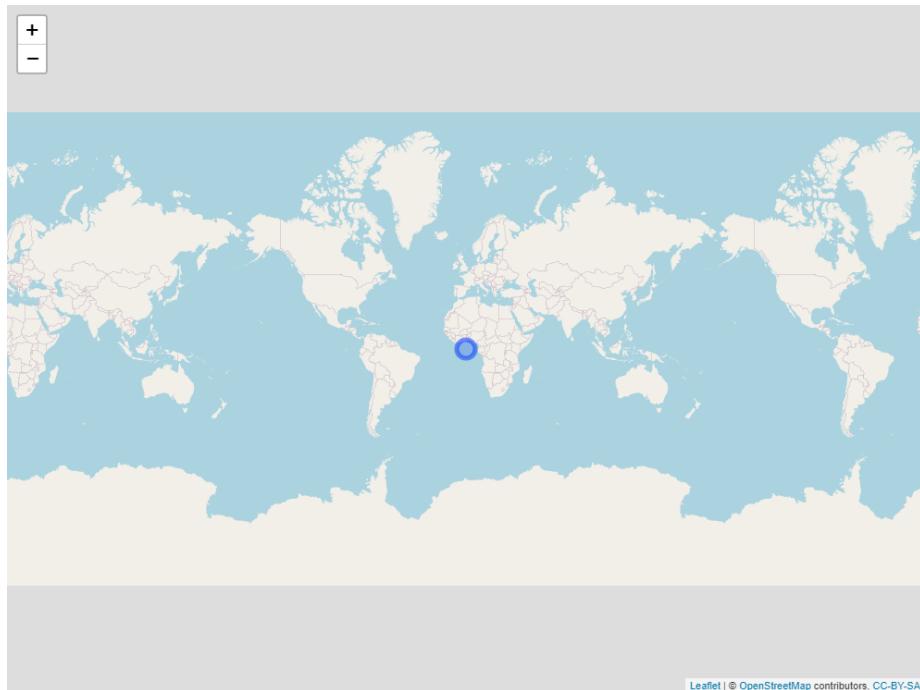
This is a complex dataset, so we will use the Minnesota state boundary as an example. In the map below, there are two objects. The pin in the map represents the map origin. The green dots indicate the Minnesota border.

```
## Warning in st_cast.sf(., "POINT"): repeating attributes for all sub-geometries
## for which they may not be constant
```



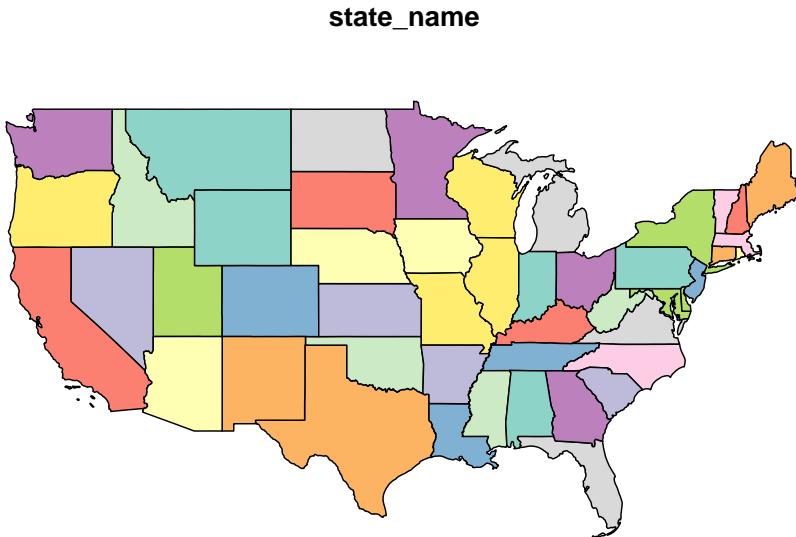
Zoom in on Minnesota and click around its borders. You will notice two things. First, each point is specified in latitude and longitude. Second, longitude (the first number) is always negative while latitude (the second number) is always positive.

The sign and size of geocoordinates in a projection system is defined two things: 1) where it places its origin (its reference point for locating objects on the map) and 2) what measurement units it uses. In the case of WGS 84, the origin is the intersection of the Prime Meridian and the Equator. Since all of the continental United States is in the western hemisphere, every state will have a negative longitude and a positive latitude. Since WGS 84 uses angles, the measurement units will be in degrees, which never exceed the range of (-180, 180) for longitude and (-90,90) for latitude.



### 12.1.2 Mercator (EPSG: 3857)

The Mercator System is commonly used to project data onto a sphere. If you look at the map below, it is very similar (actually related) to the WGS 84 map above but you may be able to see a slight “dome illusion” to the way the map is displayed. This projection is regularly used by online mapping services.



Looking at the top few rows of the Minnesota data points, we can see the units are not latitude and longitude. In this projection, they are easting and northing: measures of the distance east and north of the origin. Easting and northing are usually measured in meters

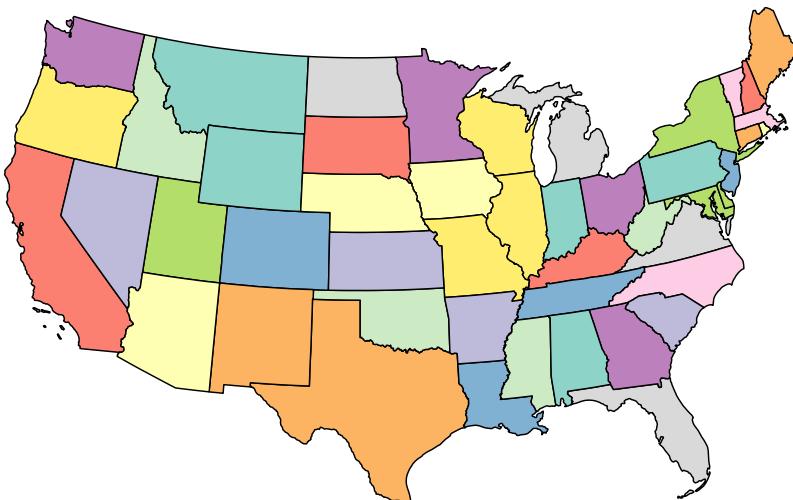
```
## Simple feature collection with 6 features and 1 field
## Geometry type: POINT
## Dimension: XY
## Bounding box: xmin: -10823490 ymin: 6274648 xmax: -10610960 ymax: 6274978
## Projected CRS: WGS 84 / Pseudo-Mercator
##   state_name           geometry
## 1  Minnesota POINT (-10823487 6274978)
## 1.1 Minnesota POINT (-10790305 6274859)
## 1.2 Minnesota POINT (-10731801 6274859)
## 1.3 Minnesota POINT (-10683932 6274859)
## 1.4 Minnesota POINT (-10613307 6274648)
## 1.5 Minnesota POINT (-10610961 6274651)
```

The origin for the Mercator projection is again the intersection of Prime Meridian and Equator, so each Minnesota border point will have a negative value for easting and a positive value for northing.

### 12.1.3 US National Atlas Equal Area (EPSG: 2163)

As the name suggests, coordinate systems like the US National Atlas Equal Area project data so that the areas of geographic objects are accurate in the map.

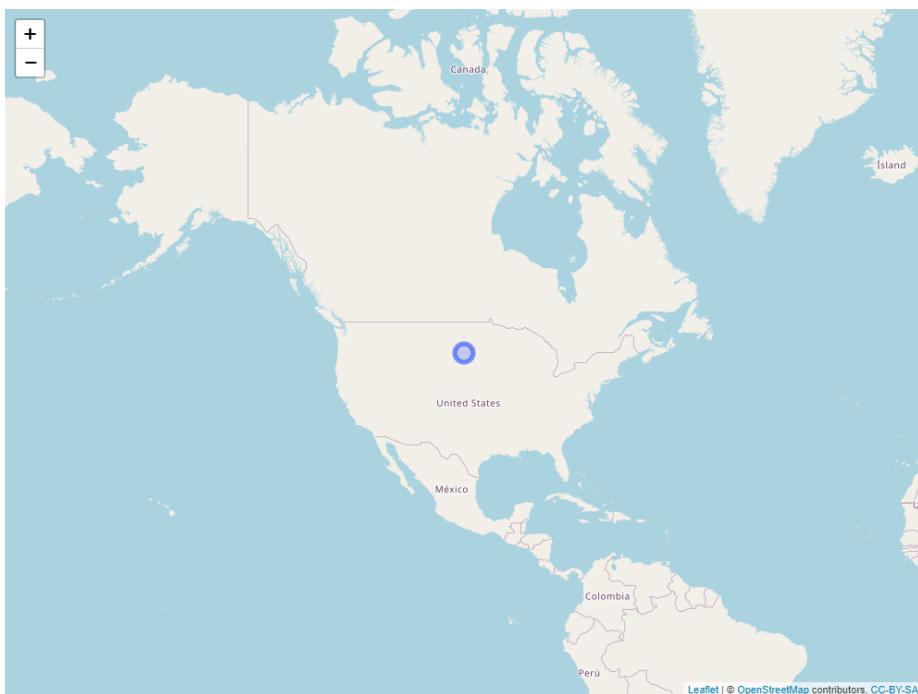
**state\_name**



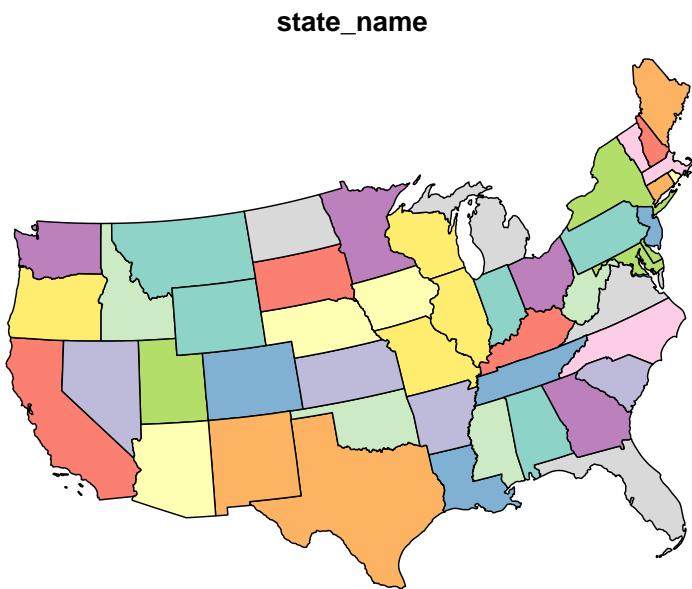
This system, like the Mercator above, uses northing and easting units. But when we look at our Minnesota border coordinates, we now notice our easting values are positive! What happened?

```
## Simple feature collection with 6 features and 1 field
## Geometry type: POINT
## Dimension: XY
## Bounding box: xmin: 202876 ymin: 448171.5 xmax: 342482.3 ymax: 454473.8
## Projected CRS: NAD27 / US National Atlas Equal Area
##   state_name           geometry
## 1   Minnesota POINT (202876 448171.5)
## 1.1 Minnesota POINT (224686.3 448890.9)
## 1.2 Minnesota POINT (263125.5 450495.2)
## 1.3 Minnesota POINT (294567.2 451996.1)
## 1.4 Minnesota POINT (340942.6 454381.6)
## 1.5 Minnesota POINT (342482.3 454473.8)
```

As you have likely guessed, our origin has changed. For this projection, our origin is in Central South Dakota.



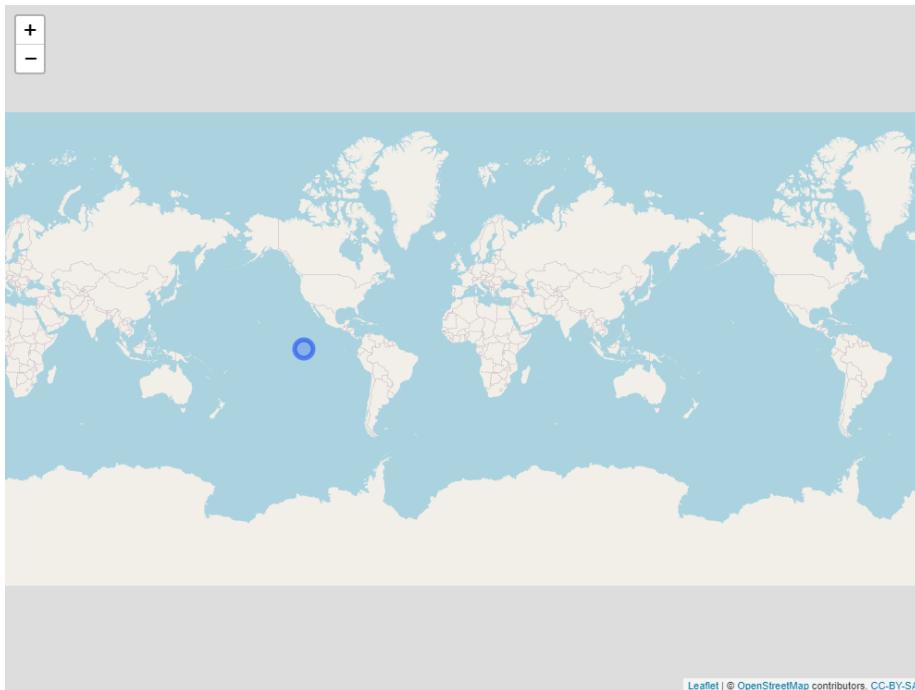
#### 12.1.4 UTM Zone 11N (EPSG: 2955)



```
## Simple feature collection with 6 features and 1 field
## Geometry type: MULTIPOLYGON
## Dimension: XY
## Bounding box: xmin: -128697 ymin: 3599652 xmax: 3013312 ymax: 5706611
## Projected CRS: NAD83(CSRS) / UTM zone 11N
##   state_name           geometry
## 1 California MULTIPOLYGON (((351881.3 37...
## 2 Wisconsin MULTIPOLYGON (((2845962 548...
## 3 Idaho MULTIPOLYGON (((480645 4915...
## 4 Minnesota MULTIPOLYGON (((1941564 561...
## 5 Iowa MULTIPOLYGON (((2169056 494...
## 6 Missouri MULTIPOLYGON (((2302630 471...
```

Here are the coordinates for the Minnesota border again.

```
## Simple feature collection with 6 features and 1 field
## Geometry type: POINT
## Dimension: XY
## Bounding box: xmin: 1941564 ymin: 5618787 xmax: 2079676 ymax: 5658008
## Projected CRS: NAD83(CSRS) / UTM zone 11N
##   state_name           geometry
## 1   Minnesota POINT (1941564 5618787)
## 1.1  Minnesota POINT (1963162 5624612)
## 1.2  Minnesota POINT (2001185 5635247)
## 1.3  Minnesota POINT (2032277 5644167)
## 1.4  Minnesota POINT (2078155 5657549)
## 1.5  Minnesota POINT (2079676 5658008)
```



### 12.1.5 Projection Summary

It is good to know some of these basic projections, but by far the most important concept of this unit is that it is important you are aware of the projection system that accompanies your spatial data. If you are assembling data from multiple shapefiles, as we will do below with soil and yield maps, you will need to account for the projections of each shapefile, to make sure they all have the same projection system.

In addition, different spatial data operations may prefer one projection system over the other. Operations that summarize areas will require projections that are based on area, not geometry. Similarly, spatial tools like rasters (which divide an area into rectangles or squares), will prefer a system that is square.

## 12.2 Shape Files

### 12.2.1 Case Study: Soybean Yield in Iowa

This is not our first encounter with shapefiles – we plotted our first shapefile map in the beginning of our course. Let's return to that dataset!

```
library(tidyverse)
library(sf)

corn_yield = st_read("data-unit-12/merriweather_yield_map/merriweather_yield_map.shp",
```

We can examine this shapefile by typing its name.

```
corn_yield
```

```
## Simple feature collection with 6061 features and 12 fields
## Geometry type: POINT
## Dimension:      XY
## Bounding box:  xmin: -93.15474 ymin: 41.66619 xmax: -93.15026 ymax: 41.66945
## Geodetic CRS:  WGS 84
## First 10 features:
##   DISTANCE SWATHWIDTH VRYIELDVOL Crop  WetMass Moisture           Time
## 1  0.9202733          5  57.38461  174 3443.652    0.00 9/19/2016 4:45:46 PM
## 2  2.6919269          5  55.88097  174 3353.411    0.00 9/19/2016 4:45:48 PM
## 3  2.6263101          5  80.83788  174 4851.075    0.00 9/19/2016 4:45:49 PM
## 4  2.7575437          5  71.76773  174 4306.777    6.22 9/19/2016 4:45:51 PM
## 5  2.3966513          5  91.03274  174 5462.851   12.22 9/19/2016 4:45:54 PM
## 6  3.1840529          5  65.59037  174 3951.056   13.33 9/19/2016 4:45:55 PM
## 7  3.3480949          5  60.36662  174 3668.554   14.09 9/19/2016 4:45:55 PM
## 8  2.6919269          5  65.85538  174 4090.685   15.95 9/19/2016 4:46:16 PM
## 9  3.1840529          5  85.21010  174 5217.806   14.74 9/19/2016 4:46:21 PM
## 10 3.1184361          5  69.64239  174 4260.025   14.65 9/19/2016 4:46:21 PM
##   Heading VARIETY Elevation           IsoTime yield_bu
## 1 300.1584  23A42  786.8470 2016-09-19T16:45:46.001Z 65.97034
## 2 303.6084  23A42  786.6140 2016-09-19T16:45:48.004Z 64.24158
## 3 304.3084  23A42  786.1416 2016-09-19T16:45:49.007Z 92.93246
## 4 306.2084  23A42  785.7381 2016-09-19T16:45:51.002Z 77.37348
## 5 309.2284  23A42  785.5937 2016-09-19T16:45:54.002Z 91.86380
## 6 309.7584  23A42  785.7512 2016-09-19T16:45:55.005Z 65.60115
## 7 310.0084  23A42  785.7840 2016-09-19T16:45:55.996Z 60.37653
## 8 345.7384  23A42  785.6068 2016-09-19T16:46:16.007Z 65.86630
## 9 353.3184  23A42  785.7545 2016-09-19T16:46:21.002Z 85.22417
## 10 353.1584 23A42  785.8365 2016-09-19T16:46:21.994Z 69.65385
##   geometry
## 1 POINT (-93.15026 41.66641)
## 2 POINT (-93.15028 41.66641)
## 3 POINT (-93.15028 41.66642)
## 4 POINT (-93.1503 41.66642)
## 5 POINT (-93.15032 41.66644)
## 6 POINT (-93.15033 41.66644)
```

```
## 7 POINT (-93.15034 41.66645)
## 8 POINT (-93.15029 41.66646)
## 9 POINT (-93.15029 41.66648)
## 10 POINT (-93.15029 41.66649)
```

The most useful shapefiles, in my experience, are presented in the “spatial feature” format above. It is, essentially, a data frame, but with a single, special geometry column that contains multiple measures per row. The geometry column is, if you will, composed of columns within a column.

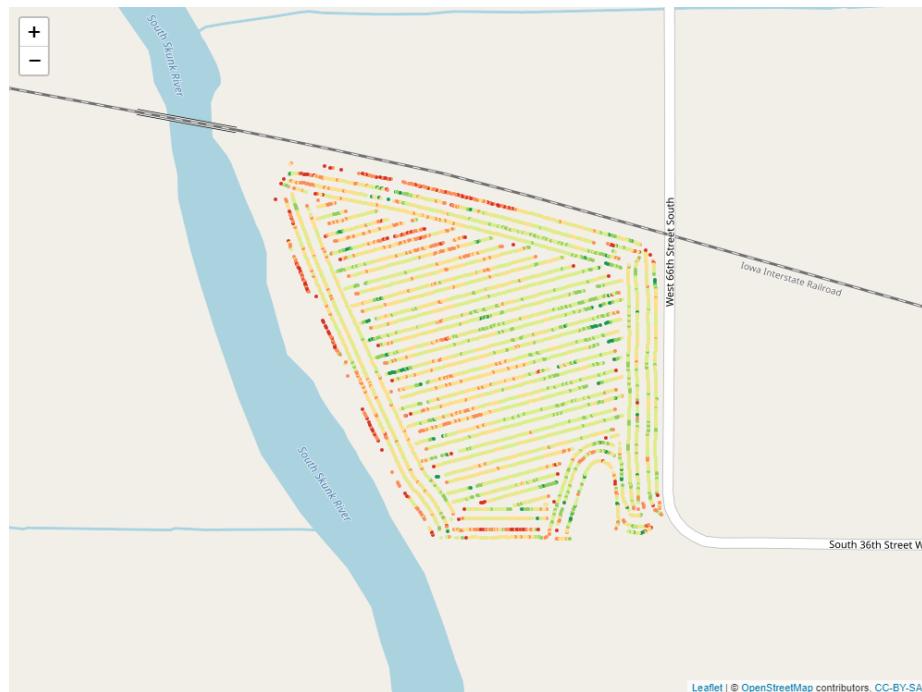
Let’s ignore the data for now and look at the information (the metadata) at the top of the output. First, let’s note the geometry type is POINT. Each row of this datafile defines one point. Shapefiles can be composed of all sorts of objects: points, lines, polygons, sets of multiple polygons, and so forth – and shapefiles can be converted between formats

Creating a successful map includes telling R what kind of object we intend to draw. So knowing the format of a shapefile is critical! a helpful starting point.

Second, look at the geographic CRS. CRS stands for Coordinate Reference System. In this case, we are already in the standard WGS 84 format we discussed earlier, so our units are latitude and longitude.

One of the things we will learn this lesson is to use *Leaflet* to create maps. Leaflet is an awesome applet whose true appreciation would require using four-letter conjunctions inappropriate for the classroom. It creates interactive maps that can zoom in, zoom out, and identify the values of individual points.

```
## Warning: package 'RColorBrewer' was built under R version 4.1.3
```



### 12.2.2 SSURGO

The Soil Survey Geographic Database (SSURGO) is maintained by the United States Department of Agriculture. It contains extensive soil surveys: soil evaluations for properties, susceptibility to weather extremes, suitability for agriculture, recreation, and buildings. The soil survey used to only be available in county books, which only special libraries had. Now, you can access all these data through R in seconds and match them precisely to a given map location.

The SSURGO data is in a database. A database is a series of tables, all describing different aspects of a data information. Each table contains 1-3 columns that are keys to match the tables with each other. Descriptions of the tables and their data can be obtained for SSURGO at:

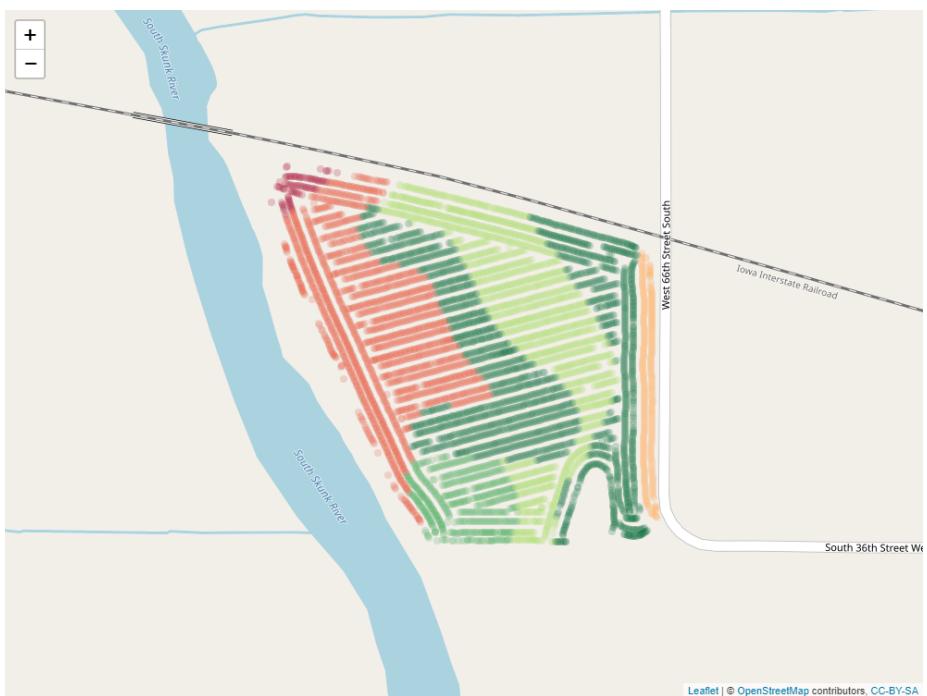
[https://data.nal.usda.gov/system/files/SSURGO\\_Metadata\\_-\\_Table\\_Column\\_Descriptions.pdf](https://data.nal.usda.gov/system/files/SSURGO_Metadata_-_Table_Column_Descriptions.pdf)

Putting all these tables together can be messy – fortunately, you only need to do it once, after which you can just change the shapefile you feed to the code. I will give you that code in the exercises this week.

Here is a SSURGO map of soil organic matter.



Here is another map, this time with the percent clay.



Now that we have our SSURGO data, we can join it with our yield data and ask questions how yields were grouped by quantitative descriptors, such as soil map unit name (“muname”), texture (“texdesc”), drainage class (“drainagecl”), or parent material (“pmkind”).

```
##   yield_bu                               muname hzname sandtotal.r
## 1 65.97034 Wiota silt loam, 0 to 2 percent slopes    H1      9.4
## 2 64.24158 Wiota silt loam, 0 to 2 percent slopes    H1      9.4
## 3 92.93246 Wiota silt loam, 0 to 2 percent slopes    H1      9.4
## 4 77.37348 Wiota silt loam, 0 to 2 percent slopes    H1      9.4
## 5 91.86380 Wiota silt loam, 0 to 2 percent slopes    H1      9.4
## 6 65.60115 Wiota silt loam, 0 to 2 percent slopes    H1      9.4
##   silttotal.r claytotal.r om.r awc.r ksat.r cec7.r     chkey texdesc
## 1       67.1        23.5   4 0.22      9  22.5 59965160 Silt loam
## 2       67.1        23.5   4 0.22      9  22.5 59965160 Silt loam
## 3       67.1        23.5   4 0.22      9  22.5 59965160 Silt loam
## 4       67.1        23.5   4 0.22      9  22.5 59965160 Silt loam
## 5       67.1        23.5   4 0.22      9  22.5 59965160 Silt loam
## 6       67.1        23.5   4 0.22      9  22.5 59965160 Silt loam
##   drainagecl slope.r pmkind           geometry
## 1 Well drained      1 Alluvium POINT (-93.15026 41.66641)
## 2 Well drained      1 Alluvium POINT (-93.15028 41.66641)
## 3 Well drained      1 Alluvium POINT (-93.15028 41.66642)
## 4 Well drained      1 Alluvium POINT (-93.1503 41.66642)
## 5 Well drained      1 Alluvium POINT (-93.15032 41.66644)
## 6 Well drained      1 Alluvium POINT (-93.15033 41.66644)
```

For example, here are soybean yields by soil texture, which would suggest a trend where soil yield increased with clay content in this field.

```
## # A tibble: 4 x 2
##   texdesc      yield_bu
##   <chr>       <dbl>
## 1 Clay loam    81.6
## 2 Silty clay loam 80.7
## 3 Silt loam    79.2
## 4 Loam         78.0
```

And this table would suggest that soybean preferred poorly drained soil to better-drained soils.

```
##          drainagecl yield_bu
## 1 Poorly drained 81.32400
## 2 Well drained   78.47489
## 3 Somewhat poorly drained 78.09084
```

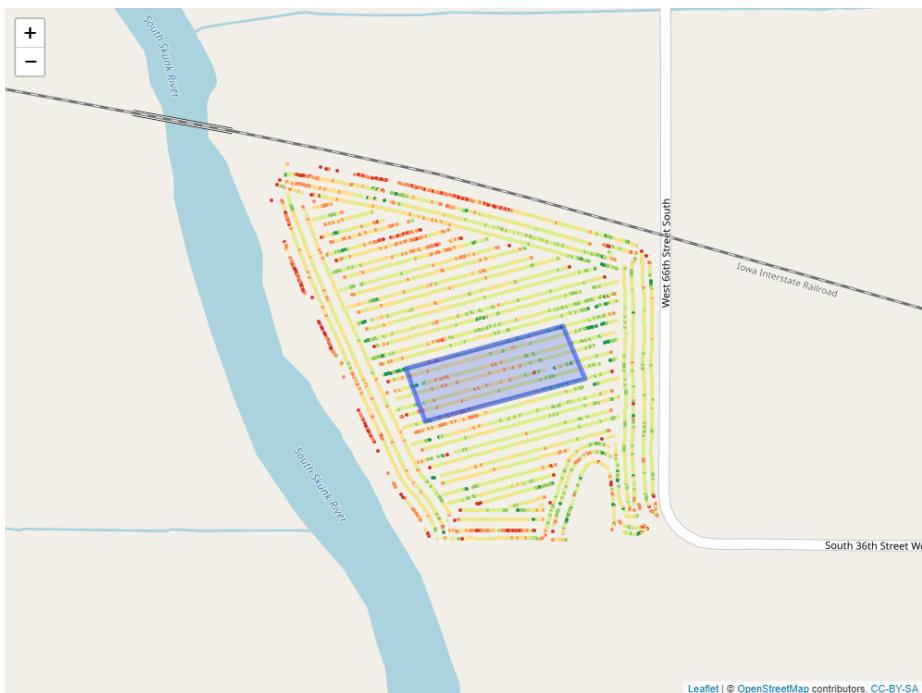
### 12.2.3 Operations with Shapes

Above, we subsetted our yield data according to different soil properties. In some cases, however, we may want to subset or group data by location.

#### 12.2.3.1 Intersection

Say, for example, we applied a foliar fertilizer treatment to part of the field, as shown in the map below.

```
## [1] "temp\\file55b060fd1edd"
```



How might we find out statistics for yield measures within that applied area?

```
## Warning: attribute variables are assumed to be spatially constant throughout all
## geometries

## [1] 79.74649
```



### 12.2.3.2 Difference

What about the yields outside that area?

```
## Warning: attribute variables are assumed to be spatially constant throughout all  
## geometries
```



```
mean(field_yield_outside$yield_bu)
```

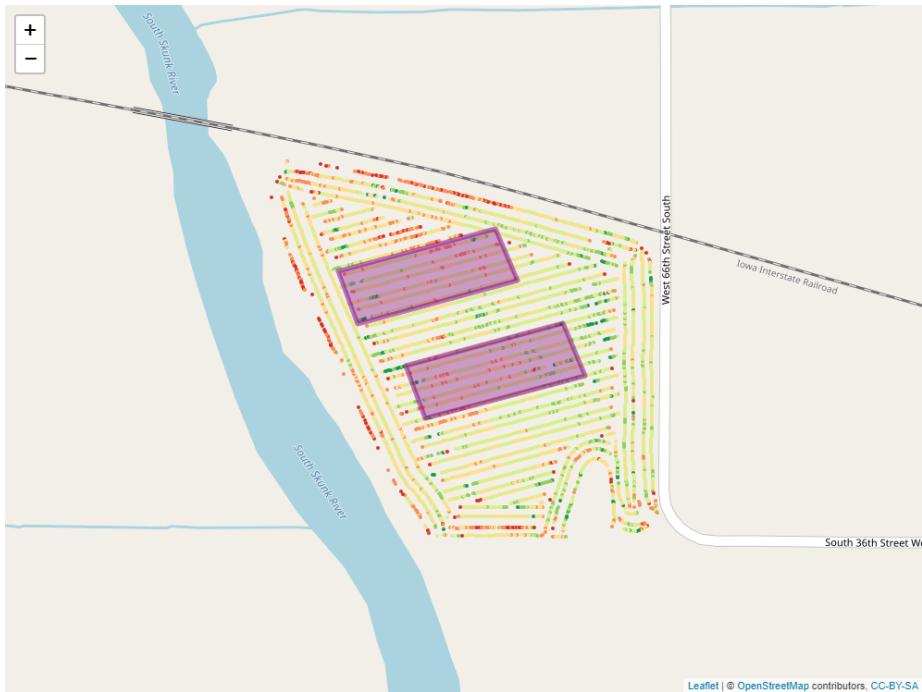
```
## [1] 80.12065
```

### 12.2.3.3 Union

What if we had two field plot areas?



If we wanted to analyze two areas together, we could use `st_union()` to combine them:



```
mean(corn_yield$yield_bu)
```

```
## [1] 80.09084
```

## 12.3 Rasters

```
library(stars)

selected_data = point_data %>%
  filter(attribute=="P_bray")

### make grid
grd = st_bbox(boundary) %>%
  st_as_stars() %>%
  st_crop(boundary)
# %>%
#   st_set_crs(6505)

# ordinary kriging -----
```

```
v = variogram(measure~1, selected_data)
m = fit.variogram(v, vgm("Sph"))
krige_plot = plot(v, model = m)

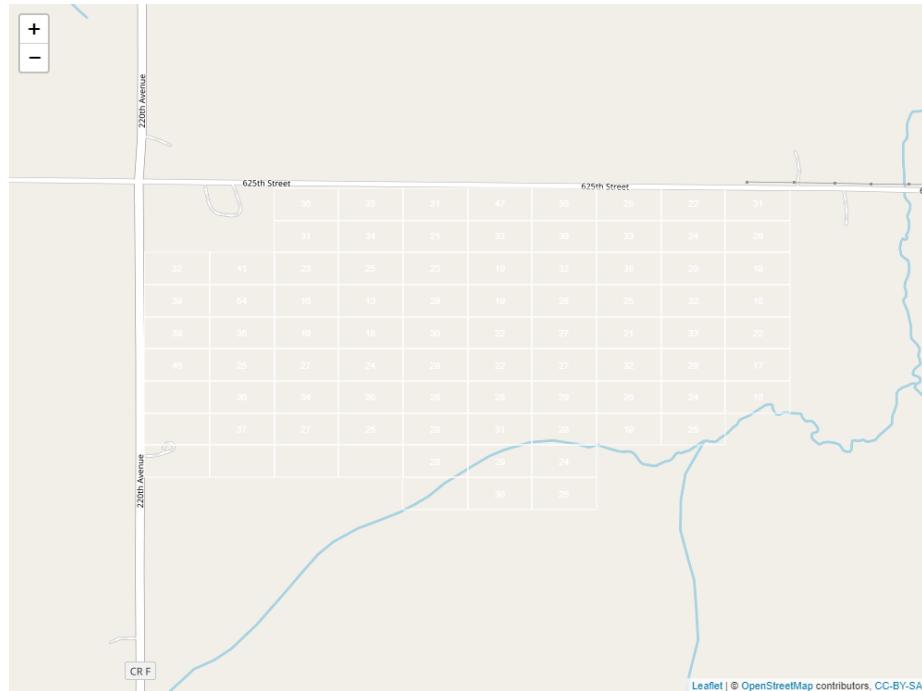
lzn.kr1 = gstat:::krige(formula = measure~1, selected_data, grd, model=m)

## [using ordinary kriging]

# plot(lzn.kr1[1])

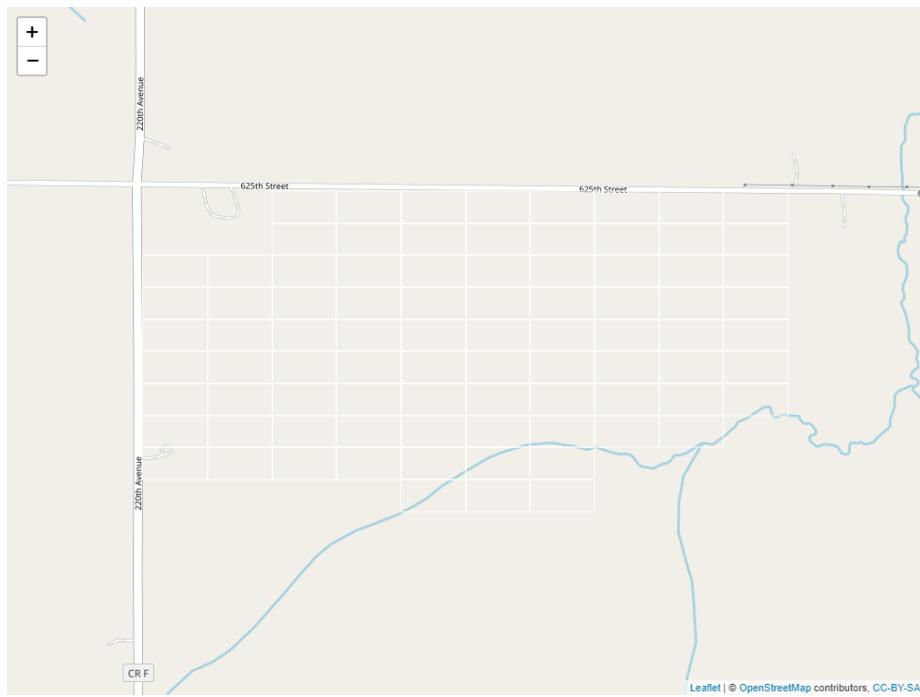
library(leafem)

soil_p_map = leaflet(lzn.kr1[1]) %>%
  addTiles() %>%
  addStarsImage(opacity = 0.5)
```

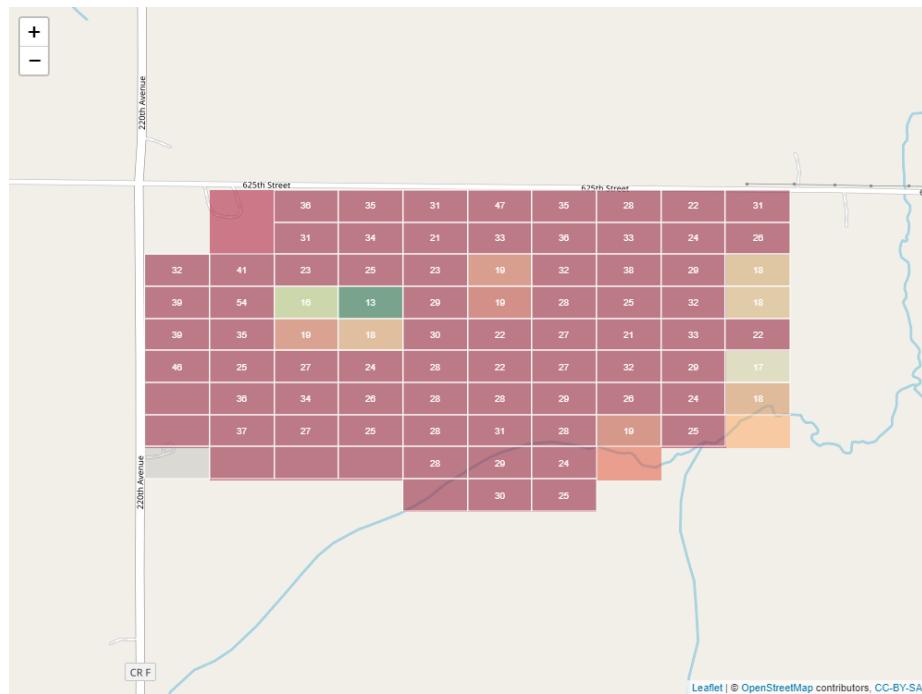


In the previous section, we worked with what are often described as vectors or shapes. That is, points which may or may not have been connected to form lines or polygons.

A raster is a grid system that we use to describe spatial variation. In essence, it is a grid system. Here is the same field we worked with in the previous section, now overlaid with a grid:



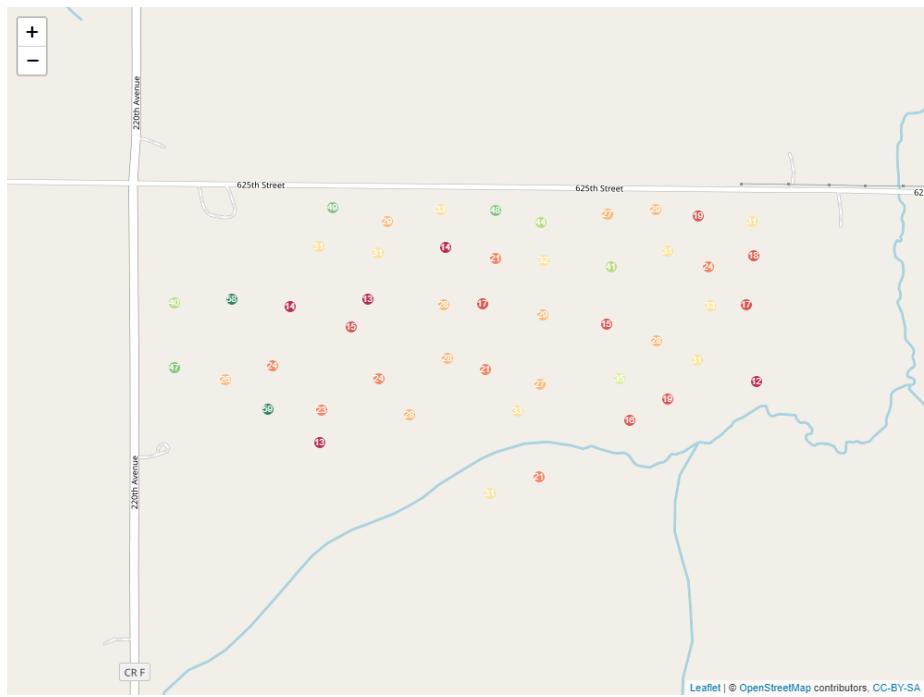
If you think it looks a little like we took a spreadsheet and trimmed it to fit our field, you are exactly right. Taking this analogy further, just as a spreadsheet is composed of cells, each containing a different value, so is a raster. Here it is, filled in with values representing predicted soil P test values:



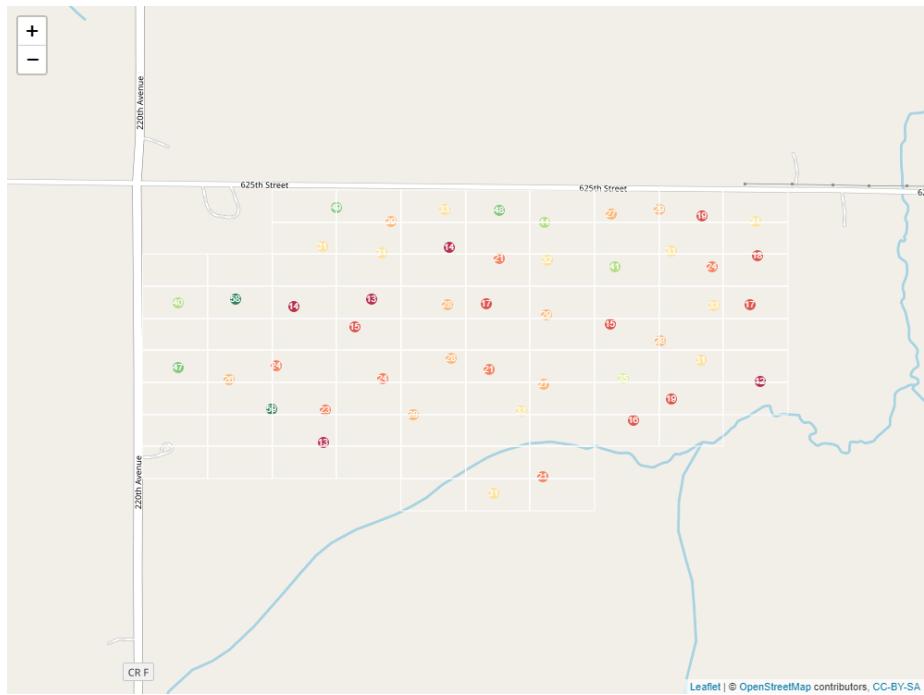
Often, the cells would be colored along a given gradient (say red-yellow-green) according to their values. This helps us to see spatial trends.

### 12.3.1 Interpolation

To create a raster that represents continuous soil trends across a field, however, we need to do a little modelling. You see, we start out with a set of soil cores like this:

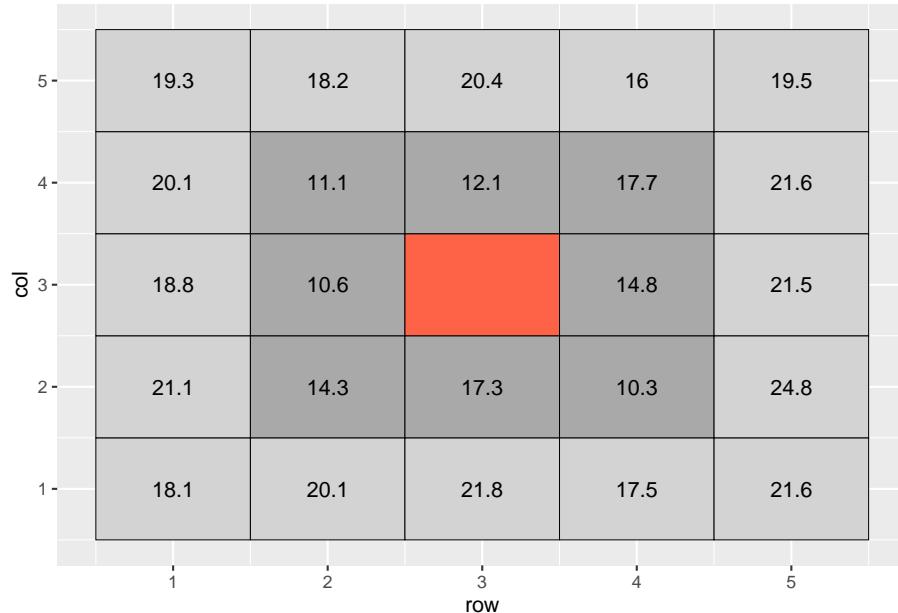


But we are trying to make predictions for each cell of our raster:



Some cells above one measure, others split a measure with a neighboring cell, and yet others contain no measure at all. In addition, even cells that contain a measure may have it in different locations relative to the cell center.

When we interpolate a raster, we make an educated guess about the values of cells in which no measure was taken, based on the values of the other cells. In the following example, the middle cell is missing:



The most basic way to interpolate this missing value would be to estimate its value as the mean value of all cells adjacent to it (dark grey in plot above). So the value of the missing cell would be equal to:

$$\text{cell value} = \text{mean}(16.6, 17.9, 13.1, 19.1, 11.6, 11.1, 16.2, 16.2) = 16.6$$

If we wanted to be a bit more accurate, we might extend out to the next ring of cells around the cell we are trying to estimate. But we probably would not want them to factor into the mean calculation as much as the first ring of cells – the difference between two measurements tends to increase with distance. So if they are two units away from the missing cell of, we might weight them so they contribute 1/4th as much to the estimate as the immediately adjacent cells.

If we were to fill out a table, it would look like this:

```
## # A tibble: 25 x 6
##       row   col cell value weight weighted_value
##   <int> <int> <int> <dbl>  <dbl>          <dbl>
## 1     1     1    18.1 18.1    1.00      18.1
## 2     1     2    20.1 20.1    1.00      20.1
## 3     1     3    21.8 21.8    1.00      21.8
## 4     1     4    17.5 17.5    1.00      17.5
## 5     1     5    21.6 21.6    1.00      21.6
## 6     2     1    21.1 21.1    1.00      21.1
## 7     2     2    14.3 14.3    1.00      14.3
## 8     2     3    17.3 17.3    1.00      17.3
## 9     2     4    10.3 10.3    1.00      10.3
## 10    2     5    24.8 24.8    1.00      24.8
## 11    3     1    18.8 18.8    1.00      18.8
## 12    3     2    10.6 10.6    1.00      10.6
## 13    3     4    14.8 14.8    1.00      14.8
## 14    3     5    21.5 21.5    1.00      21.5
## 15    4     1    20.1 20.1    1.00      20.1
## 16    4     2    11.1 11.1    1.00      11.1
## 17    4     3    12.1 12.1    1.00      12.1
## 18    4     4    17.7 17.7    1.00      17.7
## 19    4     5    21.6 21.6    1.00      21.6
## 20    5     1    19.3 19.3    1.00      19.3
## 21    5     2    18.2 18.2    1.00      18.2
## 22    5     3    20.4 20.4    1.00      20.4
## 23    5     4    16.0 16.0    1.00      16.0
## 24    5     5    19.5 19.5    1.00      19.5
## 25    3     3    16.6 16.6    1.00      16.6
```

```

## 1   1   1   1 18.1 0.25    4.53
## 2   2   1   2 20.1 0.25    5.03
## 3   3   1   3 21.8 0.25    5.45
## 4   4   1   4 17.5 0.25    4.38
## 5   5   1   5 21.6 0.25    5.4
## 6   1   2   6 21.1 0.25    5.28
## 7   2   2   7 14.3 1       14.3
## 8   3   2   8 17.3 1       17.3
## 9   4   2   9 10.3 1       10.3
## 10  5   2  10 24.8 0.25    6.2
## # ... with 15 more rows

```

The weighted value for each cell is the product of its observed value times its weight. To calculate the weighted value, we sum the weights and the weighted values. The weighted mean is then:

$$\text{weighted mean} = \frac{\sum \text{weighted value}}{\sum \text{weight}}$$

In this example, the calculation would look like:

$$\text{weighted mean} = \frac{220.45}{13} = 17.0$$

What we have just calculated is called the *inverse distance-weighted (IDW) mean* of the surrounding points. It is a simple, elegant way to estimate the missing value.

### 12.3.2 Kriging

The inverse distance-weighted mean, however, is not as accurate as which we are capable. We assume that the influence of points away from the empty cell decreases exponentially with distance. We don't consider how we would optimally weight the values of the surrounding cells.

We can develop a more complex, but likely accurate, estimate of the cell value using a different interpolation practice called *kriging*. (For some reason, I always want to insert a “d” into this term so it rhymes with “bridging”. But it is pronounced KREE-ging.) The name comes from Danie Krige, a South African geostatistician who was interested in locating gold deposits.

I will take you through the basics of kriging. A more elegant explanation of kriging can be found here: <https://pro.arcgis.com/en/pro-app/tool-reference/3d-analyst/how-kriging-works.htm>

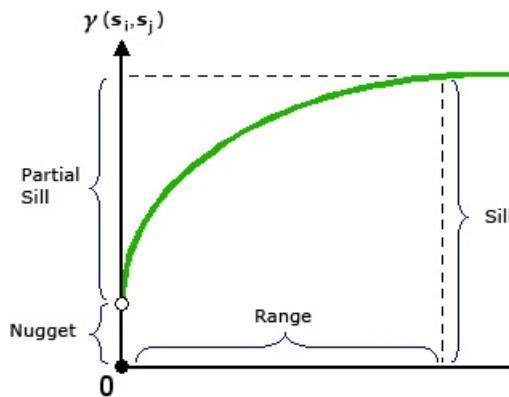


Figure 12.2: from “How Kriging Works” (<https://pro.arcgis.com/en/pro-app/tool-reference/3d-analyst/how-kriging-works.htm>)

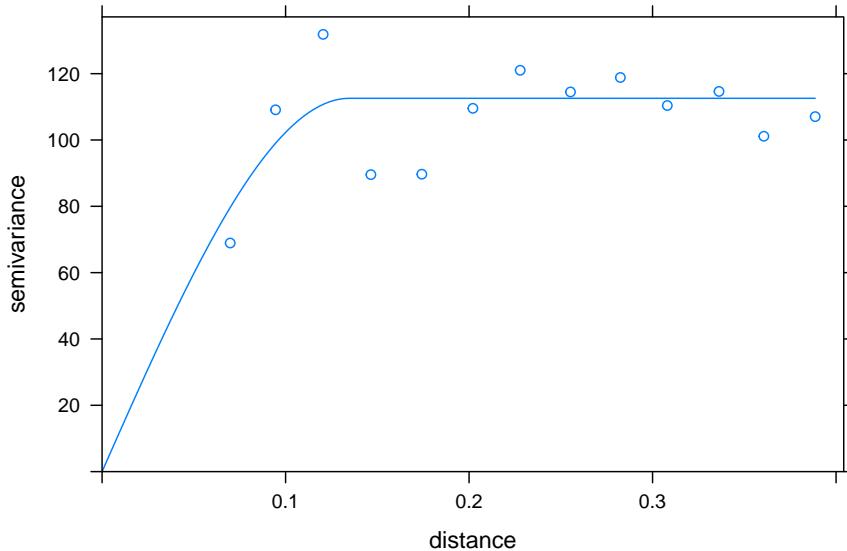
Kriging evaluates how points of varying differences from each other are correlated. These correlations are plotted in a *semivariogram*.

X is the distance between point pairs. Y is the squared difference between each pair of points selected by the software. Sometimes, pairs will be binned (similar to the binning in histograms) into according to distance (called “lag”) in this analysis. The squared differences of all pairs within a lag bin are averaged.

Does this nonlinear function look familiar by any chance? That’s right, it is a monomolecular function! The curve is described by different terms to which we are used to (and, to be honest, they don’t always make much sense.) In the kriging curve, the *Sill* is the maximum value the curve approaches. The *Nugget* is the y-intercept.

Otherwise, this curve is fit with nonlinear regression, just like the others we have seen. The calculated semivariances are then used to weight observations in calculating the weighted me. In this way, observations are weighted according to the strength of surrounding measurements, according to their measured correlation with distance.

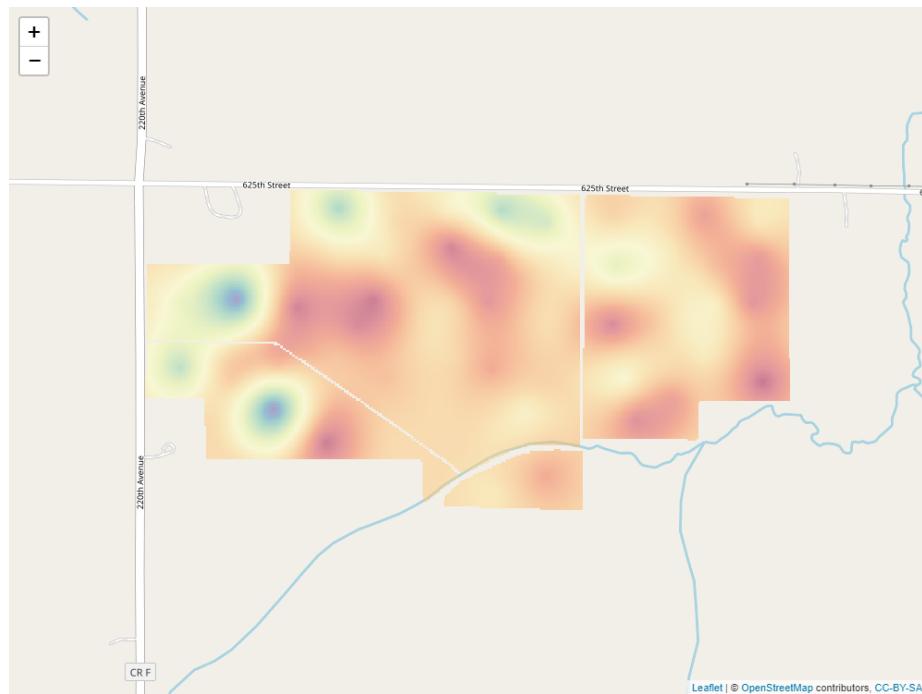
Here is the kridge plot for our soil phosphorus test data.



Using this model of the relationship between covariance and distance, then, R can determine how much to weight each observation, based on distance, to estimate values between measurement points.

When we produced our initial raster, the cell size was especially large for simplification of the raster, and to allow the cell values to be shown. When we build a raster with kriging, however, the cell size can be very small. This has the effect of getting rid of blockiness and allowing us to better predict and visualize trends in values across the landscape.

Here is our soil phosphorus test map, interpolated by R, using kriging. The red areas are areas of lower P test values. The green and blue areas have the greatest test values, and the yellow areas are intermediate.



### 12.3.3 Operations on Kriged Data

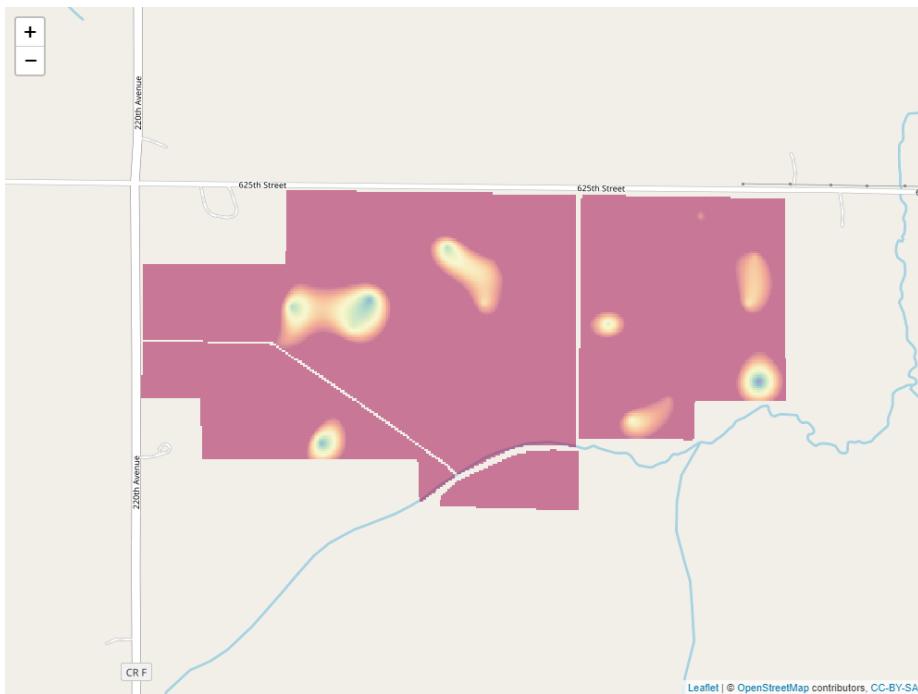
Of course, to take this example to completion, we would like to know the P-application rate for each of the cells within our raster. The University of Minnesota recommended application rate, based on soil P value and yield goal, is:

$$P_2O_5 \text{ recommended rate} = [0.700 - .035(\text{Bray P ppm})](\text{yield goal})$$

We can plug each cell's Bray P test value into this equation. For example, a point with a test value of 17 would, given a yield goal of 200, have a  $P_2O_5$  rate recommendation of:

$$P_2O_5 \text{ recommended rate} = [0.700 - .035(17)](200) = 21$$

Here is the rate recommendation map.



The blue areas now have the highest recommended rates. Yellow are intermediate. Red areas should receive no P fertilizer.

## 12.4 Exercise: Shape Files

If you are downloading data files from your tractor, applicator, or combine, they are probably in shape file format. Although I will refer to shape file in the singular, a shape file is typically bundled with multiple other files, with the same file name but different extensions: .dbf, .prj, and .shx. Of these three, the .prj is perhaps the most interesting, as it can be opened to view the projection used to georeference the records in the shape file.

### 12.4.1 Case Study

We will work with another soil test shapefile from Minnesota.

```
## Reading layer `Burnholdt_grid_sample` from data source
##   `C:\ds_ag_professionals\data-unit-12\Burnholdt_grid_sample.shp'
##   using driver `ESRI Shapefile'
## Simple feature collection with 1150 features and 9 fields
## Geometry type: POINT
```

```

## Dimension:      XY
## Bounding box:  xmin: -92.72799 ymin: 43.95069 xmax: -92.71914 ymax: 43.95704
## Geodetic CRS:  WGS 84

## Simple feature collection with 6 features and 9 fields
## Geometry type: POINT
## Dimension:      XY
## Bounding box:  xmin: -92.72784 ymin: 43.95342 xmax: -92.72423 ymax: 43.95438
## Geodetic CRS:  WGS 84
##   obs Sample_id SampleDate ReportDate P2    Grower      Field attribute measure
## 1 107        10  6/20/2019  6/20/2019  0 Burnholdt Burnholdt     Om   5.6
## 2 108        11  6/20/2019  6/20/2019  0 Burnholdt Burnholdt     Om   5.2
## 3 109        12  6/20/2019  6/20/2019  0 Burnholdt Burnholdt     Om   4.7
## 4 110        13  6/20/2019  6/20/2019  0 Burnholdt Burnholdt     Om   4.3
## 5 111        14  6/20/2019  6/20/2019  0 Burnholdt Burnholdt     Om   5.1
## 6 112        15  6/20/2019  6/20/2019  0 Burnholdt Burnholdt     Om   4.1
##               geometry
## 1 POINT (-92.72784 43.95342)
## 2 POINT (-92.72672 43.95342)
## 3 POINT (-92.72555 43.95343)
## 4 POINT (-92.72423 43.95343)
## 5 POINT (-92.72425 43.95433)
## 6 POINT (-92.72545 43.95439)

```

It is always important to inspect datasets before we analyze them. A particularly important objective in inspecting a shape file dataset is to see what type of *geometry* it includes: POINT, POLYGON, etc. Some equipment will use POINT geometry – ie a single georeference to indicate where the center of the equipment was at the time of measure. Others will use polygons to represent the width and length of the equipment. Personally, I find the latter format to be overkill, but I have come across it. (If you ever run into issues with that format, please feel free to contact me.)

We can confirm our measures here are referenced with POINT geometry.

In this exercise, we want to look at soil organic matter (attribute = Om in this file.), so let's isolate those measures. We can do so using the *filter()* function

```

single_attribute_data = soil_test_data %>%
  filter(attribute=="Om")

```

We are now ready to create our first map of the data. We will use that wonderful mapping application, leaflet. The code below consists of four lines.

- 1) “single\_attribute\_data” tells R which dataset we are going to use to build the map.

- 2) “leaflet()” tells R to use that app to build the map
- 3) “addCircleMarkers” tells R to draw point geometries. There are many other arguments we will add later to tell R what colors to use, marker size, and what value (if any) to show when the user clicks on a marker.
- 4) “addTiles()” tells R to use a street background for the map.

Well, that's cool, but what do the points mean? Let's add some color to this. To do this, we need to tie the color of the circles to the organic matter level. We need to create a palette, which we can do with the *colorBin()* function.

My favorite palette is the red-yellow-green palette we are used to in yield maps. We will load the *RColorBrewer* package, which defines both individual colors and palettes. We will then tell R in the *colorBin* command to use this palette.

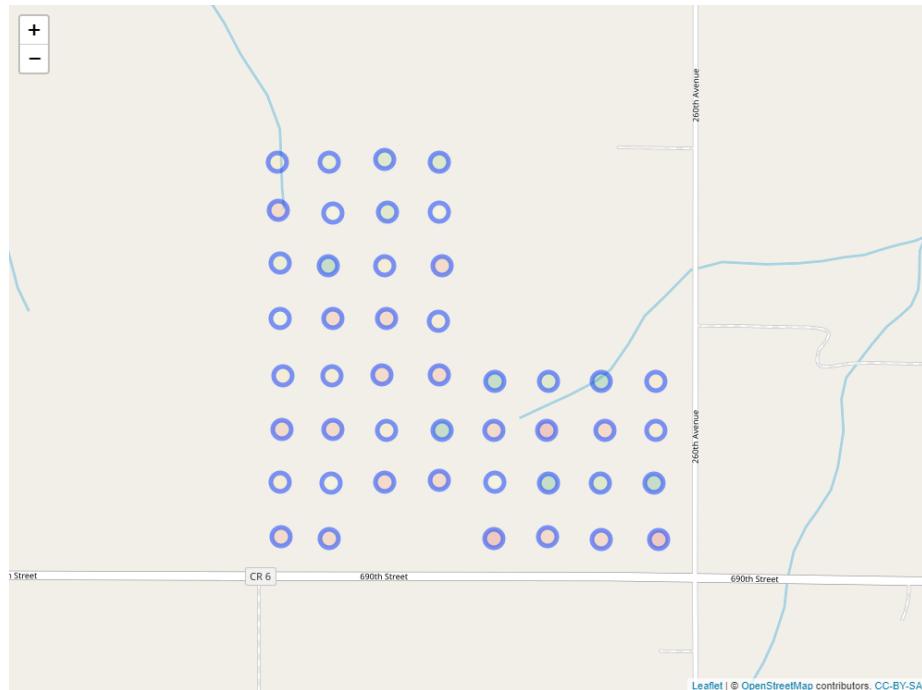
```
library(RColorBrewer)
palette_om = colorBin(palette = "RdYlGn", single_attribute_data$measure)
```

We can then color code our points by adding the *fillColor* argument to our plot code.

```
library(leaflet)

m <- single_attribute_data %>%    # starts with om dataset
  leaflet() %>%      # starts leaflet
  addCircleMarkers(
    fillColor = ~palette_om(measure)
  ) %>%      # adds markers as points
  addTiles()

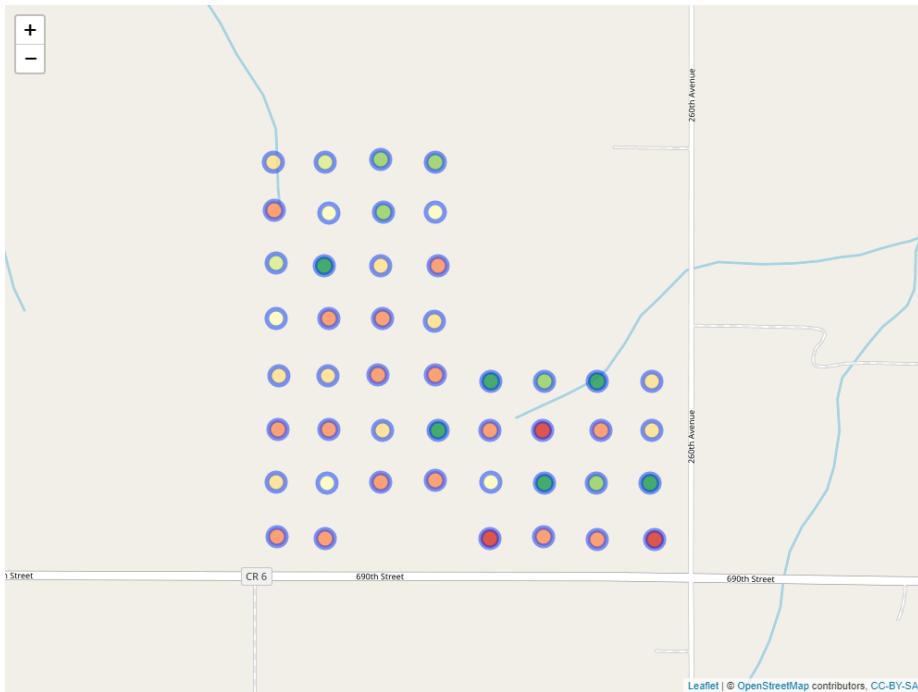
saveWidget(m, "temp.html", selfcontained = FALSE)
webshot("temp.html", cliprect = "viewport")
```



Let's make a few more tweaks. First, the colors are so transparent the wash out, so let's increase their opacity using the argument of that name.

```
m <- single_attribute_data %>%    # starts with om dataset
  leaflet() %>%      # starts leaflet
  addCircleMarkers(
    fillColor = ~palette_om(measure),
    fillOpacity = 0.8
  ) %>%      # adds markers as points
  addTiles()

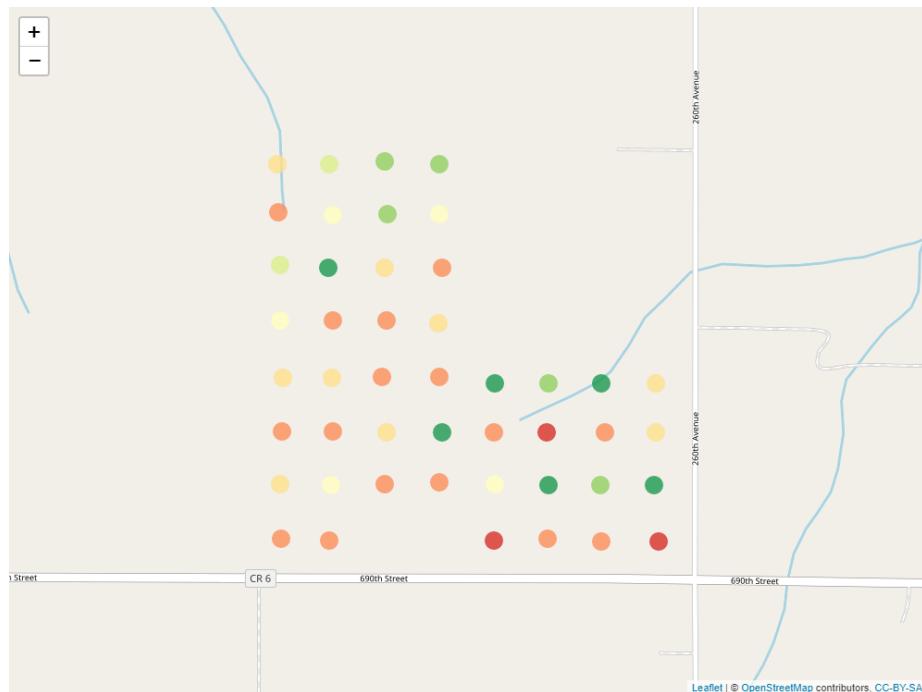
  saveWidget(m, "temp.html", selfcontained = FALSE)
  webshot("temp.html", cliprect = "viewport")
```



Next, the blue border detracts from the cells. We can set its weight to zero so it disappears.

```
m <- single_attribute_data %>%    # starts with om dataset
  leaflet() %>%      # starts leaflet
  addCircleMarkers(
    fillColor = ~palette_om(measure),
    fillOpacity = 0.8,
    weight = 0
  ) %>%    # adds markers as points
  addTiles()

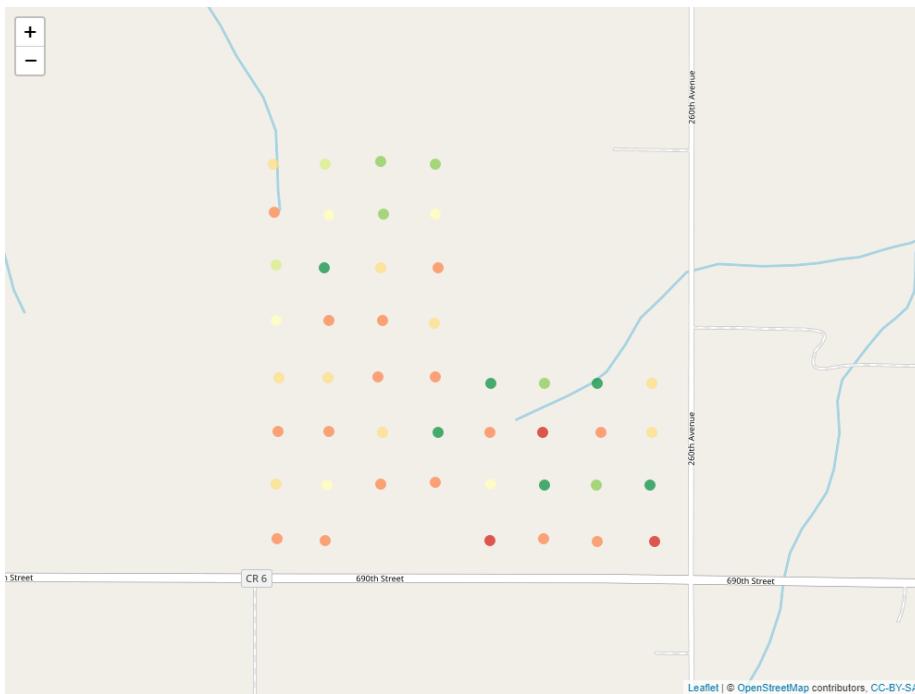
saveWidget(m, "temp.html", selfcontained = FALSE)
webshot("temp.html", cliprect = "viewport")
```



Let's make the points a little smaller using the *radius* argument, so they don't so overwhelm the field.

```
m <- single_attribute_data %>%    # starts with om dataset
  leaflet() %>%      # starts leaflet
  addCircleMarkers(
    fillColor = ~palette_om(measure),
    fillOpacity = 0.8,
    weight = 0,
    radius=6
  ) %>%      # adds markers as points
  addTiles()

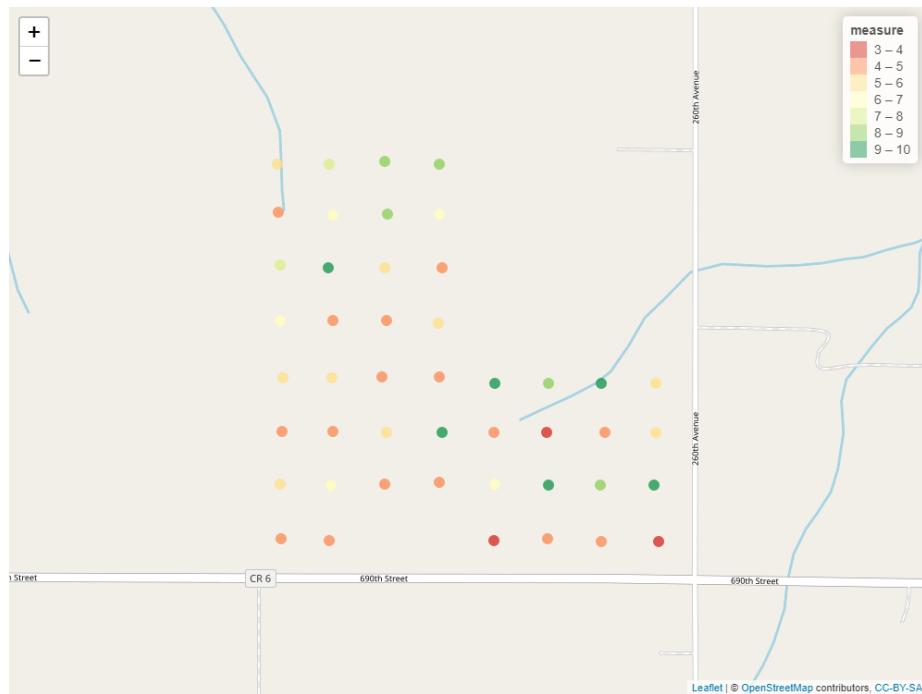
  saveWidget(m, "temp.html", selfcontained = FALSE)
  webshot("temp.html", cliprect = "viewport")
```



Finally, let's add a legend. We provide a minimum of two arguments to it. First *values* tells it that the legend will correspond to the values of measure in our dataset. Second, *pal* = tells R to use the palette we developed for organic matter.

```
m <- single_attribute_data %>%    # starts with om dataset
  leaflet() %>%      # starts leaflet
  addCircleMarkers(
    fillColor = ~palette_om(measure),
    fillOpacity = 0.8,
    weight = 0,
    radius=6
  ) %>%    # adds markers as points
  addTiles() %>%
  addLegend(values = ~measure,
            pal = palette_om)

saveWidget(m, "temp.html", selfcontained = FALSE)
webshot("temp.html", cliprect = "viewport")
```



There are thousands of other ways to modify this plot, but this little bit should get you far!

#### 12.4.2 Practice

Using the code above, plot other soil variable. Run the code below to see a list of the many different attributes you can examine.

```
unique(soil_test_data$attribute)
```

```
## [1] "Om"          "Ph"          "Bph"          "No3"          "Salinity"     "P"
## [7] "P_bray"      "P_olsen"      "M3_p"         "M3icp_p"      "K"            "Ca"
## [13] "Mg"          "Na"          "S"            "Zn"           "Cu"           "Mn"
## [19] "Fe"          "B"           "Cec"          "Ca_sat"       "Mg_sat"       "K_sat"
## [25] "H_sat"
```

Experiment with plugging and playing with the above code. (There is *never* shame in re-using code!!!). Play with the settings for fillOpacity, weight, and radius until they are to your liking. Also, try some different palettes: “RdBu”, “PRgn”, or others (list available at <https://www.r-graph-gallery.com/38-rcolorbrewers-palettes.html>).

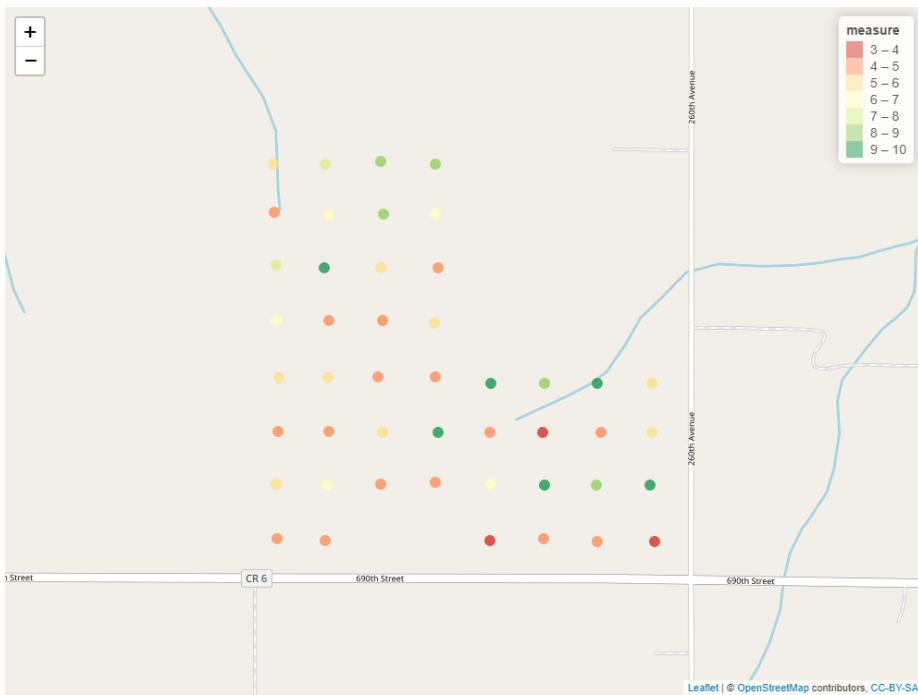
```

palette_soil = colorBin(palette = "RdYlGn", single_attribute_data$measure)

m <- single_attribute_data %>%    # starts with om dataset
  leaflet() %>%      # starts leaflet
  addCircleMarkers(
    fillColor = ~palette_soil(measure),
    fillOpacity = 0.8,
    weight = 0,
    radius=6
  ) %>%    # adds markers as points
  addTiles() %>%
  addLegend(values = ~measure,
            pal = palette_soil)

saveWidget(m, "temp.html", selfcontained = FALSE)
webshot("temp.html", cliprect = "viewport")

```



## 12.5 Exercise: SSURGO

In this lesson, we saw how the Soil Survey Geographic Database (SSURGO) database can be accessed to gain further insight into fields.

This isn't an exercise so much as a demonstration how those data can be obtained and plotted through R. It is here for your use or reference, but not a required part of this lesson.

We can load our shapefile just as we have done before. For reasons explained below, it is important to name the dataset "field".

```
library(sf)
library(tidyverse)
library(leaflet)

field = st_read("data-unit-12/exercise_data/yield.shp", quiet=TRUE)
```

First, let's create a yield map, just as we learned to in the "shapefile" exercise. If you have not completed that exercise, please do so before continuing.

```
palette_yield = colorBin("RdYlGn", field$Yld_Vol_Dr)

m <- field %>%    # starts with our dataset
  leaflet() %>%      # starts leaflet
  addCircleMarkers(
    fillColor = ~palette_yield(Yld_Vol_Dr),
    fillOpacity = 0.8,
    weight = 0,
    radius=0.1
  ) %>%      # adds markers as points
  addTiles() %>%
  addLegend(values = ~Yld_Vol_Dr,
            pal = palette_yield)

saveWidget(m, "temp.html", selfcontained = FALSE)
webshot("temp.html", cliprect = "viewport")
```



This is the one and only time in this course when I will ask you just to run a chunk of code without explaining it line-by-line. As long as your spatial feature object (the field shapefile) is named “field”, just run this chunk as it is.

##### Run this as-is: Do not modify !! #####

```
library(FedData)
library(sp)
soil_data_sp = as(field, "Spatial")
soil_map = get_ssurgo(soil_data_sp, "example")
soil_sf = st_as_sf(soil_map$spatial) %>%
  st_set_crs(4326)

mu_point = soil_map$tabular$mapunit %>%
  dplyr::select(musym, muname, mukey)

# get measures of sand, silt, clay, etc
horizon_diagnostics = soil_map$tabular$chorizon %>%
  dplyr::select(hzname, sandtotal.r, silttotal.r, claytotal.r, om.r, awc.r, ksat.r, cec7.r, cokey)

# soil texture class
```

```

texture = soil_map$tabular$chtexturegrp %>%
  filter(rvindicator=="Yes") %>%
  dplyr::select(texdesc, chkey)

# percent slope and drainage class
slope = soil_map$tabular$component %>%
  dplyr::select(drainagecl, slope.r, mukey, cokey)

parent_material = soil_map$tabular$copm %>%
  filter(pmorder==1 | is.na(pmorder)) %>%
  dplyr::select(pmkind, copmgrpkey, copmkey)

pm_group = soil_map$tabular$copmgrp %>%
  dplyr::select(cokey, copmgrpkey)

component = soil_map$tabular$component %>%
  dplyr::select(mukey, cokey)

all_soil_data = mu_point %>%
  left_join(component) %>%
  left_join(horizon_diagnostics) %>%
  group_by(mukey) %>%
  filter(chkey==min(chkey)) %>%
  ungroup() %>%
  left_join(texture) %>%
  left_join(slope) %>%
  left_join(pm_group) %>%
  left_join(parent_material) %>%
  dplyr::select(-c(copmgrpkey, copmkey, musym, cokey))

complete_soil_data = soil_sf %>%
  rename_all(tolower) %>%
  mutate(mukey = as.numeric(mukey)) %>%
  left_join(all_soil_data)

```

The soil variables included in the *complete\_soil\_data* dataset are:

“sandtotal.r” - Percent Sand “silttotal.r” - Percent Silt “claytotal.r” - Percent Clay “om.r” - Percent Organic Matter “awc.r” - Available Water Holding Capacity “ksat.r” - Saturated Hydraulic Conductivity (drainage) “cec7.r” - Cation Exchange Capacity “texdesc” - Soil Textural Class (based on percent sand, silt, and clay) “drainagecl” - Drainage Class (based on hydraulic conductivity) “slope.r” - Slope (percent change per 100 ft travelled)

We can plot any of the above variables by referencing them as “*complete\_soil\_data\$*” plus the variable of interest.

We can then create a leaflet plot as we have elsewhere in this unit.

```
library(grDevices)
pal_om = colorNumeric("RdYlGn", complete_soil_data$om.r)

m <- complete_soil_data %>%
  leaflet() %>%
  addCircleMarkers(
    radius = 4,
    weight=0,
    fillColor = ~ pal_om(om.r),
    opacity = 0.8,
    popup = as.character(complete_soil_data$om.r)
  ) %>%
  addTiles()

saveWidget(m, "temp.html", selfcontained = FALSE)
webshot("temp.html", cliprect = "viewport")
```



## 12.6 Exercise: Rasters

In the lecture, we learned that rasters were organized within a grid system, with each cell representing a unique value. Cells may be colored according to which “bin” their value falls, in order to facilitate the identification of spatial patterns.

### 12.6.1 Case Study: Soil Test Data

The following data were collected from a field in Minnesota. We will create a lime application map. To start , we need to load two shape files. The first shape file contains our soil samples.

#### 12.6.1.1 Load and Filter Data

```
library(tidyverse)
library(gstat)
library(sp)
library(raster)
library(stars)
library(leaflet)
library(leafem)

point_data = st_read("data-unit-12/Folie N & SE_grid_sample.shp", quiet=TRUE)
head(point_data)

## Simple feature collection with 6 features and 9 fields
## Geometry type: POINT
## Dimension: XY
## Bounding box: xmin: -92.79313 ymin: 43.98697 xmax: -92.78942 ymax: 43.99318
## Geodetic CRS: WGS 84
##   obs Sample_id SampleDate ReportDate P2      Grower      Field attribute
## 1  29          21 10/26/2018 10/30/2018  0 Tom Besch Folie N & SE      Om
## 2  30          22 10/26/2018 10/30/2018  0 Tom Besch Folie N & SE      Om
## 3  31          27 10/26/2018 10/30/2018  0 Tom Besch Folie N & SE      Om
## 4  32           5 10/26/2018 10/30/2018  0 Tom Besch Folie N & SE      Om
## 5  33           6 10/26/2018 10/30/2018  0 Tom Besch Folie N & SE      Om
## 6  34           8 10/26/2018 10/30/2018  0 Tom Besch Folie N & SE      Om
##   measure
## 1      3.2 POINT (-92.78954 43.9931)
## 2      3.8 POINT (-92.7895 43.9926)
## 3      3.3 POINT (-92.78942 43.98697)
```

```
## 4      5.9 POINT (-92.79313 43.99245)
## 5      3.1 POINT (-92.79294 43.99318)
## 6      1.8  POINT (-92.79145 43.9902)
```

We can see this dataset, much contains POINT geometries – that is, each row of data is related to a single georeference.

Our second shape file describes the field boundary. This is very important too – otherwise we would not know what shape and size our raster should be.

```
boundary = st_read("data-unit-12/Folie N &SE_boundary.shp", quiet=TRUE)

head(boundary)

## Simple feature collection with 1 feature and 4 fields
## Geometry type: MULTIPOLYGON
## Dimension: XY
## Bounding box: xmin: -92.79768 ymin: 43.98635 xmax: -92.78861 ymax: 43.99362
## Geodetic CRS: WGS 84
##   FieldName MapsCode    Grower      Field           geometry
## 1 NORTH & SE CIT18NW1 Tom Besch Folie N &SE MULTIPOLYGON (((-92.79234 4...
```

We see the field contains POLYGON geometry and a single row of data, which mainly describes the field name.

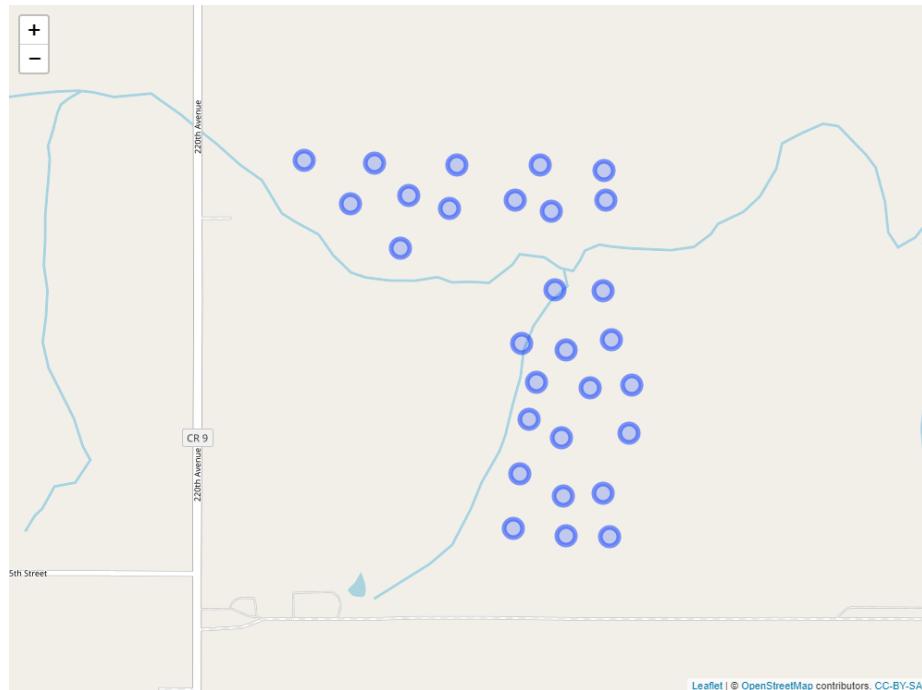
Our last step in this section is to filter the data so they only contain buffer pH data.

```
selected_data = point_data %>%
  filter(attribute=="Bph")
```

Without going through all the trouble of building out a map, we can still check the location of our points using leaflet().

```
m <- selected_data %>%
  leaflet() %>%
  addCircleMarkers() %>%
  addTiles()

saveWidget(m, "temp.html", selfcontained = FALSE)
webshot("temp.html", cliprect = "viewport")
```



### 12.6.1.2 Build a Grid

To create a raster that matches the size and shape of our field, we need to create a frame for R to fill in. We can do this really easily with the following few lines of code from the *stars* package. We will name our new raster “grd” and build it from our boundary dataset. To start, we create a rectangular grid based on the boundary box. This box extends from the point with the lowest X and Y value in a geometry to the point with the greatest X and Y value. We can determine these points for our boundary using the *st\_bbox* function.

```
library(stars)

st_bbox(boundary)

##      xmin      ymin      xmax      ymax
## -92.79767  43.98635 -92.78861  43.99362
```

The *st\_as\_stars()* function converts the shapefile to a grid for us to fill in with our raster. The last line uses the *st\_crop* function to trim our rectangular raster to the shape of our field.

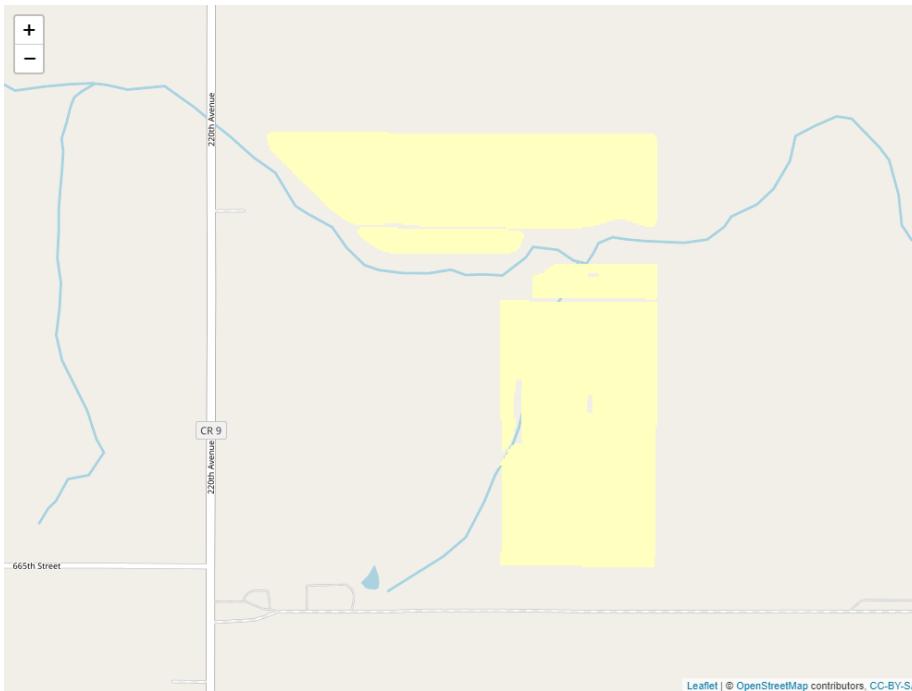
```
library(stars)

### make grid
grd = st_bbox(boundary) %>%
  st_as_stars() %>%
  st_crop(boundary)
```

We can plot the grid using leaflet. We create a generic map using the *addProviderTiles()* function. Lastly, we add the grid using *addStarsImage()*.

```
m <- grd %>%
  leaflet() %>%
  addTiles() %>%
  addStarsImage()

saveWidget(m, "temp.html", selfcontained = FALSE)
webshot("temp.html", cliprect = "viewport")
```



We won't be able to see the individual cells – they are tiny – but we can see it matches the shape of our field.

### 12.6.1.3 Build the Semivariogram

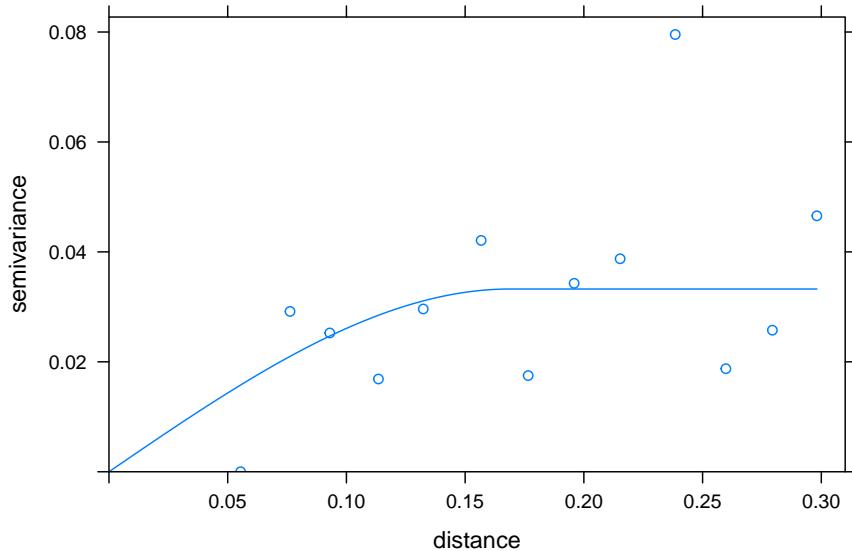
There are three steps to building a semivariogram:

- 1) Define relationship between the measure and distance with the *variogram* function. This is much like defining a linear model before we summarise it.
- 2) Fit variogram model using the *fit.variogram* function from the *gstat* package.
- 3) Visually confirm the model approximately fits the data, by using the *plot* function to show the individual variances (*v*) and the model (*m*).

```
v = variogram(measure~1, selected_data)
m = fit.variogram(v, vgm("Sph"))

## Warning in fit.variogram(v, vgm("Sph")): singular model in variogram fit

plot(v, model = m)
```



The *fit.variogram* function takes two arguments. The first is the name of the data.frame to which the distance, variance (“gamma”), and other statistics should be saved.

The second argument, *vgm*(“Sph”), tells R to fit the variances with a spherical model. There are several different models that could be used, but the spherical model is perhaps the most common.

The model appears to approximately fit the data, so we move to the next step: kriging.

#### 12.6.1.4 Kriging

Kriging is where the map magic happens. We use our variogram model to predict how related each empty cell in the raster is to each filled cell in the raster, based on their distance. The strength of that relationship is then used to weight the filled cells as they are averaged to predict the value of each empty cell.

This magic happens with a single line of code, using the *krige* function of the *gstat* package. There are four arguments to this function:

- 1) formula: the relationship we are modelling. “measure ~ 1” means we are modelling the relationship between our measure value and location. The 1 indicates that the minimum measured value does not have to be equal to zero
- 2) selected\_data: the data that are being kriged
- 3) grd: the object we created above that contains the framework or template for the raster we want to create
- 4) model=m: tells R to use the semivariogram model, m, we created and inspected above

```
library(gstat)
kriged_data = gstat::krige(formula = measure~1, selected_data, grd, model=m)
```

```
## [using ordinary kriging]
```

The object that is returned is a list. The predicted values are found in the first element of that list. For the sake of clarity, let’s extract that part of the list and rename it.

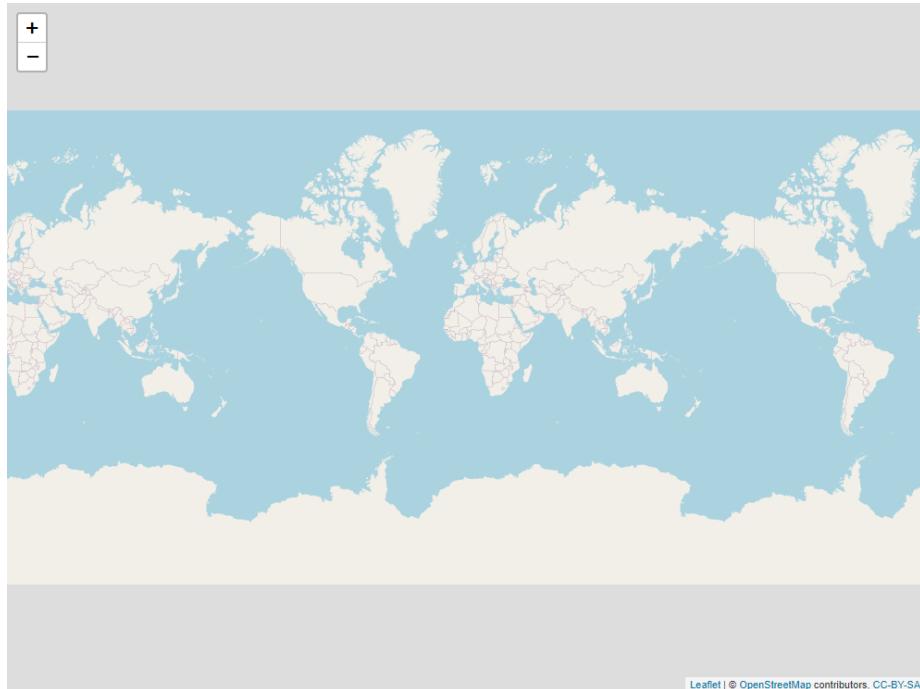
```
finished_data = kriged_data[1]
```

#### 12.6.1.5 Mapping Kruged Data

We will again use leaflet to map our data. We will start out by creating our map and adding provider tiles.

```
m <- finished_data %>%
  leaflet() %>%
  addTiles()
```

```
saveWidget(m, "temp.html", selfcontained = FALSE)
webshot("temp.html", cliprect = "viewport")
```



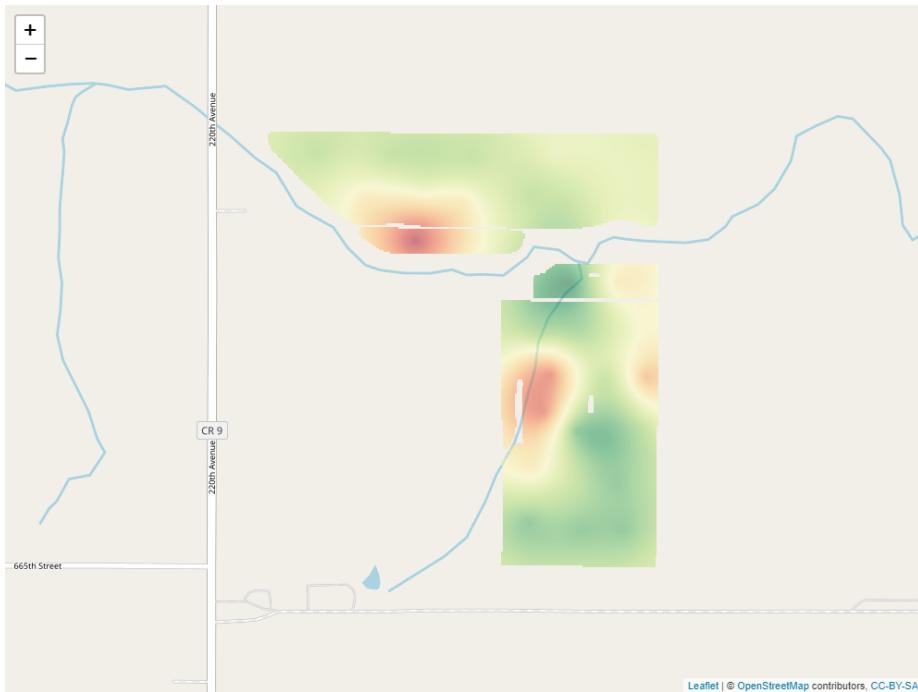
We can now add our raster (stars) image to the map using `addStarsImage`. We will set the `opacity=0.5` so we can see both the field and the test zones.

We will also set the colors to `RdYlGn`; we need to load the *RColorBrewer* package to use that palette.

```
library(RColorBrewer)

m <- finished_data %>%
  leaflet() %>%
  addTiles() %>%
  addStarsImage(opacity = 0.5,
                colors="RdYlGn")

saveWidget(m, "temp.html", selfcontained = FALSE)
webshot("temp.html", cliprect = "viewport")
```



#### 12.6.1.6 Building the Recommendation

Our recommendation only requires two more lines of code. We will create a new column, “rec”, which will consist of the predicted lime application rate based on buffer pH. If we look at University of Minnesota recommendations, we can see that the recommended rate of 100% effective neutralizing lime is equal to;

$$rate = 37000 - bpH \times 5000$$

<https://extension.umn.edu/liming/lime-needs-minnesota>

So lets go ahead and do this. We will plug “var1.pred” – the generic raster name for the predicted variable – into the equation above and solve for the recommended rate.

```
finished_data$rec = (37000 - finished_data$var1.pred*5000)
```

One last thing: if you look at the recommendation table, however, you see the recommended rate is not given for buffer pH above 6.8. Thus, if the recommendation is less than 3000, we should set it to zero. Thus we need to use another function, *if\_else*.

The *if\_else* function defines the value of a variable, based on three arguments:

- 1) the condition to be tested
- 2) the value of the variable if the condition is true
- 3) the value of the variable if the condition is false

```
finished_data$rec = if_else(finished_data$rec<3000, 0, finished_data$rec)
```

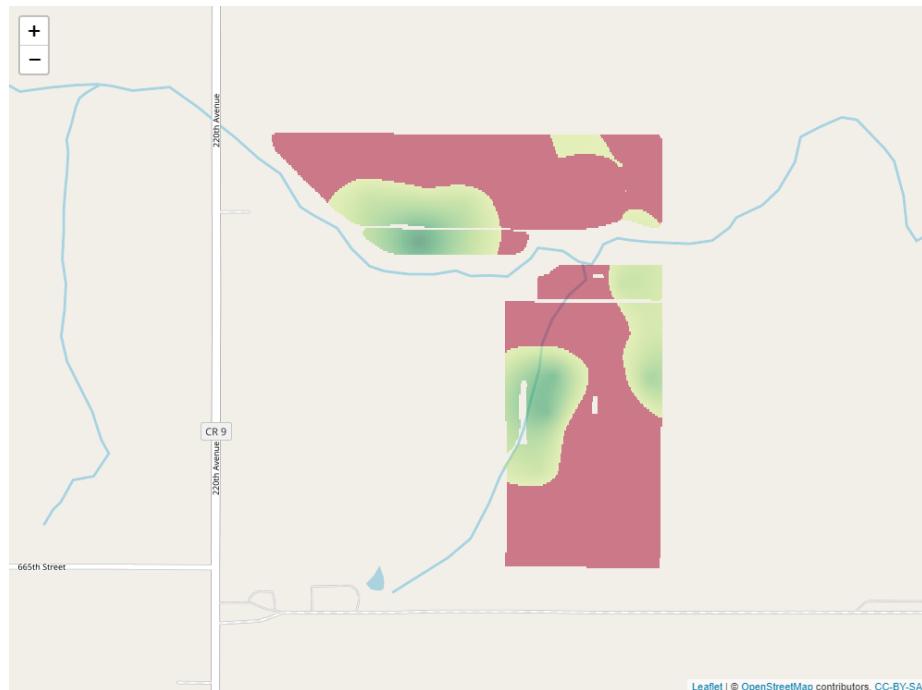
In the line above, we revise the value of the rec column. If it is less than 3000, we set the rec to zero. If it is 3000 or greater, we use it as-is.

#### 12.6.1.7 Mapping the Recommendation

Finally, we can map the recommendation, similar to how we mapped the test levels above.

```
m <- leaflet(finished_data["rec"]) %>%
  addTiles() %>%
  addStarsImage(opacity = 0.5,
                colors = "RdYlGn")

saveWidget(m, "temp.html", selfcontained = FALSE)
webshot("temp.html", cliprect = "viewport")
```



The green areas above are the ones we should “Go” on – lime should be applied to them.

### 12.6.2 Practice 1

Take the map above and see if you can create soil K raster for the data above.

- 1) change the selected data below from “Bph” to “K”.

```
selected_data = point_data %>%
  filter(attribute=="Bph")
```

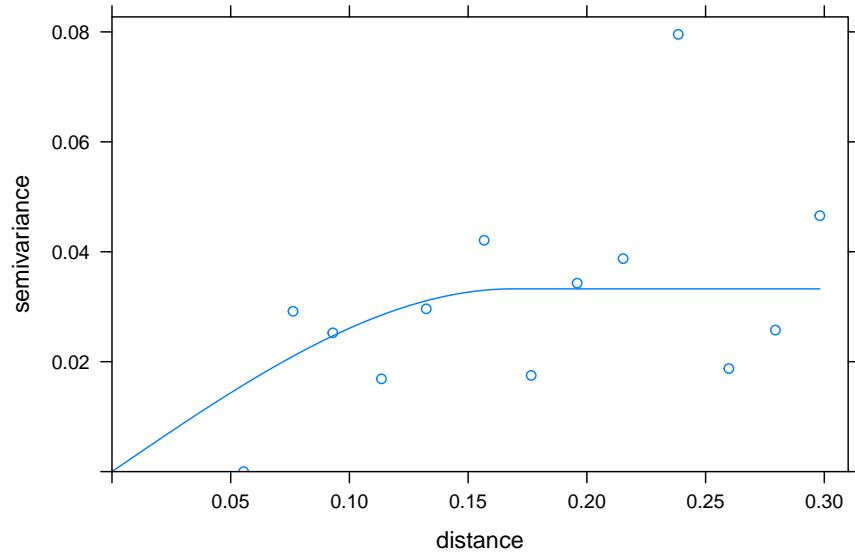
- 2) Create the raster framework.

```
### make grid
grd = st_bbox(boundary) %>%
  st_as_stars() %>%
  st_crop(boundary)
```

- 3) Create and examine the variogram. The model should roughly fit the variances.

```
v = variogram(measure~1, selected_data)
m = fit.variogram(v, vgm("Sph"))
```

```
## Warning in fit.variogram(v, vgm("Sph")): singular model in variogram fit
plot(v, model = m)
```



- 4) Krige the data and extract the predicted (or interpolated) values,

```
kriged_data = gstat:::krige(formula = measure~1, selected_data, grd, model=m)

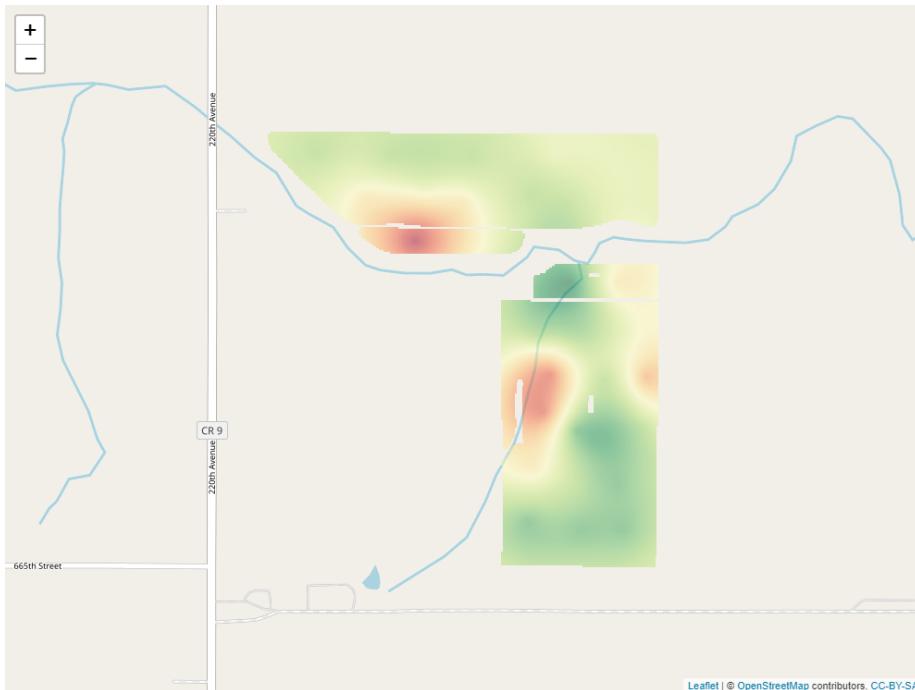
## [using ordinary kriging]

finished_data = kriged_data[1]
```

- 5) Create the map of soil K values!

```
m <- finished_data %>%
  leaflet() %>%
  addTiles() %>%
  addStarsImage(opacity = 0.5,
                colors="RdYlGn")

saveWidget(m, "temp.html", selfcontained = FALSE)
webshot("temp.html", cliprect = "viewport")
```



- 6) Calculate recommended application rates, assuming a 200 bushel/acre yield target. Change the code below accordingly. The K recommendation formula is:

$$\text{K recommendation} = (1.12 - 0.0056 \text{finished\_data\$var1.pred})200$$

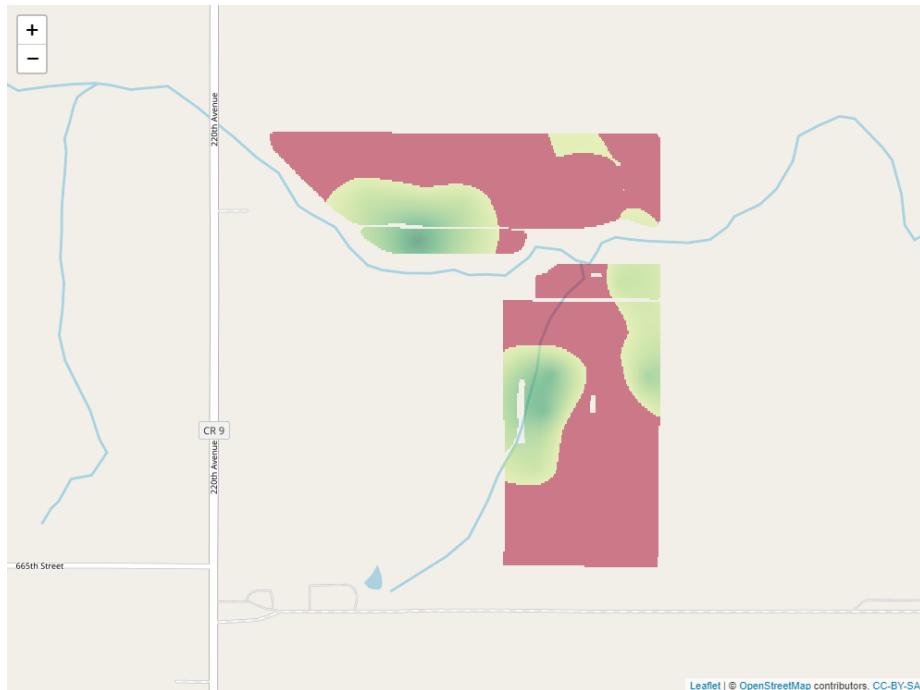
just plug that into the first line of code. K is not recommended for locations that had a soil test greater than 200 ppm, so adjust your recommendation accordingly using the second line of code.

```
finished_data$rec = (37000 - finished_data$var1.pred*5000)
finished_data$rec = if_else(finished_data$rec<3000, 0, finished_data$rec)
```

- 7) Plot the recommended rates:

```
m <- leaflet(finished_data["rec"]) %>%
  addTiles() %>%
  addStarsImage(opacity = 0.5,
                colors = "RdYlGn")

saveWidget(m, "temp.html", selfcontained = FALSE)
webshot("temp.html", cliprect = "viewport")
```



### 12.6.3 Practice 2

Take the map above and see if you can create soil test pH raster for the data above. You should only need to change “selected\_data” to select “Ph”. There is no recommendation rate to create.

- 1) Filter the attribute for “pH”.

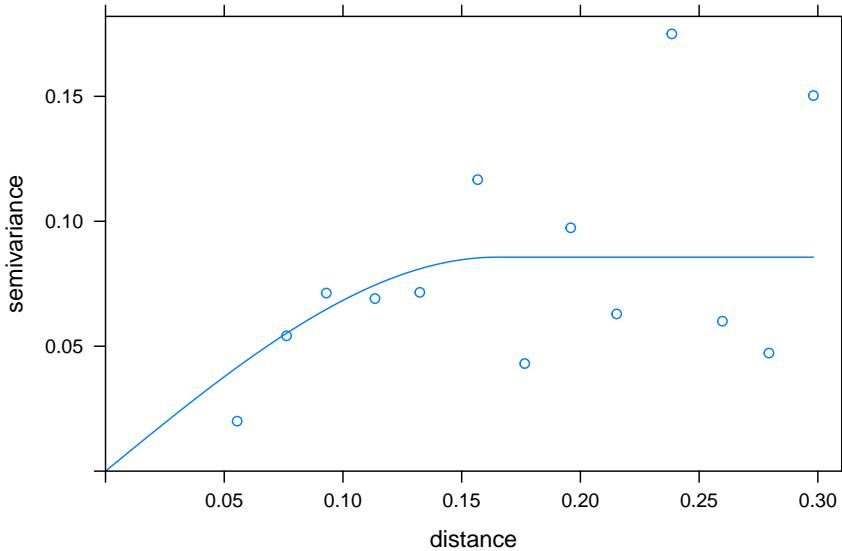
```
selected_data = point_data %>%
  filter(attribute=="Ph")
```

- 2) Create the raster framework.

```
### make grid
grd = st_bbox(boundary) %>%
  st_as_stars() %>%
  st_crop(boundary)
```

- 3) Create and examine the variogram. The model should roughly fit the variances.

```
v = variogram(measure~1, selected_data)
m = fit.variogram(v, vgm("Sph"))
plot(v, model = m)
```



- 4) Krige the data and extract the predicted (or interpolated) values.

```
kriged_data = gstat::krige(formula = measure~1, selected_data, grd, model=m)

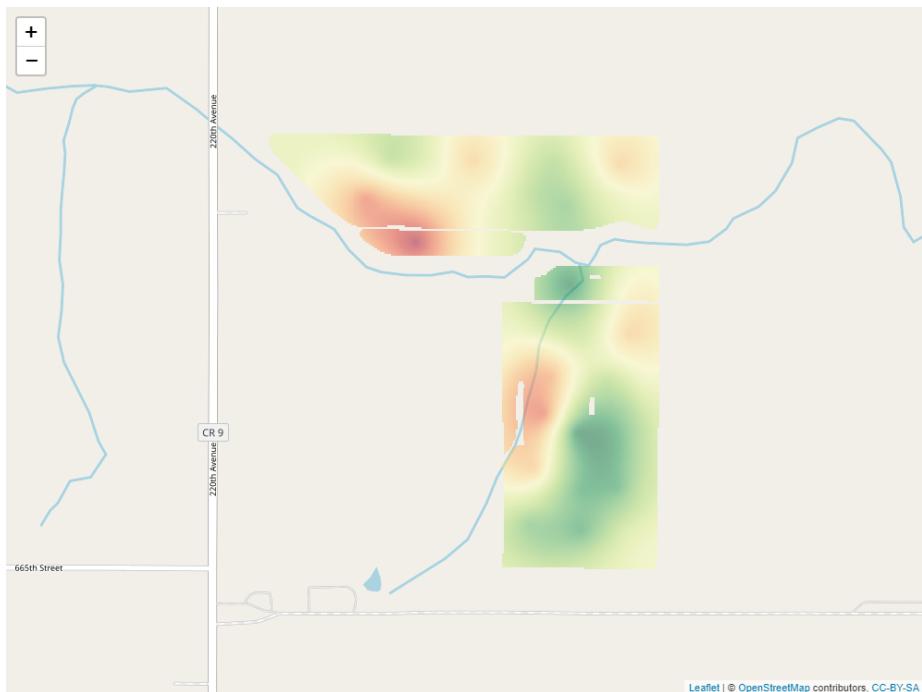
## [using ordinary kriging]

finished_data = kriged_data[1]
```

- 5) Create the map of soil Ph values!

```
m <- finished_data %>%
  leaflet() %>%
  addTiles() %>%
  addStarsImage(opacity = 0.5,
                colors="RdYlGn")

saveWidget(m, "temp.html", selfcontained = FALSE)
webshot("temp.html", cliprect = "viewport")
```



## Chapter 13

# Machine Learning

Machine Learning is an advanced data science topic that I have struggled over whether to include in this text. On the one hand, it is not generally considered part of classical statistics. In addition, unless your plans include becoming a data analyst, you are unlikely to run these analyses on your own. On the other hand, machine learning is playing an increasing role in agricultural (and all areas of) data. It is a topic in which I have been immersed during the past couple of years.

Most importantly, if you are using tools to select hybrids, adjust nitrogen rates, or predict yield, those tools likely incorporate aspects of machine learning. In your daily lives, machine learning determines what advertisements you see, your social media feed, even the results from your search engines. This lesson emphasizes literacy about machine learning – there are no coding exercises included.

### 13.1 Machine Learning

To start with, why is it called “machine learning”? It sounds like statistics-meets-steampunk. To answer this, think about a machine that learns. In other words: a robot. In machine learning, we use complex algorithms that, in theory, could be used by artificial intelligence to learn about their environment. There is a strong predictive component to this: the machine could learn to predict the outcomes of future events by building models based on data it has already obtained. Hello, Skynet.

This sword, however, is also a plowshare. Machine learning can help us understand how many variables can simultaneously predict outcomes, in agriculture, medicine, and climate. Furthermore, machine learning takes a different approach to models than the methods we have learned earlier. In previous units, we have learned testing methods that were *parameteric*. We defined a linear

model, then used our tests (which were all variations of regression) to parameterize it. Parameteric => parameter.

We've also learned that parametric methods have their challenges. One challenge is non-normal or skewed data. Another is heterogeneity of variances. These challenges required us to transform the data, a step which can generate confusion in analyzing and summarizing data. We were also warned when working with multiple linear regression models to beware of multicollinearity (correlations between independent variables) and heteroscedasticity (unequal variances).

Machine learning uses *nonparametric* tests. As the name suggests, these do not use parameters. Instead of regression models, machine learning tools use logical tests (for example, does an observed value fall in a range) to "decide" what value to predict. They measure the "similarity" between two locations to decide whether an observation in one location is likely to occur in the other. They figure out how to group similar observations into groups or "clusters".

In this unit, we will study three examples of machine learning. First, we will see how cluster analysis might be used to divide a sales territory into multiple regions, based on environment. Second, we will learn how to use *nearest-neighbor* analysis to predict yield for a given county, given its similarity to other counties. Finally, we will learn how *classification trees* can be used to explain yield response to multiple environmental factors.

This unit will be more focused on what I call "data literacy" than execution. I want you to be aware of these three tools, which are being used around you in the agronomy world. I want you to be able to speak of them, to ask questions, if engaged. I will resist going deep into the weeds with them, however, because they are often best suited to hundreds (ok) or many thousands (even better) of data points. It is unlikely you will use them in your Creative Component, or even your day-to-day research.

That said, just a few years ago, neither did I. Awareness and curiosity in data science are always good – there is always a more powerful way to address our questions.

## 13.2 Cluster Analyses

Cluster analysis takes a set of individuals (sometimes called examples in this analyses) and divides it into a set number of groups, based on the similarities between the individuals. These groups are defined such that the individuals within a group are very similar to each other, but very different from individuals in other groups.

By itself, clustering doesn't answer any questions about the values of the individuals in our data set. Unlike a classification tree or a k-Nearest Neighbor

analysis, it cannot be used to interpolate values between data points, or predict the value of a future observation.

To further illustrate cluster analysis, it is described in the data science community as “unsupervised learning.” This means that we do not begin with a variable of interest, nor a hypothesis about how changes in one variable relate to changes in another.

Why would we want to cluster data? The reason is that sometimes, when we are dealing with many variables, our analyses may become more intuitive if there are ways to break data down into groups. Clustering is a way of reducing or categorizing our data so it becomes less-overwhelming for us to deal with.

### 13.2.1 Case Study: Grouping Midwestern Environments

For example, anytime we are working with environmental data, we have potentially enormous datasets. Rather than try juggle the meanings of all these quantitative variables in our heads, we may want to describe the data as a series of groups. For example, we might say that a county in South Dakota is cold and dry, a county in northern Illinois is moderate temperature and moderate wetness, and a farm in western Ohio is warm and rainy. Great, we now have three groups. But how would other counties in the midwest sort into these three categories?

For an agronomic testing program, this is a very important question. If the intent is to address the individual needs of three different environments, we need to define those regions so that a sufficient, but not excessive, number of research locations can be designated within each environment.

Lets start out with dataset of county environments. The county environments are mostly soil data, which are not expected to change (outside of human influence) in the short term. Two data are climatic: growing degree days (gdd) and precipitation (ppt). Their values in this dataset reflect a 20-year mean.

Below I have plotted the mean annual precipitation (in millimeters). We see that precipitation is lowest in the Dakotas and generally increases as we move south and east.

```
library(readr)
library(tidyverse)

## -- Attaching packages ----- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v dplyr    1.0.8
## v tibble   3.1.6     v stringr  1.4.0
## v tidyverse 1.2.0    vforcats  0.5.1
## v purrr    0.3.4
```

```
## Warning: package 'ggplot2' was built under R version 4.1.3

## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

library(sf)

## Linking to GEOS 3.9.1, GDAL 3.2.1, PROJ 7.2.1; sf_use_s2() is TRUE

library(leaflet)
library(RColorBrewer)

## Warning: package 'RColorBrewer' was built under R version 4.1.3

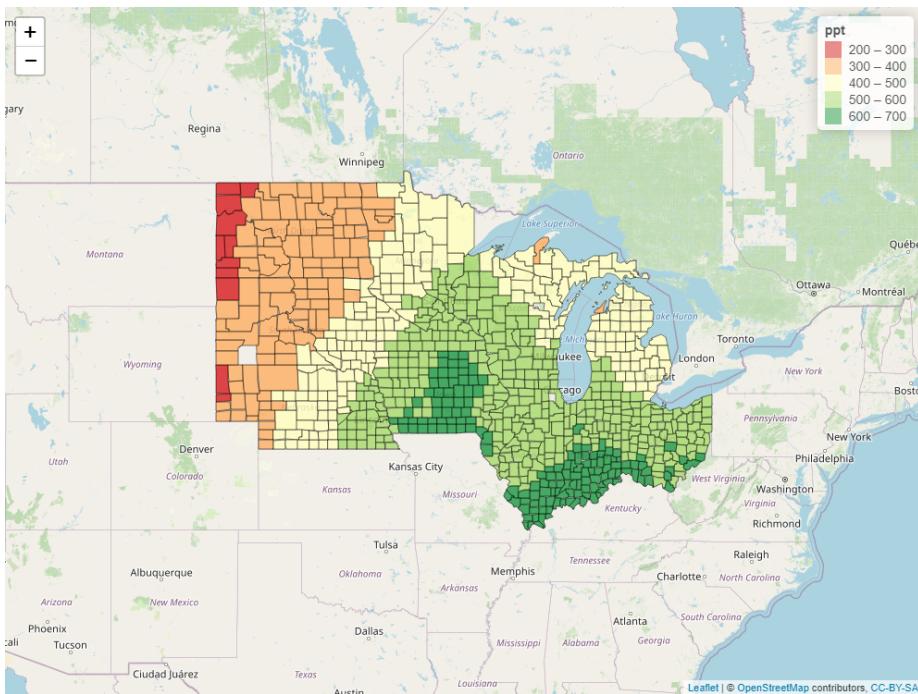
library(htmlwidgets)
library(webshot2)

clean_data_w_geo <- st_read("data-unit-13/midwest_county_data.shp", quiet=TRUE)

pal_ppt = colorBin("RdYlGn", bins=5, clean_data_w_geo$ppt)

m <- clean_data_w_geo %>%
  leaflet() %>%
  addTiles() %>%
  addPolygons(
    fillColor = ~pal_ppt(ppt),
    fillOpacity = 0.8,
    weight=1,
    color = "black"
  ) %>%
  addLegend(pal = pal_ppt,
            values = ~ppt
  )

#above code won't knit -- created static image for markdown
saveWidget(m, "temp.html", selfcontained = FALSE)
webshot("temp.html", cliprect = "viewport")
```



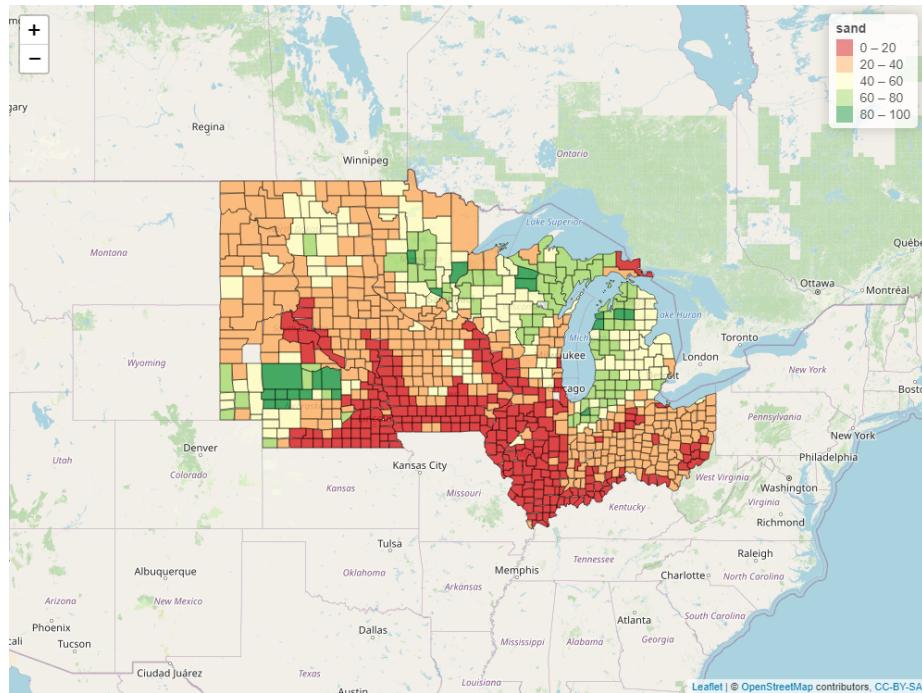
Next up, here is a plot of percentage sand. This is a fascinating map. We can see the path of the glaciers down through Iowa and the Des Moines lobe. Areas with low sand (red below) are largely below the southern reach of the glacier. These areas tend to be high in silt. Conversely, we can see the outwash areas (green below) of the Nebraska Sand Hills, central Minnesota, northern Wisconsin, and Western Michigan, where rapid glacial melting sorted parent material, leaving them higher in sand and rock fragments.

```
pal_sand = colorBin("RdYlGn", bins = 5, clean_data_w_geo$sand)

m <- clean_data_w_geo %>%
  leaflet() %>%
  addTiles() %>%
  addPolygons(
    fillColor = ~pal_sand(sand),
    fillOpacity = 0.8,
    weight=1,
    color = "black"
  ) %>%
  addLegend(pal = pal_sand,
            values = ~sand
  )

```

```
#above code won't knit -- created static image for markdown
saveWidget(m, "temp.html", selfcontained = FALSE)
webshot("temp.html", cliprect = "viewport")
```



We can continue looking at different variables, and through that process might arrive at a general conclusions. Counties that are further south or east tend to receive greater precipitation and more growing degree days (GDD). Counties that are further north and east tend to have more sand and less silt or clay in their soils.

But where do these boundaries end? Say you have a product, like pyraclostrobin, that may reduce heat, and you want to test it across a range of environments that differ in temperature, precipitation and soil coarseness. How would you sort your counties into three groups? This is where cluster analysis becomes very handy.

### 13.2.2 Scaling

When we conduct a cluster analysis, we must remember to scale the data first. Clustering works by evaluating the distance between points. The distance between points  $p$  and  $q$  is calculated as:

$$dist(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_n - q_n)^2}$$

Remember, this distance is Euclidean – it is the same concept by which we calculate the hypotenuse of a triangle, only in this case we are working with a many more dimensions. Euclidean distance is commonly described when we refer to “as the crow flies” – it is the shortest distance between two points.

Cluster analysis will work to minimise the overall distance. If one variable has a much wider range of values than another, it will be weighted more heavily by the clustering algorithm. For example, in our dataset above, growing degree days range from 1351 to 4441, a difference of almost 3100. Precipitation, on the other hand, only ranges from 253 to 694, a different of 341. Yet we would agree the that growing degree days and precipitation are equally important to growing corn!

The solution is to *scale* the data so that each variable has a comparable range of values. Below are the cumulative growing degree days, precipitation, and percent clay values for the first 10 counties in our dataset. You can see how they differ in the range of their values.

```
cluster_vars_only = clean_data_w_geo %>%
  dplyr::select(whc, sand, silt, clay, om, ppt, cum_gdd) %>%
  na.omit()

cluster_vars_only %>%
  filter(row_number() <= 10) %>%
  dplyr::select(cum_gdd, ppt, clay) %>%
  kableExtra::kbl()

## Registered S3 method overwritten by 'webshot':
##   method      from
##   print.webshot webshot2
```

cum_gdd	ppt	clay	geometry
3140.368	598.4478	33.73933	MULTIPOLYGON (((-94.70063 4...
3145.911	609.1156	34.32862	MULTIPOLYGON (((-94.92759 4...
2633.780	599.4642	28.56537	MULTIPOLYGON (((-91.61083 4...
3241.873	659.3385	37.16523	MULTIPOLYGON (((-93.09759 4...
2989.952	598.6462	30.46776	MULTIPOLYGON (((-95.09286 4...
2869.339	606.5850	26.67489	MULTIPOLYGON (((-92.29879 4...
2796.425	612.5765	21.99091	MULTIPOLYGON (((-92.55449 4...
2971.723	611.0333	21.81959	MULTIPOLYGON (((-94.1647 42...
2727.186	618.2398	21.67287	MULTIPOLYGON (((-92.55421 4...
2683.756	620.8813	21.66446	MULTIPOLYGON (((-92.08166 4...

Here are those same 10 counties, with their values for cum\_gdd, ppt, and clay scaled. One of the easiest ways to scale the data are to convert their original

values to Z-scores, based on their normal distributions. We do this exactly the way we calculated Z-scores in Unit 2.

We can see, for each variable, values fall mainly between -1 and 1.

```
cluster_ready_data = cluster_vars_only %>%
  st_drop_geometry() %>%
  as.data.frame() %>%
  scale()

cluster_ready_data %>%
  as.data.frame() %>%
  filter(row_number() <= 10) %>%
  dplyr::select(cum_gdd, ppt, clay)

##      cum_gdd      ppt      clay
## 1  0.55154310  0.8572820  1.1905194
## 2  0.56106037  0.9708866  1.2651893
## 3 -0.31828780  0.8681063  0.5349214
## 4  0.72583066  1.5057241  1.6246202
## 5  0.29327347  0.8593956  0.7759755
## 6  0.08617543  0.9439380  0.2953761
## 7 -0.03902028  1.0077425 -0.2981361
## 8  0.26197313  0.9913093 -0.3198444
## 9 -0.15790581  1.0680532 -0.3384352
## 10 -0.23247627  1.0961831 -0.3395009
```

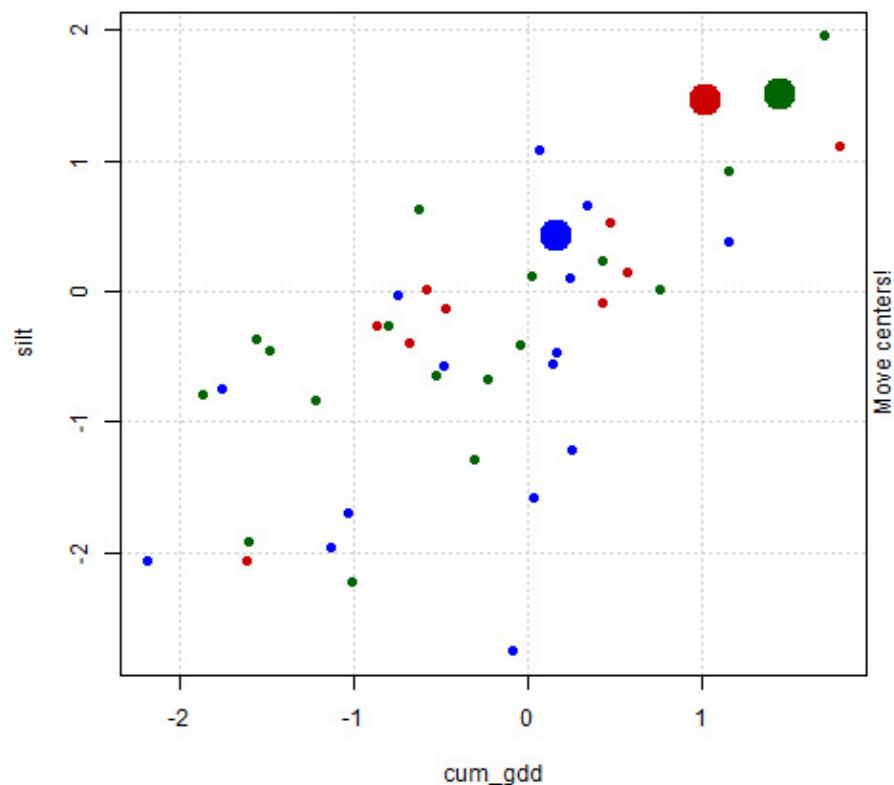
### 13.2.3 Clustering Animation

```
library(animation)
km_animation_data = cluster_ready_data %>%
  as.data.frame() %>%
  sample_n(45) %>%
  dplyr::select(cum_gdd, silt) %>%
  as.matrix()
```

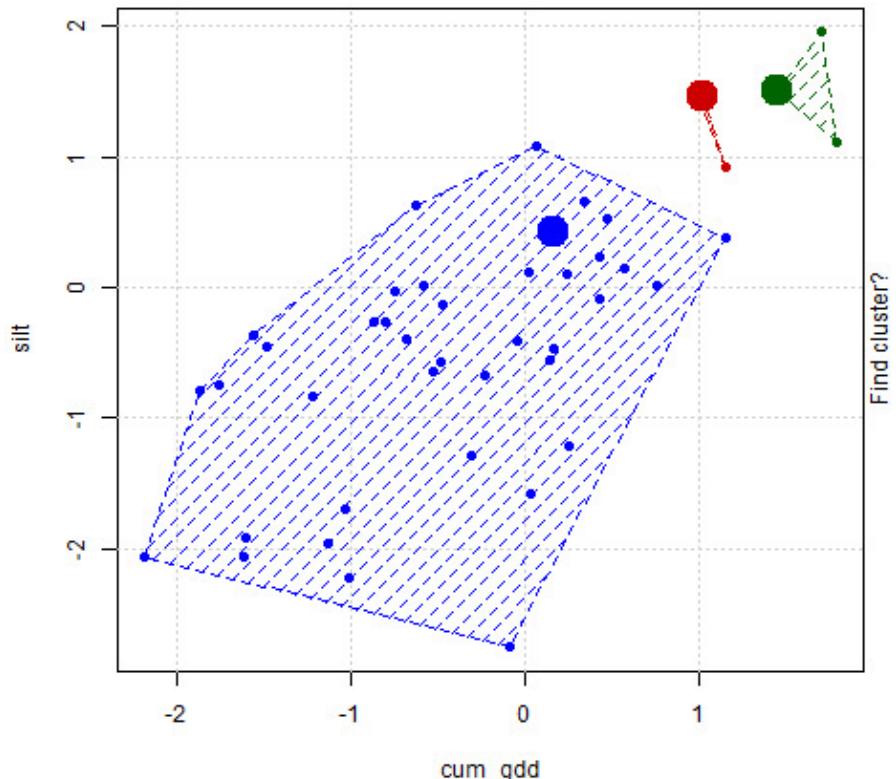
Using two variables, cum\_gdd and silt and the animation below, let's walk through how clustering works.

In the plot above, cum\_gdd is on the X-axis and silt is on the Y-axis. We want to cluster (or divide) our points into three groups. Colors have been assigned randomly to the points (smaller circles). They are scattered instead of being grouped by proximity to each other. After clustering, the points will be colored according to group.

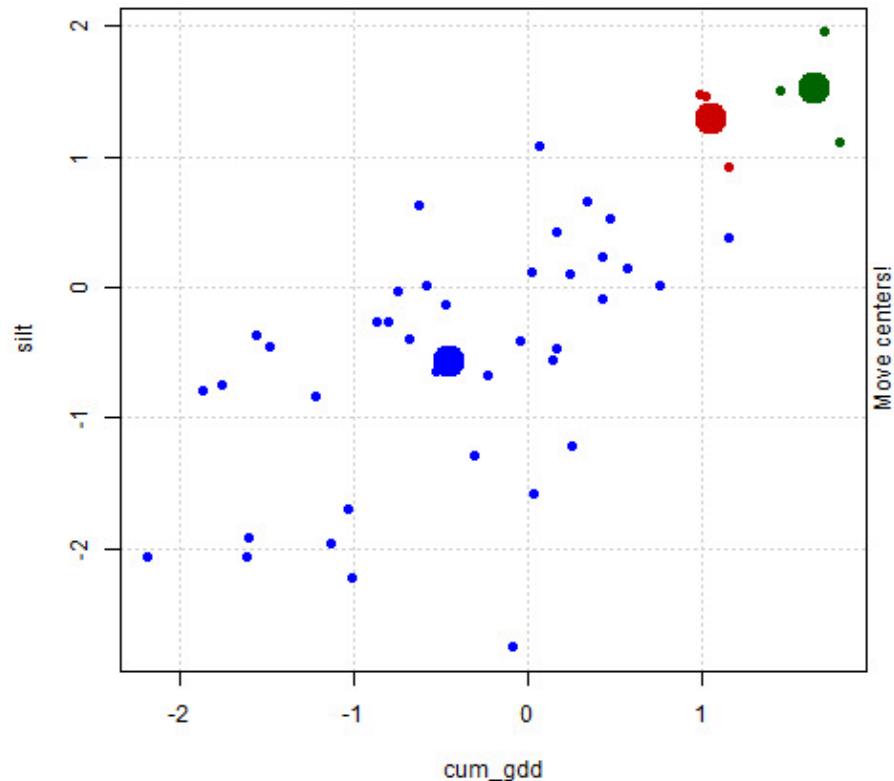
The first step is to randomly place three *centroids* in our plot. These are indicated by the large circles.



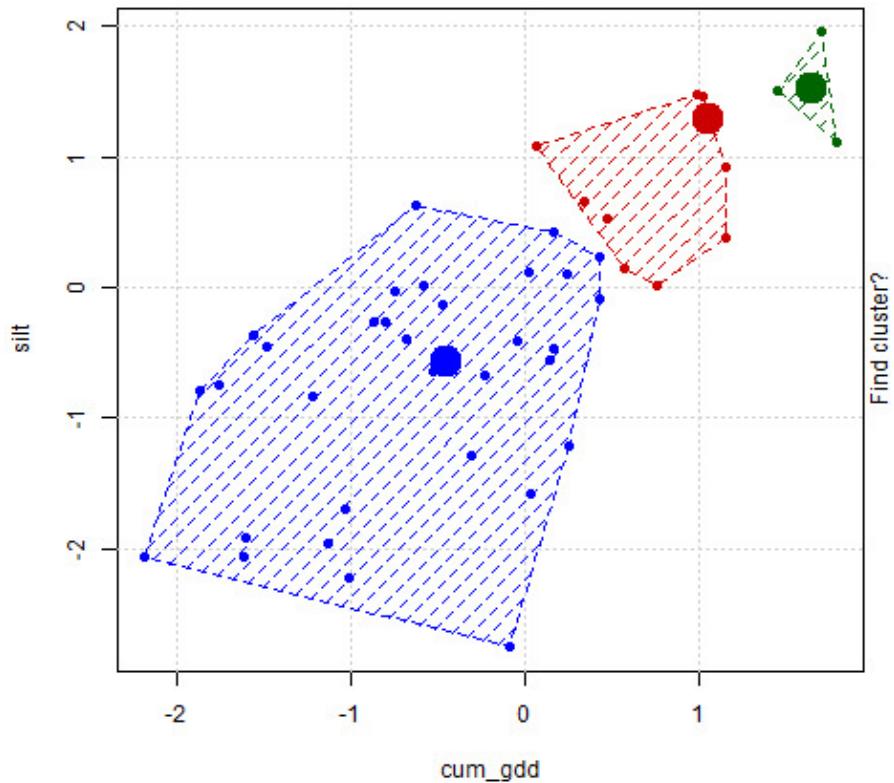
- 2) Next, for each point, we classify each point by its nearest centroid. In the plot below, this is represented by each point taking the same color as its nearest centroid.



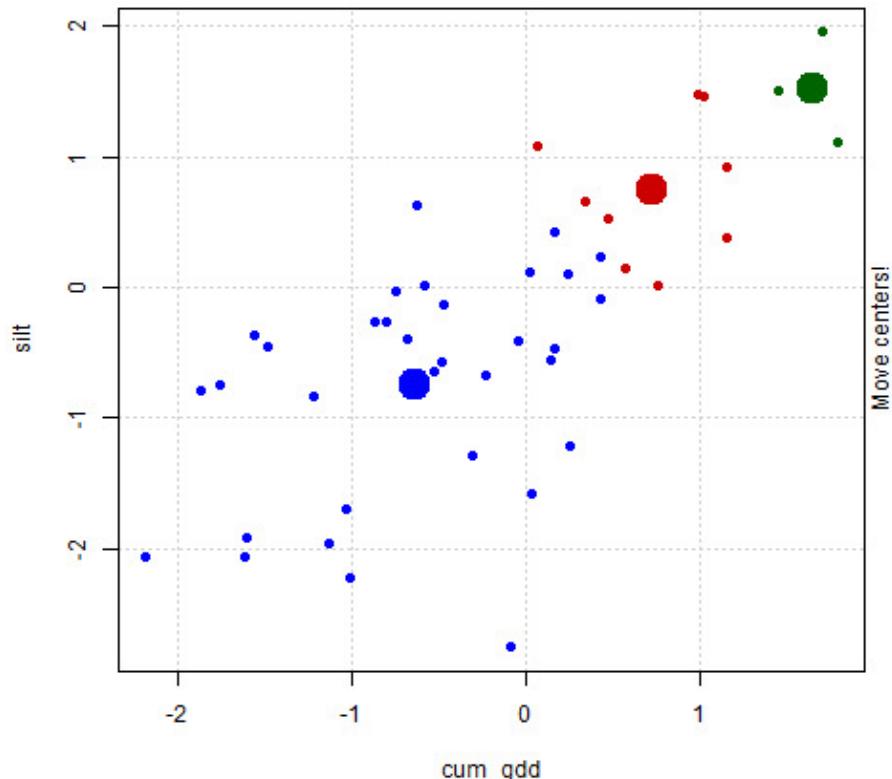
- 3) But now our centroid is not in the center of the group. So we will move it to the center of the group, as seen in the plot below.



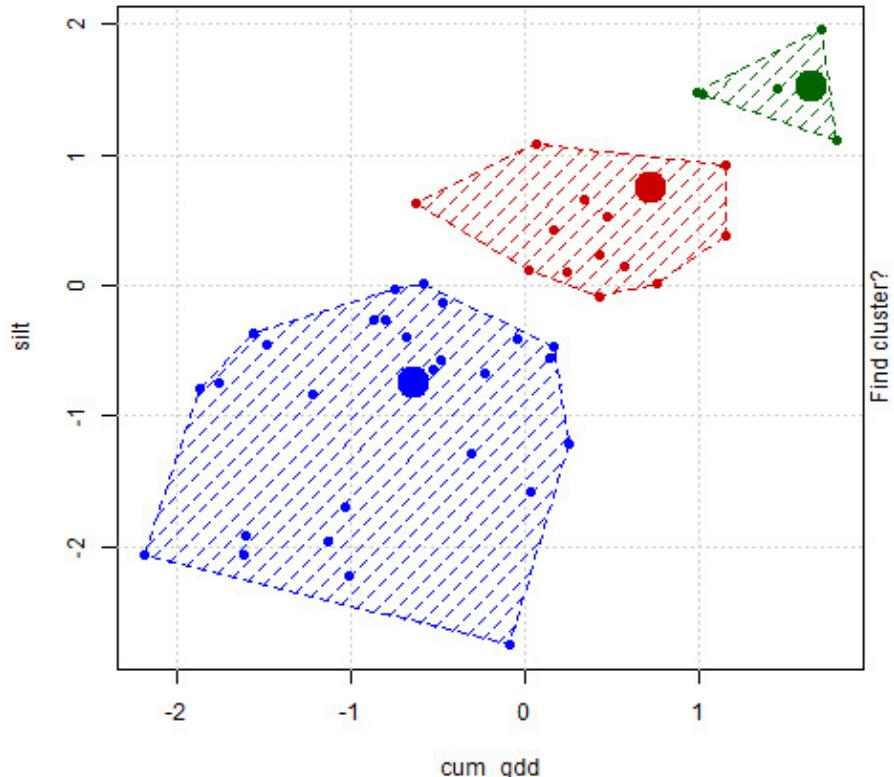
As soon as we move the centroids to the middle of its group, however, a problem occurs. Remember, each point is identified by the nearest centroid. So the groupings of the points again changes. In particular, the number of points in the red grouping increases.



We move our centroids again. You will notice both the red and blue centroids have moved down and to the left.



We reclassify the points. The number of red points has again increased.



This process repeats over and over again until, eventually, there is no more movement in the centroids. At this point, the cluster analysis is said to have converged on its final groupings. The positions of the centroids are typically used to define the new groupings.

```
set.seed(112020)
kmean_silt_gdd = kmeans(km_animation_data, 3)

kmean_silt_gdd$centers %>%
  as.data.frame() %>%
  rownames_to_column("cluster") %>%
  kableExtra::kbl()
```

cluster	cum_gdd	silt
1	-1.3045471	-1.1556767
2	-0.2557197	0.2991182
3	1.0860757	0.9984785

The table above contains the final center estimates from the cluster algorithm. Cluster 1 is warmer and has siltier soil. Cluster 2 has medium numbers of growing degree days and silt. Cluster 3 is cooler and has soils lower in clay.

### 13.2.4 County Cluster Analysis

Now, let's run the cluster analysis with all of our variables. The cluster centers within each variable are shown below.

```
# cluster identification
clusters = kmeans(cluster_ready_data, 3)

clusters$centers %>%
  kableExtra::kbl()
```

whc	sand	silt	clay	om	ppt	cum_gdd
-0.0152348	0.1147455	-0.3219194	0.2808926	0.1693996	-0.7947552	-0.6478638
0.4188589	-0.7234396	0.7811347	0.4051389	-0.4065930	0.7064892	0.7585691
-1.0435272	1.6497023	-1.4394576	-1.5196343	0.7460924	-0.4326339	-0.8192046

```
# classify centers and rejoin with original data
centers = as.data.frame(clusters$centers) %>%
  rownames_to_column("cluster") %>%
  gather(attribute, value, -cluster) %>%
  mutate(named_value = case_when(value < -0.45 ~ "low",
                                   value > 0.45 ~ "high",
                                   value >= -0.45 & value <= 0.45 ~ "ave")) %>%
  dplyr::select(-value) %>%
  spread(attribute, named_value) %>%
  mutate(cluster = as.numeric(cluster)) %>%
  rename(om_class = om,
         ppt_class = ppt,
         clay_class = clay,
         silt_class = silt,
         sand_class = sand,
         whc_class = whc,
         gdd_class = cum_gdd)
```

```

cluster_df <- clusters$cluster %>%
  enframe(name=NULL, value="cluster")

classified_data <- cluster_ready_data %>%
  cbind(cluster_df) %>%
  left_join(centers, by="cluster")

centers %>%
  kableExtra::kbl()

```

cluster	clay_class	gdd_class	om_class	ppt_class	sand_class	silt_class	whc_class
1	ave	low	ave	low	ave	ave	ave
2	ave	high	ave	high	low	high	ave
3	low	low	high	ave	high	low	low

We can now plot our clusters on a map. Cluster 1, which has a greater growing degree days, precipitation and silt, accounts for much of the lower third of our map. Cluster 2, which has lower growing degree days and precipitation, accounts for most of northwestern quarter of counties. Cluster 3, which is most remarkable for its higher sand content, accounts for parts of Nebraska, central Wisconsin and counties to the south and east of Lake Michigan.

```

classified_data_w_geo = clean_data_w_geo %>%
  dplyr::select(stco, geometry) %>%
  cbind(classified_data) %>%
  st_as_sf()
classified_data_w_geo = st_transform(classified_data_w_geo, 4326)

pal_ppt = colorFactor("RdYlGn", classified_data_w_geo$cluster)

m <- classified_data_w_geo %>%
  leaflet() %>%
  addTiles() %>%
  addPolygons(
    fillColor = ~pal_ppt(cluster),
    fillOpacity = 0.8,
    weight=1,
    color = "black",
    popup = paste0("om: ", classified_data_w_geo$om_class, "<br>",
                  "ppt: ", classified_data_w_geo$ppt_class, "<br>",
                  "clay: ", classified_data_w_geo$clay_class, "<br>",
                  "silt: ", classified_data_w_geo$silt_class, "<br>",
                  "sand: ", classified_data_w_geo$sand_class, "<br>",
                  "whc: ", classified_data_w_geo$whc_class, "<br>",
                  "gdd: ", classified_data_w_geo$gdd_class)

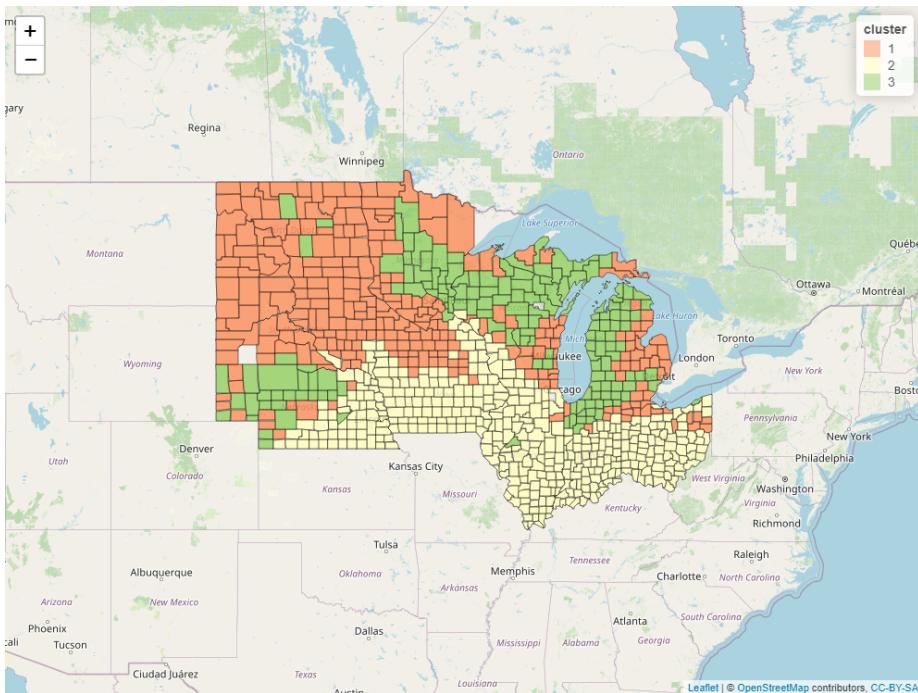
```

```

) %>%
addLegend(pal = pal_ppt,
           values = ~cluster
)

#above code won't knit -- created static image for markdown
saveWidget(m, "temp.html", selfcontained = FALSE)
webshot("temp.html", cliprect = "viewport")

```



### 13.3 k-Nearest-Neighbors

In *k*-*Nearest-Neighbors* (kNN) analyses, we again use the Euclidian distance between points. This time, however, we are not trying to cluster points, but to guess the value of a variable one observation, based on how similar it is to other observations.

### 13.3.1 Case Study: Guessing County Yields based on Environmental Similarity

For this example, we will continue to work with our county environmental dataset. This dataset includes yields for corn, soybean, and, where applicable, wheat and cotton. What if, however, our corn yield data were incomplete: that is, not every county had a recorded yield for corn. How might we go about guessing/estimating/predicting what that corn yield might be?

Well, if you are a farmer and you are interested in a new product – what might you do? You might ask around (at least the farmers who aren’t bidding against you for rented land) and see whether any of them have used the product and what response they observed.

In considering their results, however, you might incorporate additional information, namely: how similar are their farming practices to yours? Are they using the same amount of nitrogen fertilizer? Do they apply it using the same practices (preplant vs split) as you? Are they planting the same hybrid? What is their crop rotation? What is their tillage system?

In other words, as you ask around about others experiences with that product, you are also, at least intuitively, going to weigh their experiences based on how similar their practices are to yours. (Come to think of it, this would be great dataset to mock up for a future semester!)

In other words, you would predict your success based on your nearest neighbors, either by physical distance to your farm, or by the similarity (closeness) of your production practices. If that makes sense, you now understand the concept of nearest neighbors.

For our county data scenario, we will simulate a situation where 20% of our counties are, at random, missing a measure for corn yield.

```
yield_and_env = clean_data_w_geo

library(caret)

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##      lift
```

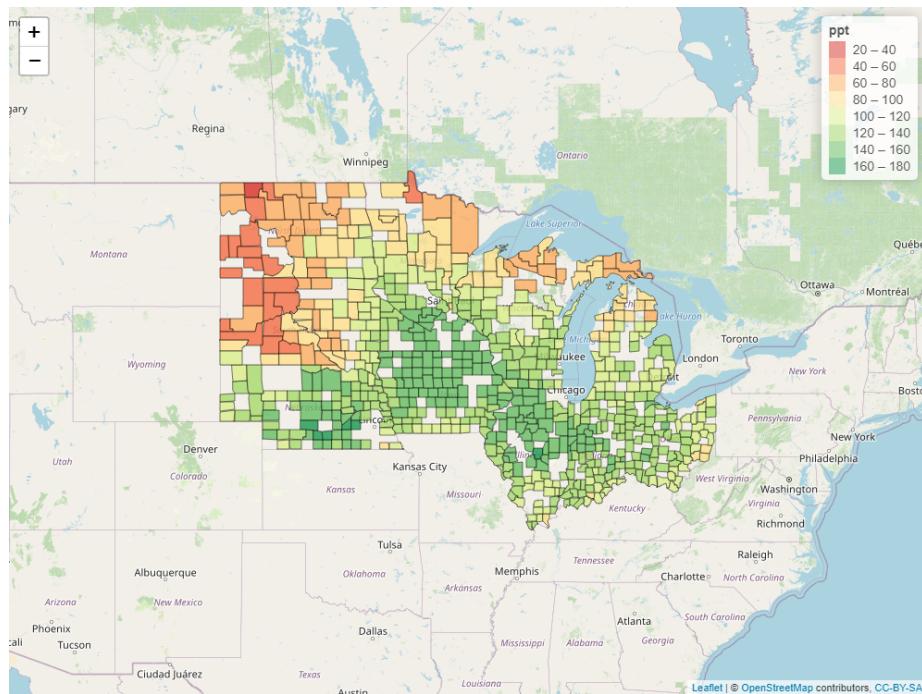
```
set.seed(112120)
trainingRows = createDataPartition(yield_and_env$corn, p=0.80, list = FALSE)
trainPartition = yield_and_env[trainingRows,]
trainData = trainPartition %>%
  st_drop_geometry() %>%
  dplyr::select(-stco)
testData = yield_and_env[-trainingRows,]
```

```
trainPartition = st_transform(trainPartition, 4326)

pal_corn = colorBin("RdYlGn", bins=5, trainPartition$corn)

m <- trainPartition %>%
  leaflet() %>%
  addTiles() %>%
  addPolygons(
    fillColor = ~pal_corn(corn),
    fillOpacity = 0.8,
    weight=1,
    color = "black",
    popup = paste("yield:", as.character(round(trainPartition$corn,1)))
  ) %>%
  addLegend(pal = pal_corn,
            values = ~ppt
  )

#above code won't knit -- created static image for markdown
saveWidget(m, "temp.html", selfcontained = FALSE)
webshot("temp.html", cliprect = "viewport")
```



### 13.3.2 Scaling

Remember, kNN analysis is like cluster analysis in that it is based on the *distance* between observations. kNN seeks to identify  $k$  neighbors that are most similar to the individual for whom we are trying to predict the missing value.

Variables with a greater range of values will be more heavily weighted in distance calculations, giving them excessive influence in how we measure distance (similarity).

The solution to this is to scale the data the same as we did for cluster analysis.

### 13.3.3 k-Nearest-Neighbor Animation

In kNN analysis, for each individual for which we are trying to make a prediction, the distance to each other individual is calculated. As a reminder, the Euclidean distance between points  $p$  and  $q$  is calculated as:

$$dist(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 \dots (p_n - q_n)^2}$$

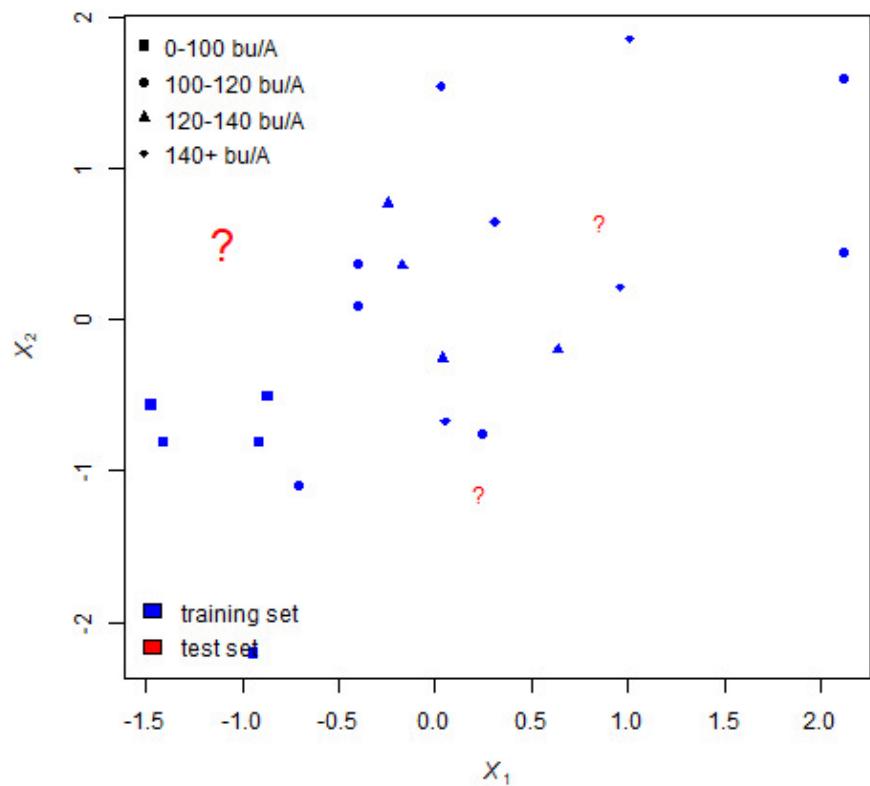
The  $k$  nearest (most similar) observations are then identified. What happens next depends on whether the measure we are trying to predict is qualitative

or quantitative. Are we trying to predict a drainage category? Whether the climate is “hot” vs “cold”? In these cases, the algorithm will identify the value that occurs most frequently among those neighbors. This statistic is called the *mode*. The mode will then become the predicted value for the original individual. Let’s demonstrate this below.

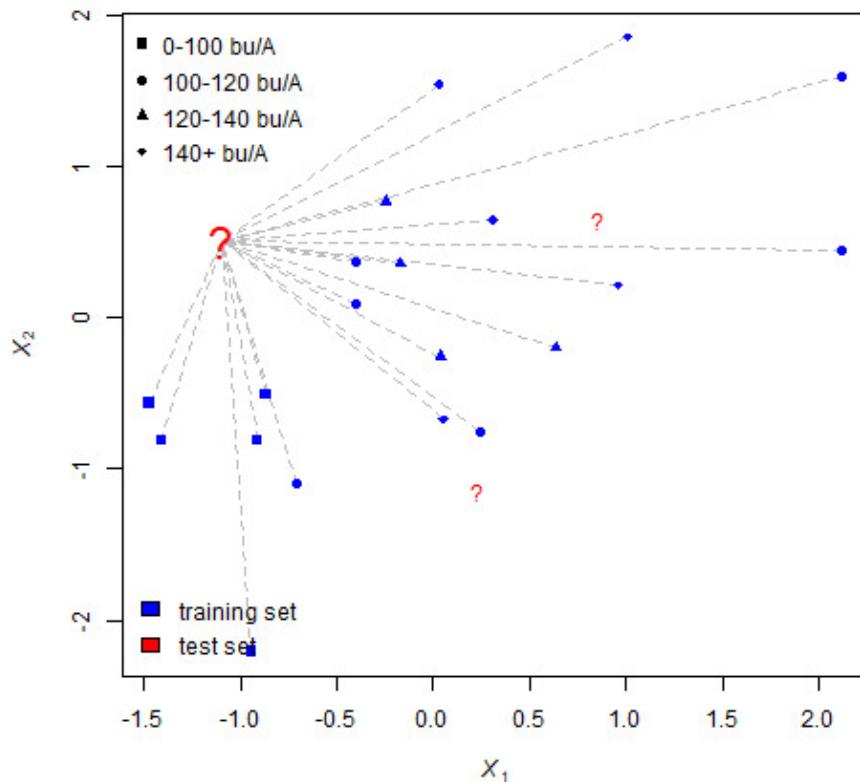
In this example, corn yield is predicted based on cumulative growing degree days and percent silt. Because it is easier to illustrate qualitative predictions, corn yield has been categorized within four ranges: 0-100 (circle), 100-120 (triangle), 120-140 (cross), and 140+ (“x”).

The known yields, referred to as the training set, are colored blue. The individuals for which we are trying to predict yield are represented by red question marks. Although the axes names are ambiguous, each known point is plotted according to its cumulative growing degree days (cum\_gdd) and silt content.

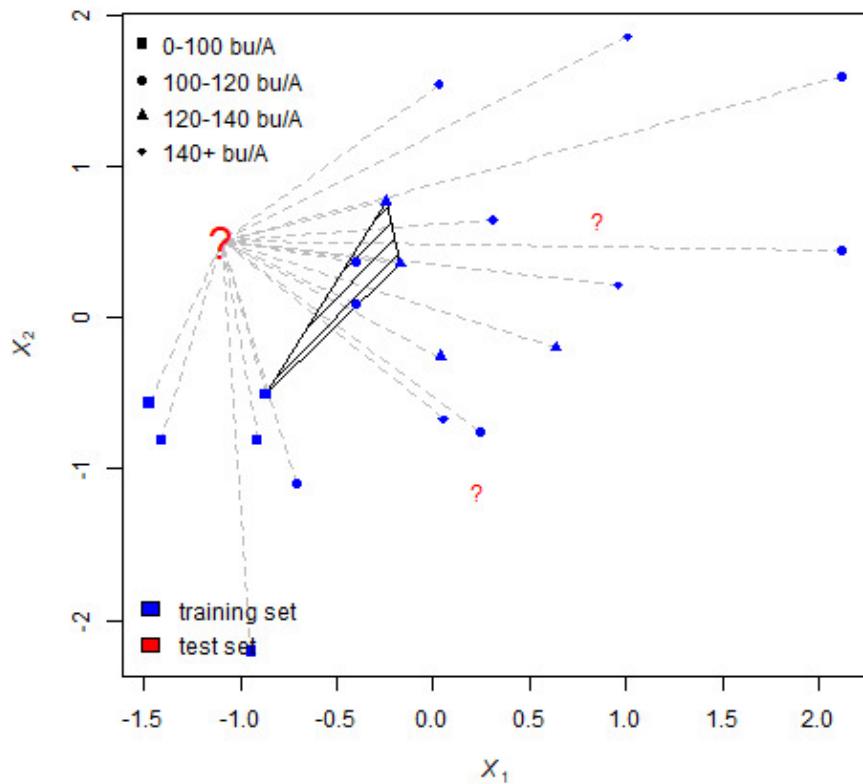
This example uses  $k=5$ . That is, the missing values of individuals will be calculated based on the most frequent yield classification of their 5 nearest neighbors. There are three points, indicated by red question marks for which we are going to try to pick the value. We start with the leftmost question mark.



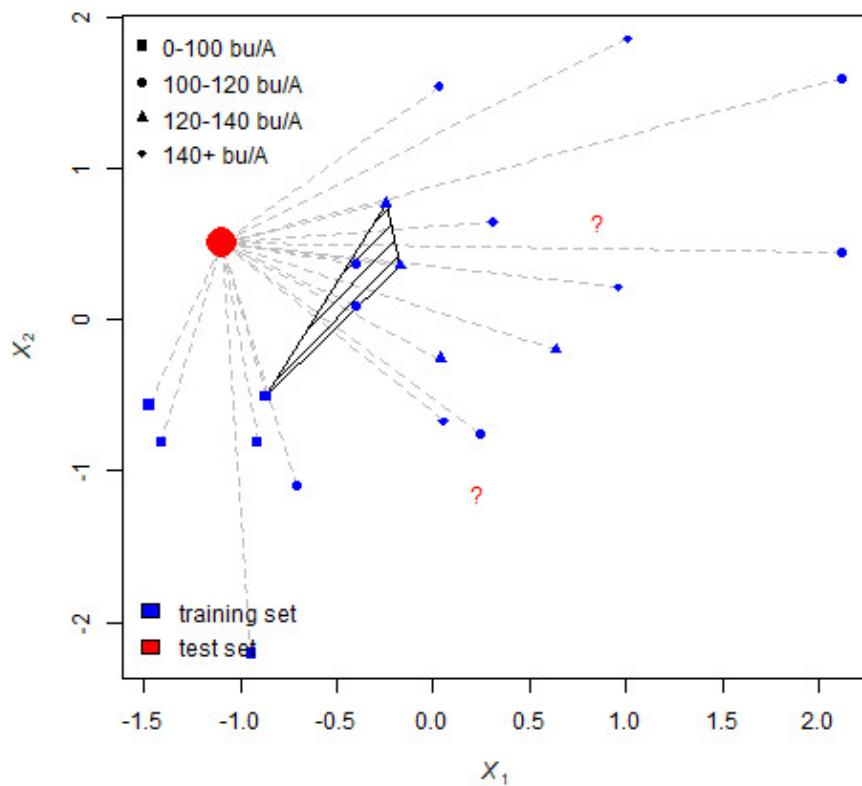
For that point, we measure the distance to every other point:



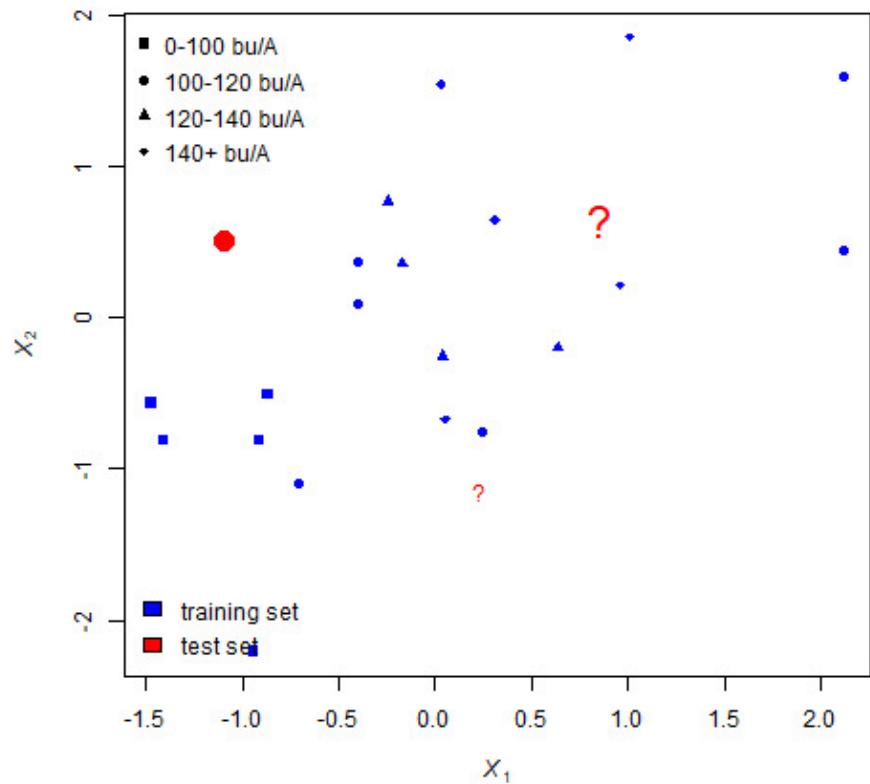
And identify the nearest 5 points (neighbors). Of the five nearest neighbors, there is one square (yield up to 100 bushels), two circles (indicating yields between 100 and 120 bushels) and two triangles (yields between 120 and 140 bushels).



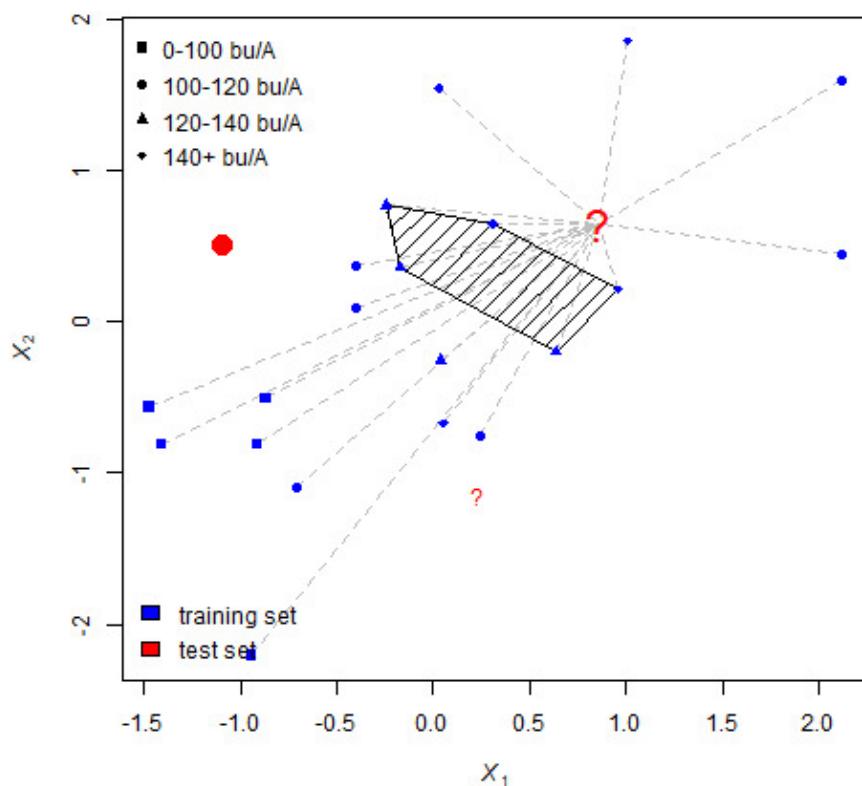
There are two each circles and triangles, so R randomly selects from those two values. The missing value is classified as a circle.



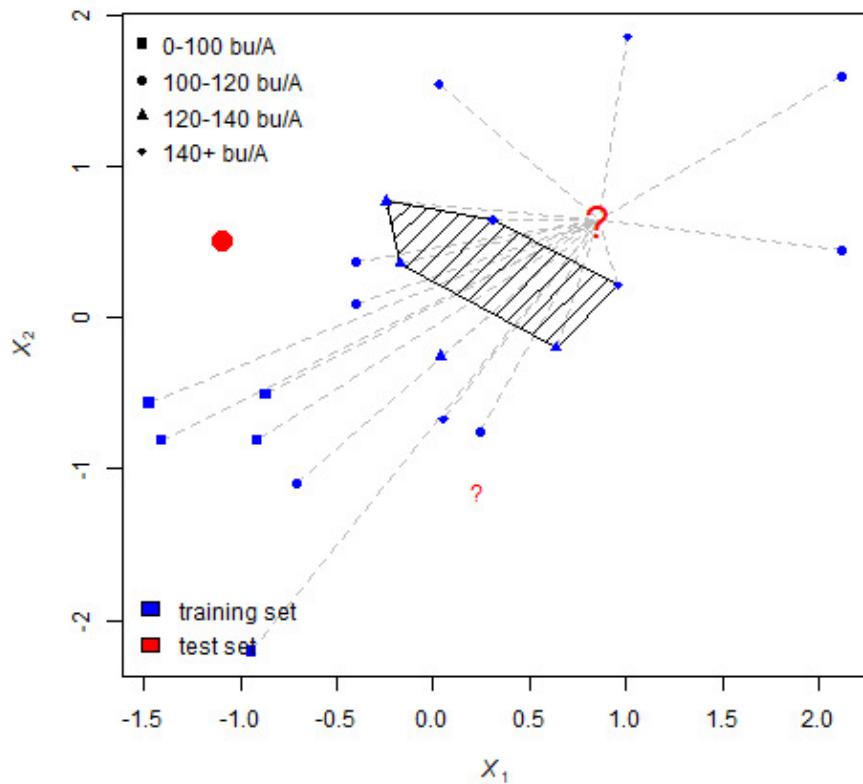
Let's repeat this process for the next missing value:



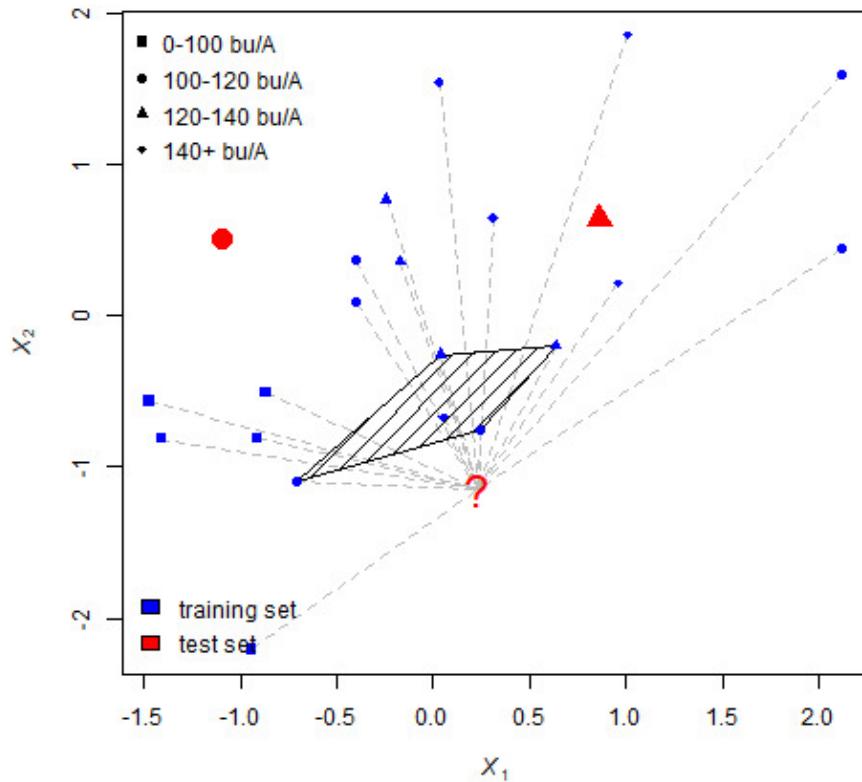
For this point, the five nearest neighbors are again identified.



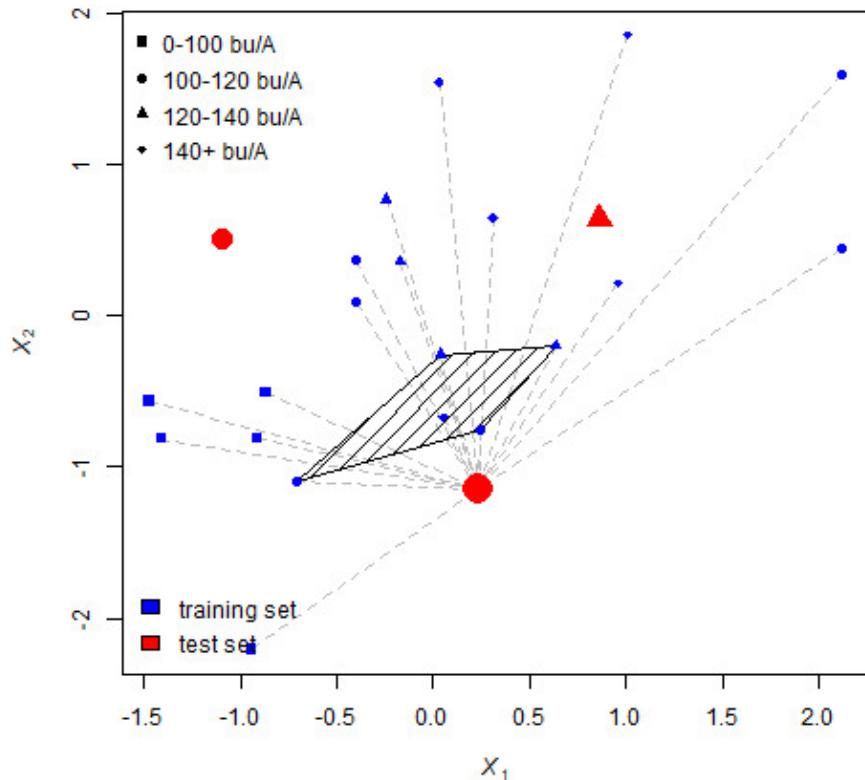
In this case there are three triangles and two circles, so the missing value is estimated to be a triangle.



For the last point, the nearest 5 neighbors include two circles, two triangles, and one diamond.



R randomly chooses between the triangle and circle – again, the circle wins.



### 13.3.4 Choosing $k$

In the example above, a value of 5 chosen arbitrarily to keep the illustration from becoming too muddled. How should we choose  $k$  in actual analyses?

Your first instinct might be to select a large  $k$ . Why? Because we have learned throughout this course that larger sample sizes reduce the influences of outliers on our predictions. By choosing a larger  $k$ , we reduce the likelihood an extreme value (a bad neighbor?) will unduly skew our prediction. We reduce the potential for bias.

The flip side of this is: the more neighbors we include, the less “near” they will be to the individual for which we are trying to make a prediction. We reduce the influence of outlier, but at the same time predict a value that is closer to the population mean, rather than the true missing value. In other words, a larger  $k$  also reduces our precision.

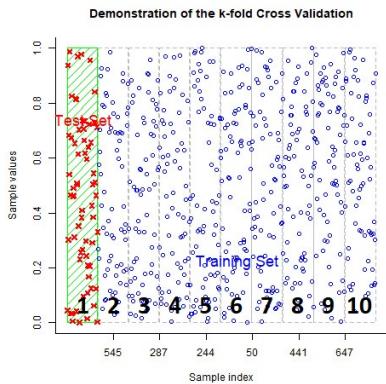
A general rule of thumb is to choose a  $k$  equal to the square root of the total number of individuals in the population. We had 50 individuals in the population above, so we could have set our  $k$  equal to 7.

### 13.3.5 Model Cross-Validation

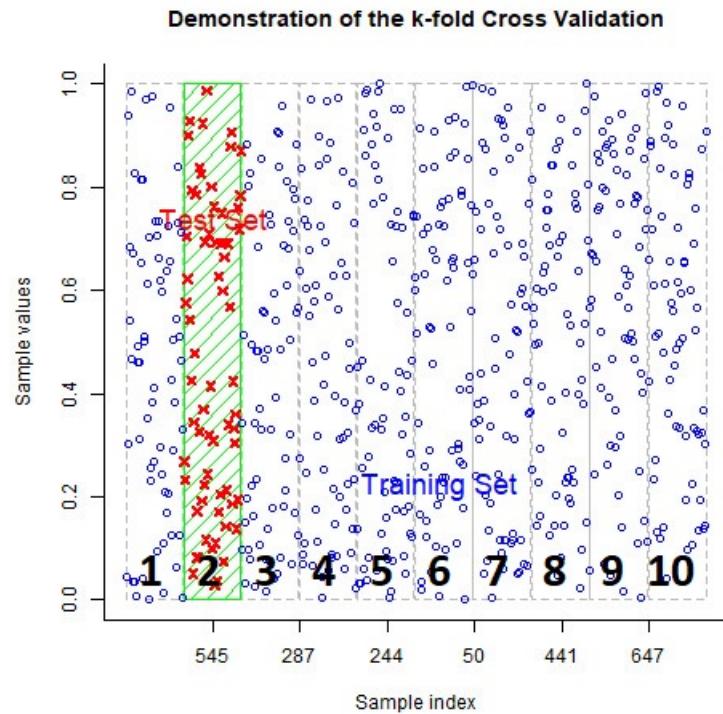
When we fit a kNN model, we should also cross-validate it. In cross-validation, the data are portioned into two groups: a training group and a testing group. The model is fit using the training group. It is then used to make predictions for the testing group.

The figures below illustrate a common cross-validation method, k-fold cross validation. In this method, the data are divided into  $k$  number of groups. In this example,  $k=10$ , meaning the data are divided into 10 groups. Nine of the groups (in blue) are combined to train the model. The tenth group (in red) is the testing group, and used to validate the fit. The validation process goes through 10 rounds.

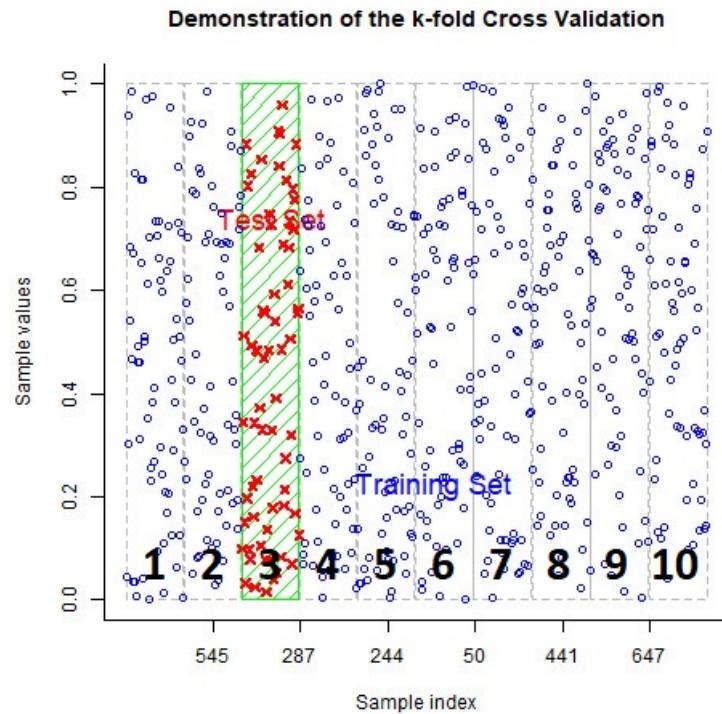
In the first round, Groups 2 through 9 are used to train the model. The fit of the model is then tested by how closely it can predict the observed values in Group 1.



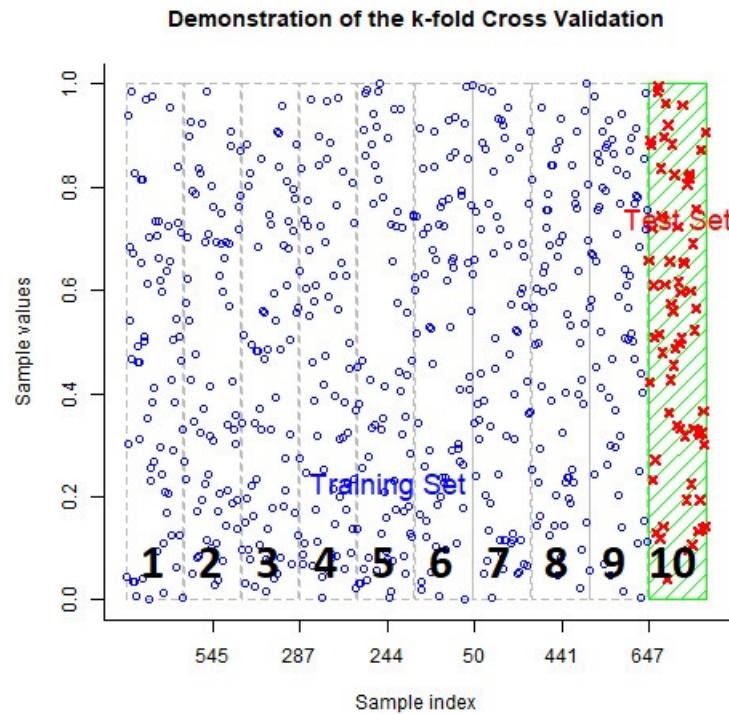
In the first round, Groups 1, 3, 4, 5, 6, 7, 8, 9, and 10 are used to train the model. The fit of the model is then tested by how closely it can predict the observed values in Group 2.



In the third round, Groups 1, 2, 4, 5, 6, 7, 8, 9, and 10 are used to train the model. The fit of the model is then tested by how closely it can predict the observed values in Group 3.



This continues until each group has been used to train and test the model.



A linear regression of the predicted test values on the actual test values is then conducted and the regression coefficient,  $R^2$ , is calculated. The Root Mean Square Error is also calculated during the regression. You will recall the Root MSE is a measure of the standard deviation of the difference between the predicted and actual values.

### 13.3.6 Yield Prediction with Nearest Neighbor Analysis

As mentioned above, nearest neighbor analysis also works with qualitative data. In this case, the values of the nearest neighbors are averaged for the variable of interest. In R, there are various functions for running a nearest-neighbor analysis. The output below was generated using the *caret* package, a very powerful tool which fits and cross-validates models at once.

```
library(caret)
ctrl = trainControl(method="repeatedcv", number=10, repeats = 1)

#knn
knnFit = train(corn ~ .,
                data = trainData,
```

```

method = "knn",
preProc = c("center", "scale"),
trControl = ctrl)

knnFit

## k-Nearest Neighbors
##
## 665 samples
##    7 predictor
##
## Pre-processing: centered (7), scaled (7)
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 599, 599, 598, 599, 597, 598, ...
## Resampling results across tuning parameters:
##
##     k   RMSE      Rsquared    MAE
##     5   11.04766  0.8129430  7.996958
##     7   10.99521  0.8177150  8.001320
##     9   11.07555  0.8157225  8.147071
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 7.

```

In the above output, the following are important:

- Resampling: this line tells us how the model was cross-validated (10-fold cross validation)
- Identification of the final value used for the model: Remember when we discussed how to select the appropriate  $k$  (number of neighbors)? R compares multiple values of  $k$ . In this case, it identifies  $k=7$  as optimum for model performance.
- Model statistics table. Statistics for three levels of  $k$  are given: 5, 7, and 9. RMSE is the root means square error, Rsquared is  $R^2$ , the regression coefficient, and MAE is Mean Absolute Error (an alternative calculation of error that does not involve squaring the error).

Looking at the table results for  $k=7$ , we see the model had an  $R^2$  of almost 0.81. This is pretty impressive, given that were predicting mean county corn yield using relatively few measures of soil and environment. The RMSE is about 11.1. Using this with our Z-distribution, we can predict there is a 95% chance the actual corn yield will be within  $1.96 \times 11.1 = 21.8$  bushels of the predicted yield.

In this example, as you might have suspected, we did know the corn yield in the counties where it was “missing”; we just selected those counties at random and used them to show how k-nearest-neighbors worked.

So let’s use the our nearest-neighbor model to predict the yields in those “missing” counties and compare them to the actual observed yields. We can use a simple linear regression model for this, where:

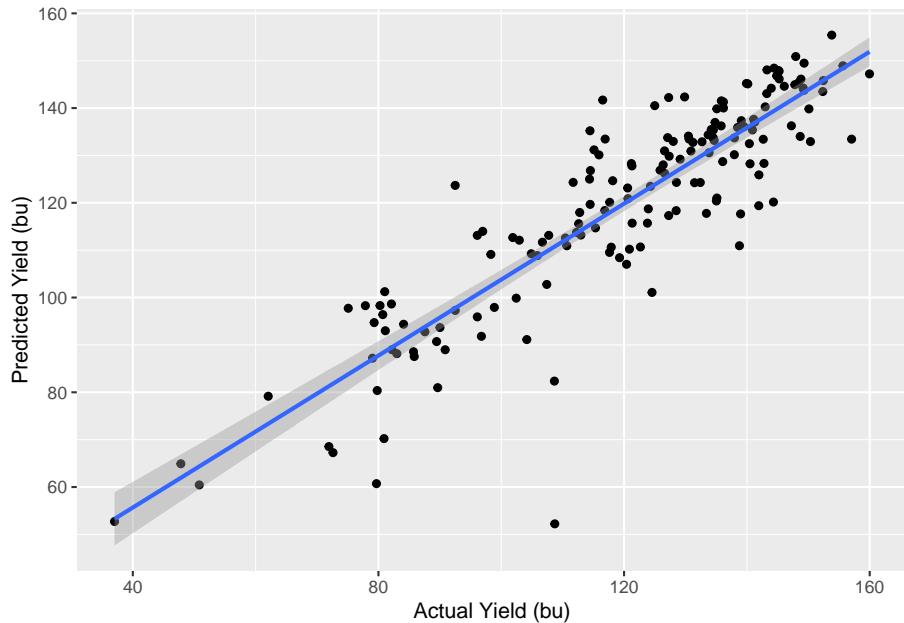
$$\text{Predicted Yield} = \alpha + \beta \cdot \text{Actual Yield}$$

Below, the predicted yields (y-axis) are regressed against the actual yields (x-axis).

```
testData$corn_pred = predict(knnFit, testData)

testData %>%
  ggplot(aes(x=corn, y=corn_pred)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(x="Actual Yield (bu)", y="Predicted Yield (bu)")

## `geom_smooth()` using formula 'y ~ x'
```



We can see the predicted and actual yields are strongly correlated, as evidenced by their linear distribution and proximity to the regression line. The regression model statistics are shown below.

```
library(broom)

## Warning: package 'broom' was built under R version 4.1.3

model = lm(corn_pred~corn, data=testData)
glance(model) %>%
  dplyr::select(r.squared, sigma, p.value) %>%
  kableExtra::kbl()
```

r.squared	sigma	p.value
0.785811	10.35285	0

Above are select statistics for our model. We see the r.squared, about 0.83 is even better greater than in our cross-validation. Sigma, the standard deviation, is smaller than in the validation, barely 9 bushels. Finally, we see the significance of our model is very, very small, meaning the relationship between predicted and actual yield was very strong.

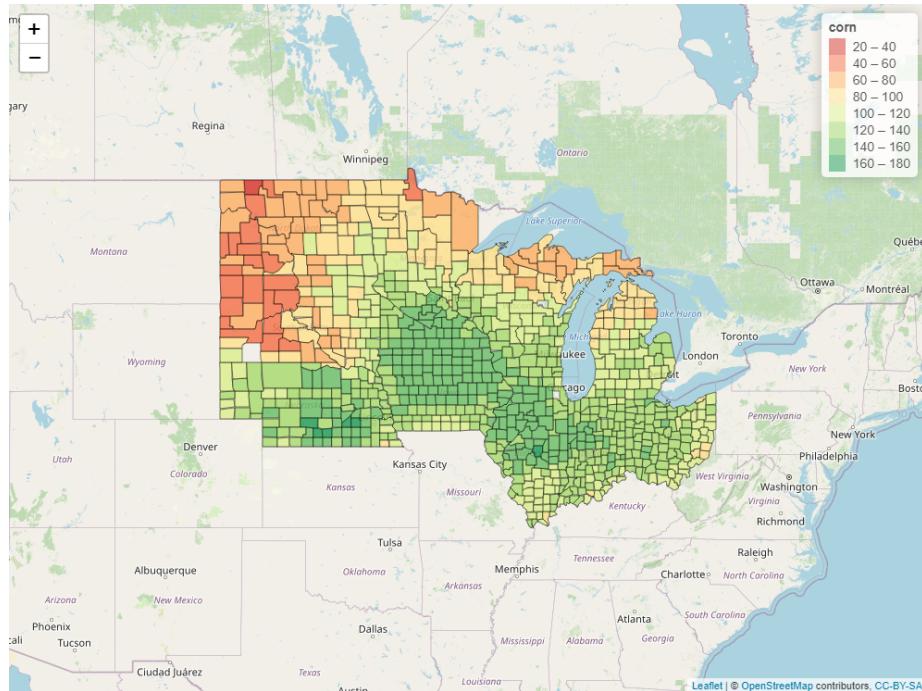
```
complete_dataset = testData %>%
  dplyr::select(-corn) %>%
  rename(corn=corn_pred) %>%
  rbind(trainPartition)
```

Finally, we can fill in the missing county yields in the map with which we started this section. We see the predicted yields for the missing county seem to be in line with their surrounding counties; there are no islands of extremely high or extremely low yields. You can see the yield of each county by clicking on it.

```
pal_corn = colorBin("RdYlGn", bins=5, complete_dataset$corn)

m <- complete_dataset %>%
  leaflet() %>%
  addTiles() %>%
  addPolygons(
    fillColor = ~pal_corn(corn),
    fillOpacity = 0.8,
    weight=1,
    color = "black",
    popup = paste("yield:", as.character(round(complete_dataset$corn,1)))
  ) %>%
  addLegend(pal = pal_corn,
            values = ~corn)
```

```
#above code won't knit -- created static image for markdown
saveWidget(m, "temp.html", selfcontained = FALSE)
webshot("temp.html", cliprect = "viewport")
```



## 13.4 Classification Trees

The last data science tool we will learn in this unit the classification tree. The classification tree uses a “divide and conquer” or sorting approach to predict the value of a response variable, based on the levels of its predictor variables.

The best way to understand how a classification tree works is to play the game Plinko. In Plinko, a player drops disc at the top of a table studded with pins.



The disc falls down the table, bouncing to one side or the other of each successive pin.







The bin determines what fabulous prize the player may have won.



Each level of the decision tree can be thought of as having pins. Each pin is a

division point along variable of interest. If an observation has a value for that variable which is greater than the division point, it is classified one way and moves to the next pin. If less, it moves the other way where it encounters a different pin.

### 13.4.1 Features

Classification trees are different from cluster and nearest neighbor analyses in that they identify the relative importance of *features*. Features include every predictor variable in the dataset. The higher a feature occurs in a classification tree, the more important they are in predicting the final value.

Classification trees therefore shed a unique and valuable insight into the importance of variables, distinct from all other analyses we have learned. They provide insight into *feature importance*, not just their significance as a predictor.

### 13.4.2 Quantitative (Categorical) Data

A classification tree predicts the category of the response variable. We will start with yield categories, just as we did with the nearest neighbors section, because the results are intuitively easier to visualize and understand.

The classification process begins with the selection of a root. This is the feature that, when split, is the best single predictor of the response variable. The algorithm not only selects the root; it also determines where within the range (the threshold) of that feature to divide the populations.

What is the criterion for this threshold? It is the value that divides the population into two groups in which the individuals within the group are as similar as possible.

These results are displayed as a tree; thus, the “classification tree”. The results of our first split are shown below:

```
library(party)

## Loading required package: grid

## Loading required package: mvtnorm

## Loading required package: modeltools

## Loading required package: stats4

## Loading required package: strucchange
```

```
## Loading required package: zoo

## 
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
## 
##     as.Date, as.Date.numeric

## Loading required package: sandwich

## 
## Attaching package: 'strucchange'

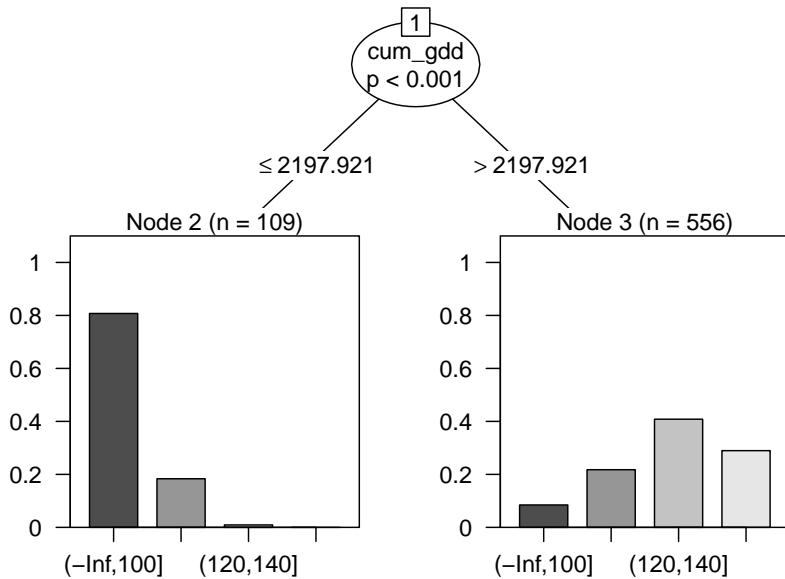
## The following object is masked from 'package:stringr':
## 
##     boundary

x = trainData %>%
  dplyr::select(cum_gdd, silt, corn)

classes = cut(x$corn, breaks=c(-Inf, 100, 120, 140, Inf))
x = x %>%
  dplyr::select(-corn)

fit = ctree(classes ~., data=x,
            controls = ctree_control(maxdepth = 1))

plot(fit)
```



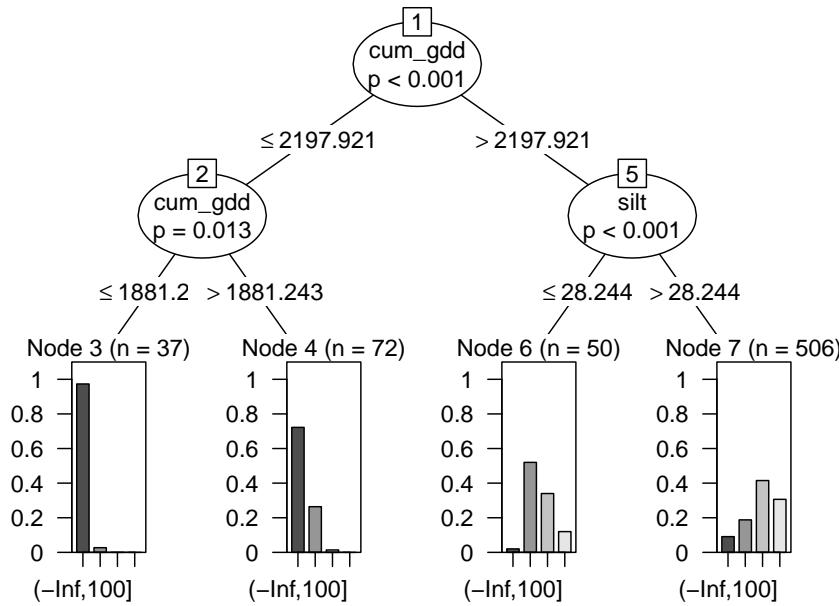
Cumulative growing degree days (`cum_gdd`) was selected as the root, and the threshold or split value was approximately 2198 GDD. Counties where the cumulative GDD was less than this threshold tended to have yields in the lower two classes; above this threshold, counties tended to have yields in the greater two classes.

This particular algorithm also calculates a p-value that the distribution of the counts among the yield classes is different for the two groups. We see the difference between the two groups (greater vs less than 2198 GDD) is highly significant, with a p-value  $< 0.001$ .

Next, let's add another level to our tree. The two subgroups will each be split into two new subgroups.

```
fit = ctree(classes ~., data=x,
            controls = ctree_control(maxdepth = 2))

plot(fit)
```



Our tree now has three parts. The root was discussed before. At the bottom of the tree, we have four nodes: Node 3, Node 4, Node 6, and Node 7. These represent the final groupings of our data.

In the middle, we have two branches, labelled as branches 2 and 5 of our tree. Each branch is indicated by an oval. The branch number is in a box at the top of the oval.

Counties that had  $\text{cum\_gdd} < 2198$  were divided at branch 2 by  $\text{cum\_gdd}$ , this time into counties with less than 1881  $\text{cum\_gdd}$  and counties with more than 1881  $\text{cum\_gdd}$ . Almost all counties with less than 1881  $\text{cum\_gdd}$  had yields in the lowest class. Above 1881  $\text{cum\_gdd}$ , most yields were still in the lowest class, but some were in the second lowest class.

Counties with greater than 2198  $\text{cum\_gdd}$  were divided at branch 5 by soil percent silt. Counties with soils that with less than 28.2 percent slit had yields that were mostly in the middle two yield classes. Above 28.2 percent silt, most yields were in the upper two classes.

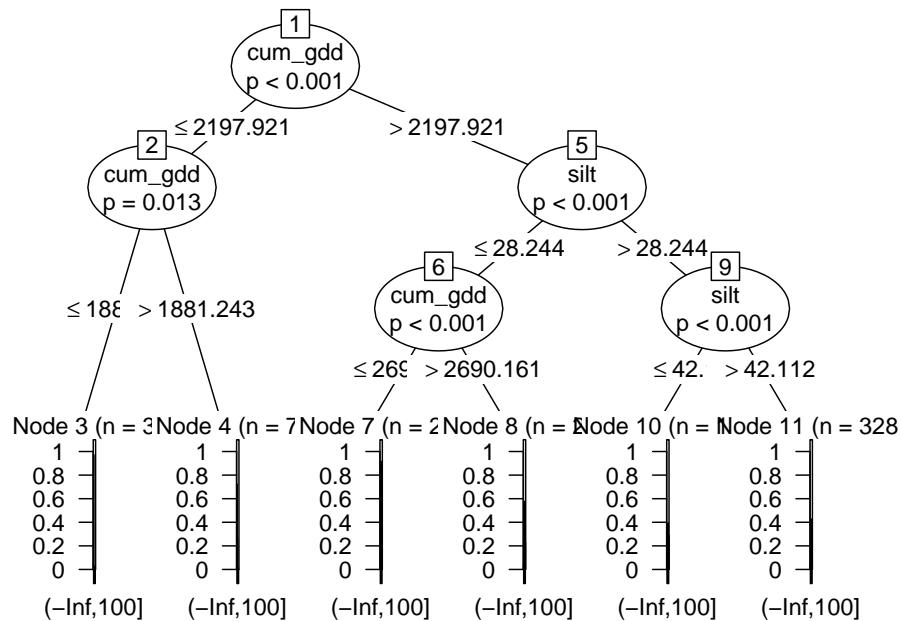
The beauty of the regression tree is we are left with a distinct set of rules for predicting the outcome of our response variable. This lends itself nicely to explaining our production. Why do we predict a county yield will be in one of the two highest classes? Because the county has cumulative GDDs greater than 2198 and a soil that has a percent silt greater than 28.2

The algorithm will continue to add depths to the tree until the nodes are as homogeneous as possible. The tree, however, will then become too complex to

visualize. Below we have three levels to our tree – we can see how difficult it is to read.

```
fit = ctree(classes ~., data=x,
            controls = ctree_control(maxdepth = 3))

plot(fit)
```

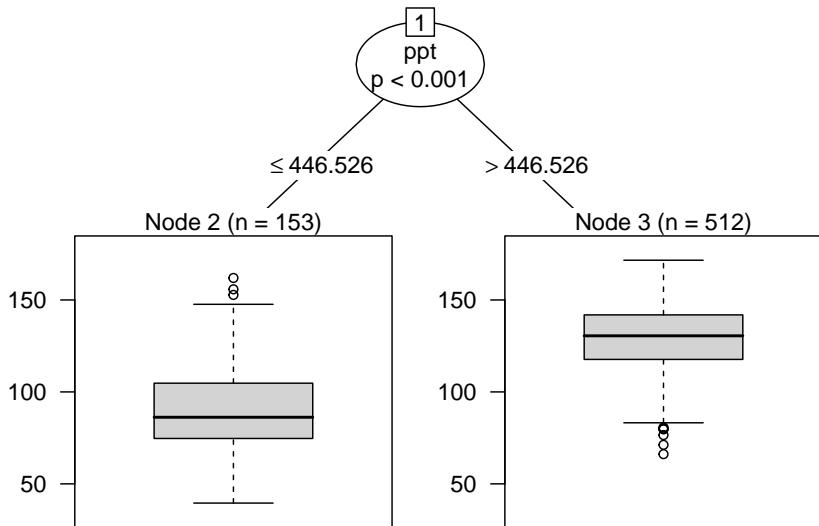


### 13.4.3 Quantitative (Continuous) Data

The same process above can be used with quantitative data – in this example, the actual yields. When we use yield, the root of our tree is not cumulative growing degree days, but total precipitation. Instead of a bar plot showing us the number of individuals in each class, we get a boxplot showing the distribution of the observations in each node. When we work with quantitative data, R identifies the feature and split that minimizes the variance in each node.

```
fit = ctree(corn ~., data=trainData,
            controls = ctree_control(maxdepth = 1))

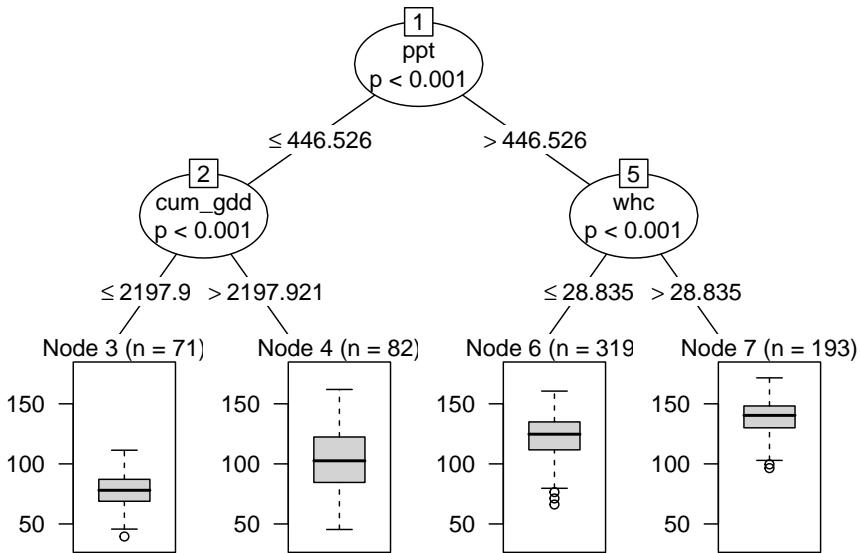
plot(fit)
```



When we add a second level to our tree, we see that observations where precipitation was less than or equal to about 446.5 mm/season, the data was next split by cum\_gdd. Counties with a cum\_gdd less than about 2198 had lower yields than counties with a cum\_gdd > 2198.

Where ppt was greater than 446 mm per season, individuals were next split by water holding capacity (whc). Counties where whc was greater than about 28.8 cm/cm soil had greater yield than counties with less water holding capacity.

```
fit <- ctree(corn ~ ., data=trainData,
               controls = ctree_control(maxdepth = 2))
plot(fit)
```



Like before, we can continue adding branches, although our virtual interpretation of the data will become more challenging.

#### 13.4.4 Overfitting

In nearest-neighbors analysis, we learned the size of  $k$  (the number of neighbors) was a tradeoff: too few neighbors, and the risk of an outlier skewing the prediction increased. Too many neighbors, and the model prediction approached the mean of the entire population. The secret was to solve for the optimum value of  $k$ .

There is a similar tradeoff with classification trees. We can continue adding layers to our decision tree until only one observation is left in each node. At that point, we will perfectly predict each individual in the population. Sounds great, huh?

Yeah, but what about when you use the classification model (with its exact set of rules for the training population) on a new population? It is likely the new population will follow the more general rules, defined by the population splits closer to the root of the tree. It is less likely the population will follow the specific rules that define the nodes and the branches immediately above.

Just as a bush around your house can become messy and overgrown, so can a classification tree become so branched it loses its value. The solution in each case is to prune, so that the bush regains its shape, and the classification tree approaches a useful combination of general application and accurate predictions.

### 13.4.5 Cross Validation

Just as with nearest-neighbors, the answer to overfitting classification trees is cross-validation. Using the same cross-validation technique (10-fold cross-validation) we used above, let's fit our classification tree.

```
library(caret)
ctrl = trainControl(method="repeatedcv", number=10, repeats = 1)

#knn
partyFit = train(corn ~.,
                  data = trainData,
                  method = "ctree",
                  trControl = ctrl)

partyFit

## Conditional Inference Tree
##
## 665 samples
##    7 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 598, 599, 599, 599, 598, 598, ...
## Resampling results across tuning parameters:
##
##   mincriterion  RMSE      Rsquared     MAE
##   0.01          13.05396  0.7412480  9.689862
##   0.50          13.20743  0.7342159  9.974611
##   0.99          15.13028  0.6567629  11.933852
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mincriterion = 0.01.
```

The output looks very similar to that for nearest neighbor, except instead of  $k$  we see a statistic called `mincriterion`. The `mincriterion` specifies the maximum p-value for any branch to be included in the final model.

As we discussed above, the ability of the classification tree to fit new data decreases as the number of branches and nodes increases. At the same time, we want to include enough branches in the model so that we are accurate in our classifications.

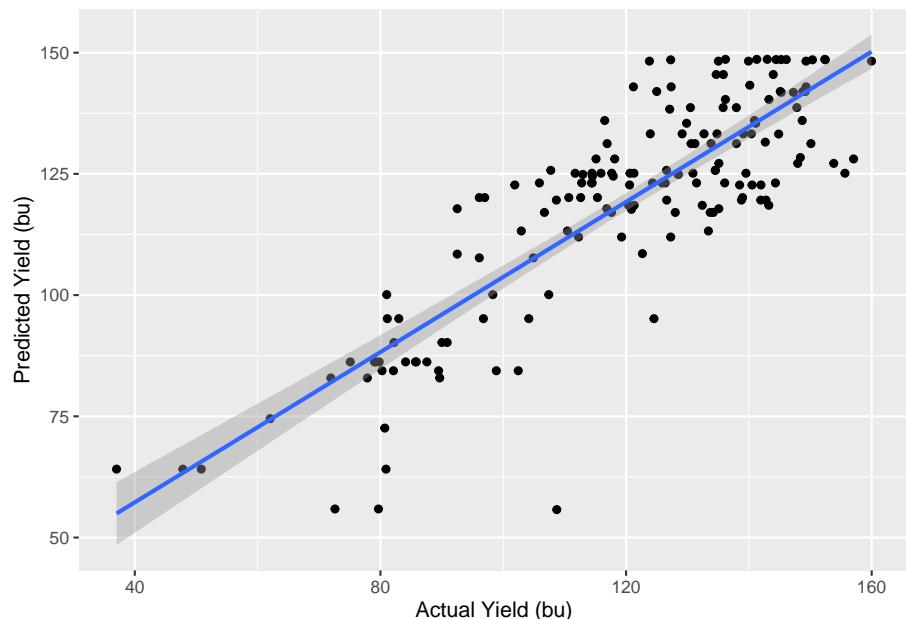
In the output above, the `mincriterion` was 0.01, meaning the p-value must be no greater than 0.01. This gave us an `Rsquared` of about 0.745 and and `Root MSE` of about 13.0.

As with the  $k$ -Nearest-Neighbors analysis, we can compare the predicted yields for the counties dropped from our model to their actual values.

```
testData$corn_pred = predict(partyFit, testData)

testData %>%
  ggplot(aes(x=corn, y=corn_pred)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(x="Actual Yield (bu)", y="Predicted Yield (bu)")

## `geom_smooth()` using formula 'y ~ x'
```



```
library(broom)
model = lm(corn_pred~corn, data=testData)
glance(model) %>%
  dplyr::select(r.squared, sigma, p.value)
```

```
## # A tibble: 1 x 3
##   r.squared sigma  p.value
##       <dbl>  <dbl>    <dbl>
## 1     0.720  12.0  1.39e-46
```

We see the model strongly predicts yield, but not as strongly as  $k$ -Nearest Neighbors.

### 13.4.6 Random Forest

There is one last problem with our classification tree. What if the feature selected as the root of our model is the best for the dataset to which we will apply the trained model? In that case, our model may not be accurate. In addition, once a feature is chosen as the root of a classification model, the features and splits in the remainder of the model also become constrained.

This not only limits model performance, but also gives us an incomplete sense of how each feature truly effected the response variable. What if we started with a different root feature? What kind of model would we get?

This question can be answered with an *ensemble* of classification trees. This ensemble is known as a *random forest* (i.e. a collection of many, many classification trees). The random forest generates hundreds of models, each starting with a randomly selected feature as the root. These models are then combined as an ensemble to make predictions for the new population.

```
rfFit = train(corn ~ .,
              data = trainData,
              method = "rf",
              trControl = ctrl)

rfFit

## Random Forest
##
## 665 samples
##    7 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 598, 600, 600, 601, 599, 597, ...
## Resampling results across tuning parameters:
##
##     mtry   RMSE      Rsquared    MAE
##     2      10.559385  0.8360096  7.868132
##     4      9.983132   0.8482014  7.327396
##     7      9.788704   0.8519592  7.159581
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 7.
```

If we compare our model results to the individual classification tree above, and to the performance of our nearest-neighbor model, we see the random forest provides a better fit of the data.

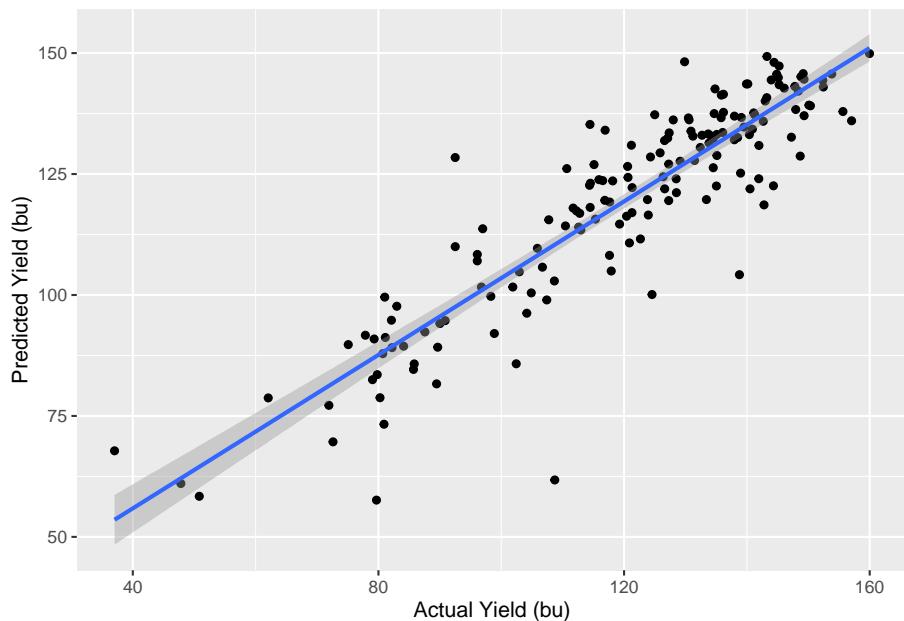
```

testData$corn_pred = predict(rfFit, testData)

testData %>%
  ggplot(aes(x=corn, y=corn_pred)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(x="Actual Yield (bu)", y="Predicted Yield (bu)")

## `geom_smooth()` using formula 'y ~ x'

```



```

library(broom)
model = lm(corn_pred~corn, data=testData)
glance(model) %>%
  dplyr::select(r.squared, sigma, p.value)

```

```

## # A tibble: 1 x 3
##   r.squared sigma  p.value
##       <dbl>  <dbl>    <dbl>
## 1     0.810  9.51  3.09e-60

```

### 13.4.7 Feature Importance

When we use random forest analysis, there is no one tree to review – the forest, after all, is a collection of many hundreds or thousands of trees. There is also no longer one set of rules to explain the predicted values, since all trees were used to predict corn yield.

Random forest analysis, however, allows us to rank the features by importance. Feature importance is calculated from the number of times a feature occurs in the top branches of tree and the most frequent level at which a feature first occurs in the model.

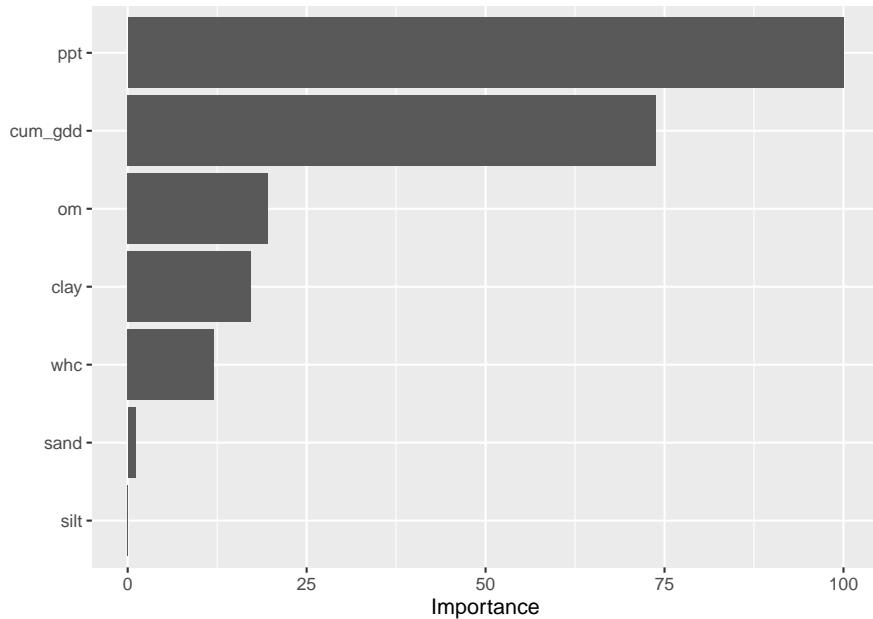
We see in the barplot below the feature importance in our random forest model. Precipitaion was by far the single most important feature in predicting yield. The next most important feature, as we might expect, was cumulative growing degree days.

```
library(vip)

##
## Attaching package: 'vip'

## The following object is masked from 'package:utils':
##     vi

vip(rfFit)
```



After that, we might wonder whether soil biology or texture might be the next most important feature. Our feature importance ranking suggests that organic matter is slightly more important than clay, followed by water holding capacity. We might infer that soil fertility which increases with organic matter and clay, was more important than drainage, which would be predicted by water holding capacity and our last variable, sand.

## 13.5 Summary

With that, you have been introduced to three (four, if we include random forest) powerful machine learning tools. I don't expect that you will personally use these tools on a regular basis; thus, this lesson is not hands-on like other lessons. It is most important that you are aware there is a whole world of statistics beyond what we have learned this semester, beyond designed experiments and the testing of treatments one-plot-at-a-time.

This is not to disparage or minimize the statistics we learned earlier in this course. Controlled, designed-experiments will always be key in testing new practices or products. They will be how products are screened in the greenhouse, in the test plot, in the side-by-side trial on-farm. They are critical.

But for big, messy problems, like predicting yield (or product performance) county-by-county, machine learning tools are very cool. They are behind the hybrid recommended for your farm, the nitrogen rate recommended for your

sidedress. They are way cool and offer wicked insights and I believe it is worth your while – as agricultural scientists – to be aware of them.

```
# knitr::opts_chunk$set(cache = F)
# write("TMPDIR = './temp'", file=file.path(Sys.getenv('R_USER'),
# # '.Renviron'))
```



# Chapter 14

## Putting it all Together

If I have done my job properly, your head is swimming with the different possibilities for collecting and analyzing data. Now it is time for us to back up and revisit the tests we have learned and understand how to choose a design or test for a given situation. Furthermore, it is important we round out this semester by understanding how to report results to others. This includes what to report and how to present it.

In contrast with more general statistics courses, I have tried to build this course around a series of trials which you are likely to conduct or from which you will use data. These range from a simple yield map through more complex factorial trials through nonlinear data and application maps. In this unit, we will review those scenarios and the tools we used to hack them.

### 14.1 Scenario 1: Yield Map (Population Summary and Z-Distribution)

*You are presented with a yield map and wish to summarize its yields so you can compare it with other fields. Since you have measured the entire field with your yield monitor, you are summarizing a population and therefore will calculate the population mean and standard deviation.*

We started out the semester with a yield map, like this one:

```
library(tidyverse)
library(sf)
library(leaflet)
library(grDevices)
library(htmlwidgets)
```

```

library(webshot2)

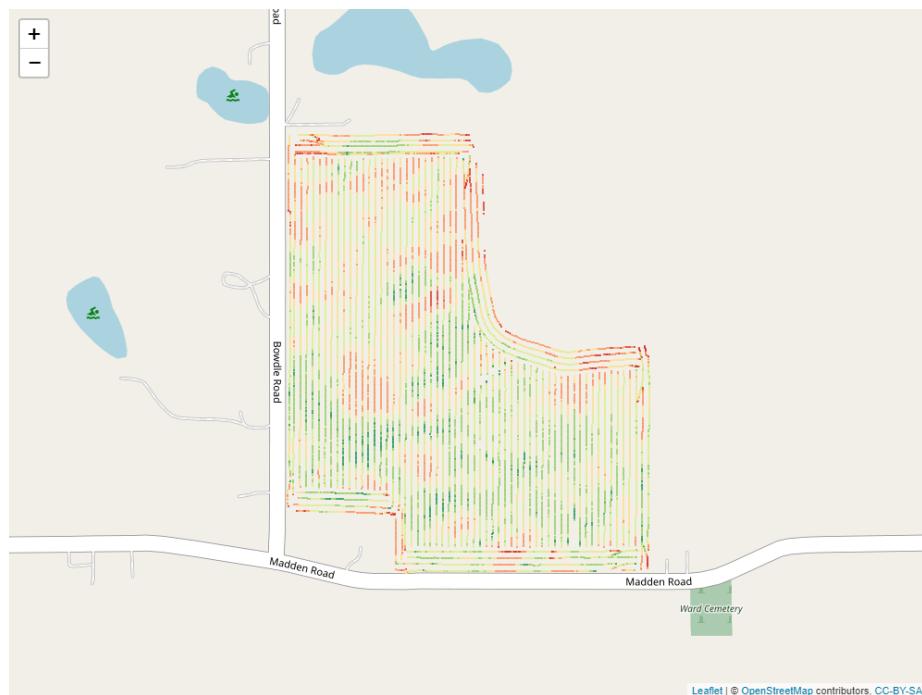
logan = st_read("data-unit-14/Young North Corn 20.shp", quiet=TRUE) %>%
  filter(Yld_Vol_Dr >=50 & Yld_Vol_Dr <=350)

pal = colorBin("RdYlGn", logan$Yld_Vol_Dr)

m <- logan %>%
  leaflet() %>%
  addTiles() %>%
  addCircleMarkers(
    radius = 1,
    fillColor = ~pal(Yld_Vol_Dr),
    weight = 0,
    fillOpacity = 1,
    popup = as.character(logan$Yld_Vol_Dr)
  )

#above code won't knit -- created static image for markdown
saveWidget(m, "temp.html", selfcontained = FALSE)
webshot("temp.html", cliprect = "viewport")

```



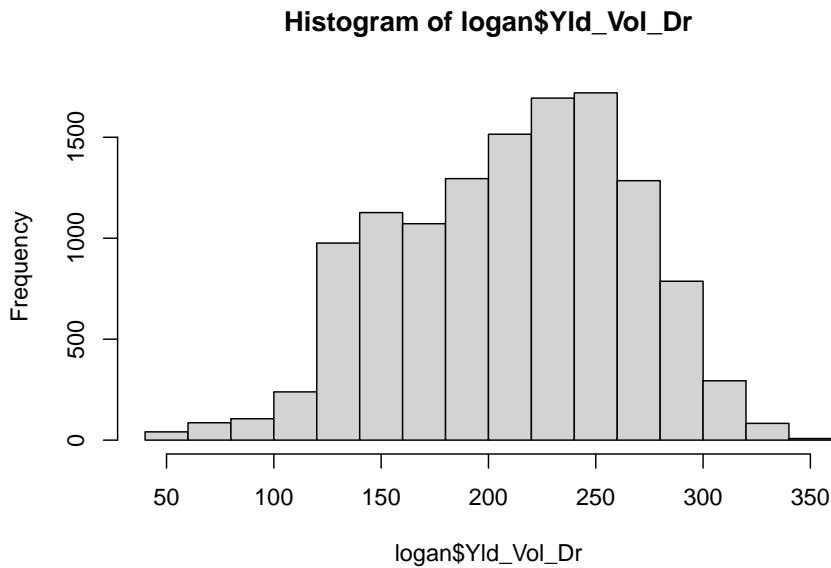
In the map above, the higher-yielding areas are colored green, while the lower-

#### 14.1. SCENARIO 1: YIELD MAP (POPULATION SUMMARY AND Z-DISTRIBUTION)549

yielding areas are orange to red. How would we summarise the yields above for others?

To start with, let's describe the center of the population. Should we use the mean or median? Either is appropriate if the data are normally distributed; if the data are skewed, the median will be a better measure of center. We can check the distribution using a histogram:

```
hist(logan$Yld_Vol_Dr)
```



We inspect the data to see if it fits a normal distribution (“bell”) curve. This histogram is admittedly ugly, but is roughly symmetrical. We can also quickly inspect our distribution using the summary command on our yield variable.

```
summary(logan$Yld_Vol_Dr)
```

```
##   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## 50.1 169.2 216.6 211.1 252.3 348.2
```

The median and mean are similar, so it would be acceptable to use either to describe the population center. Note the median is a little larger than the mean, so our data is skewed slightly to the right.

How would we describe the spread? That's right: we would use the standard deviation:

```
sd(logan$Yld_Vol_Dr)
```

```
## [1] 54.10512
```

Our standard deviation is about 54 bushels. What is the significance of this? We would expect 95% of the individuals (the yield measures) in this population to be within 1.96 standard deviations, or  $1.96 \times 54.1 = 106.0$  \$, of the mean. Rounding our mean to 211, we would expect any value less than  $211 - 106 = 105$  or greater than  $211 + 105 = 316$  to occur rarely in our population.

## 14.2 Scenario 2: Yield Estimate (Sampling t-Distribution)

*We are on a “Pro-Growers” tour of the Corn Belt or we are estimating yield on our own farm in anticipation of harvest or marketing grain.*

Let’s take our field above and sample 20 ears from it. After counting rows around the ear and kernels per row, we arrive at the following 20 estimates.

```
## [1] 256 136 244 154 191 241 292 225 183 277 249 207 290 203 213 226 266 275 151
## [20] 181
```

The mean of our samples is 223 bushels per acre. We know this is probably not the actual mean yield, but how can we define a range of values that is likely to include the true mean for this field?

We can calculate a 95% confidence interval for this field. That confidence interval defines a fixed distance above and below our sample mean. Were we to repeat our sampling 100 times, in about 95 of our samples the population mean would be within our confidence interval.

Remember, our confidence interval is calculated as:

$$CI = \bar{x} + t_{\alpha, df} \times SE$$

Where  $\bar{x}$  is the sample mean,  $t$  is the t-value, based on our desired level of confidence and the degrees of freedom, and  $SE$  is the standard error of the mean. In this case, we desire 95% confidence and have 19 degrees of freedom (since we have 20 samples). The t-value to use in our calculation is therefore:

```
t_value = qt(0.975, 20)
t_value
```

```
## [1] 2.085963
```

Our standard error is equal to the standard deviation of our samples.

```
yield_sd = sd(yield_sample)
yield_sd
```

```
## [1] 46.97144
```

Our standard error of the mean is our sample standard deviation, divided by the square root of the number of observations (20 ears) in the sample:

```
SE = yield_sd/sqrt(20)
SE
```

```
## [1] 10.50313
```

Our confidence interval has a lower limit of  $223 - 2.09 * 10.5 = 201.1$  and an upper limit of  $223 + 2.09 * 10.5 = 244.9$ . We would present this confidence interval as:

$$(201.1, 244.9)$$

We know the combine map above the true population mean was 211.1, which is included in our confidence interval.

### 14.3 Scenario 3: Side-By-Side (t-Test)

*You are a sales agronomist and want to demonstrate a new product to your customer. You arrange to conduct a side-by-side trial on their farm.*

Knowing every field has soil variations, you divide a field into 8 pairs of strips. Each strip in a pair is treated either with the farmer's current system (the control) or the farmer's system *plus* the new product. You create the following paired plot layout.

plots		
		treatment
802		control
801		treatment
702		control
701		control
602		treatment
601		treatment
502		control
501		treatment
402		control
401		treatment
302		control
301		treatment
202		control
201		treatment
102		treatment
101		control

We observe the following values in our trial:

block	treatment	yield
1	control	172.2
1	treatment	170.9
2	treatment	168.7
2	control	169.1
3	treatment	178.3
3	control	144.8
4	control	178.5
4	treatment	183.5
5	control	177.6
5	treatment	193.6
6	treatment	186.8
6	control	176.5
7	control	172.7
7	treatment	196.0
8	control	188.9
8	treatment	183.7

We run a t-test, as we learned in Units 4 and 5, which calculates the probability the difference between the control and treatment is equal to zero. Because we, as sales persons, are only interested in whether our treatment produces greater yield, we run a one-sided test. Our null hypothesis is therefore that the treatment produces yield equal to or less than the control. Our alternative

hypothesis (the one we hope to confirm) is that the treatment yields more than the control.

```
t_test = t.test(yield ~ treatment, data=yields, paired=TRUE, alternative="less")
t_test
```

```
##
##  Paired t-test
##
## data: yield by treatment
## t = -2.1424, df = 7, p-value = 0.03469
## alternative hypothesis: true difference in means is less than 0
## 95 percent confidence interval:
##       -Inf -1.174144
## sample estimates:
## mean of the differences
##                      -10.15
```

In the t-test above, we tested the difference when we subtracted the control yield from the treatment yield. We hoped this difference would be less than zero, which it would be if the treatment yield exceeded the control yield. We see the difference, -10.15, was indeed less than zero. Was it significant? Our p-value was 0.03, indicating a small probability that the true difference was actually zero or greater than zero.

We also see our confidence interval does not include zero or any positive values. We can therefore report to the grower that our treatment yielded more than the control.

## 14.4 Scenario 4: Fungicide Trial (ANOVA CRD or RCBD)

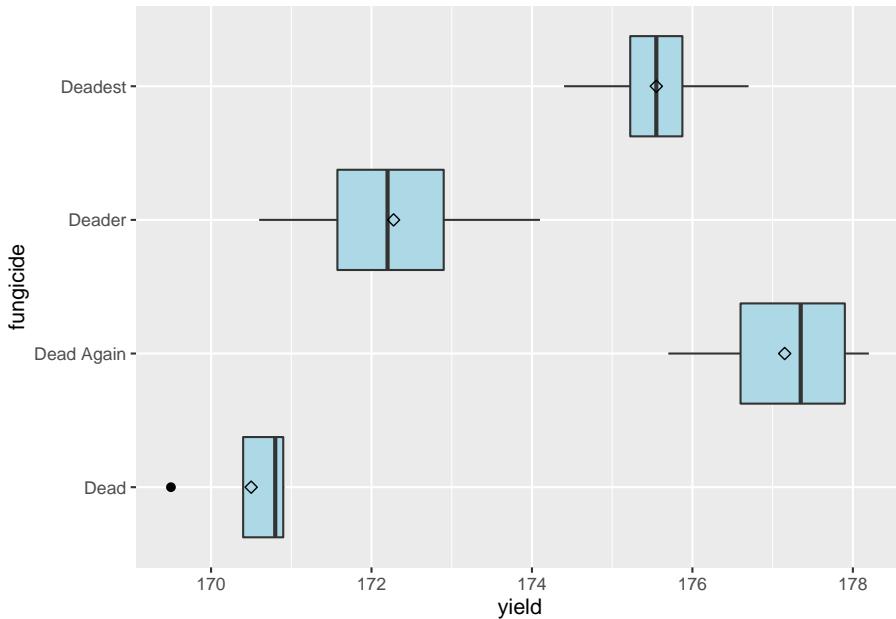
*We want to compare three or more fungicides that differ in their qualities, as opposed to their quantity.*

We design and conduct a randomized complete block design experiment in which four fungicides are compared:

plot	block	fungicide	yield
101	1	Deadest	174.4
102	1	Dead	169.5
103	1	Dead Again	176.9
104	1	Deader	170.6
201	2	Dead Again	175.7
202	2	Deadest	175.5
203	2	Deader	171.9
204	2	Dead	170.9
301	3	Deader	172.5
302	3	Dead Again	178.2
303	3	Dead	170.9
304	3	Deadest	175.6
401	4	Deadest	176.7
402	4	Deader	174.1
403	4	Dead Again	177.8
404	4	Dead	170.7

We can begin our analysis of results by inspecting the distribution of observations within our treatment. We can take a quick look at our data with the boxplot we learned in Unit 9.

```
fungicide_final %>%
  ggplot(aes(x=fungicide, y=yield)) +
  geom_boxplot(outlier.colour="black", outlier.shape=16,
               outlier.size=2, notch=FALSE, fill="lightblue") +
  coord_flip() +
  stat_summary(fun=mean, geom="point", shape=23, size=2)
```



Hmmmm, fungicide “Dead” has an outlier. Shoots. Let’s look more closely at our situation. First, back to our field notes. We check our plot notes – nothing appeared out of the ordinary about that plot. Second, we notice the “Dead” treatment has a tighter distribution than the other three treatments: our outlier would not be an outlier had it occurred in the distribution of Deader. Finally, we note the outlier differs from from the mean and median of “Dead” by only a few bushels – a deviation we consider to be reasonable given our knowledge of corn production. We conclude the outlier can be included in the dataset.

We will go ahead and run an Analysis of Variance on these data, as we learned in Unit 8. Our linear model is:

$$Y_{ij} = \mu + B_i + T_j + BT_{ij}$$

Where  $\mu$  is the population mean,  $Y$  is the yield of the  $j_{th}$  treatment in the  $i_{th}$  block,  $B$  is the effect of the  $i_{th}$  block,  $T$  is the effect of the  $j_{th}$  treatment, and  $BT_{ij}$  is the interaction between block and treatment. This model forms the basis of our model statement to R:

```
fungicide_model = aov(yield ~ block + fungicide, data=fungicide_final)
summary(fungicide_model)
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
## block	3	9.10	3.03	5.535	0.0197 *
## fungicide	3	109.93	36.64	66.884	1.79e-06 ***

```
## Residuals     9   4.93    0.55
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We see the effect of the fungicide treatment is highly significant, so we will separate the means using the LSD test we learned in Unit 8.

```
library(agricolae)
lsd = LSD.test(fungicide_model, "fungicide")
lsd

## $statistics
##      MSerror Df      Mean       CV t.value      LSD
## 0.5478472  9 173.8688 0.4257045 2.262157 1.183961
##
## $parameters
##      test p.adjusted name.t ntr alpha
## Fisher-LSD      none fungicide  4 0.05
##
## $means
##           yield      std.r      LCL      UCL      Min      Max      Q25      Q50
## Dead        170.500 0.6733003 4 169.6628 171.3372 169.5 170.9 170.400 170.80
## Dead Again 177.150 1.1090537 4 176.3128 177.9872 175.7 178.2 176.600 177.35
## Deader      172.275 1.4522970 4 171.4378 173.1122 170.6 174.1 171.575 172.20
## Deadest     175.550 0.9398581 4 174.7128 176.3872 174.4 176.7 175.225 175.55
##           Q75
## Dead        170.900
## Dead Again 177.900
## Deader      172.900
## Deadest     175.875
##
## $comparison
## NULL
##
## $groups
##           yield groups
## Dead Again 177.150     a
## Deadest     175.550     b
## Deader      172.275     c
## Dead        170.500     d
##
## attr(,"class")
## [1] "group"
```

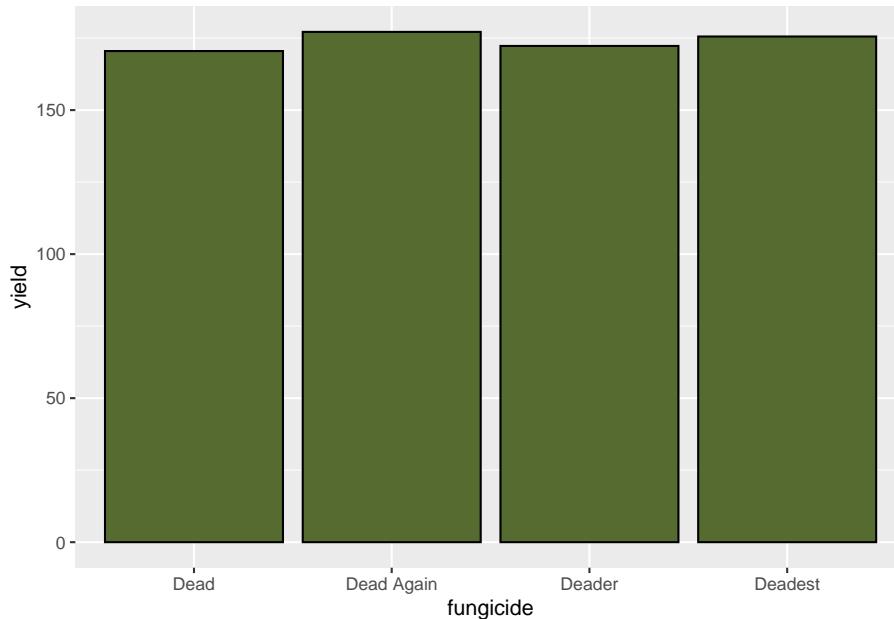
Each of the fungicides produced significantly different yields, with “Dead Again” the highest-yielding. We can plot these results in a bar-plot.

```

lsd_groups = lsd$groups %>%
  rownames_to_column("fungicide")

lsd_groups %>%
  ggplot(aes(x=fungicide, y=yield)) +
  geom_bar(stat = "identity", fill="darkolivegreen", color="black")

```



## 14.5 Scenario 5: Hybrid Response to Fungicide Trial (ANOVA Factorial or Split Plot)

We want to test two factors within the same trial: three levels of hybrid (Hybrids “A”, “B”, and “C”) and two levels of fungicide (“treated” and “untreated”).

The treatments are arranged in a factorial randomized complete block design, like we learned in Unit 7.

plot	block	fungicide	hybrid	yield
101	R1	treated	B	199.1
102	R1	untreated	A	172.1
103	R1	untreated	B	187.4
104	R1	treated	C	204.1
105	R1	untreated	C	195.5
106	R1	treated	A	187.2
201	R2	treated	A	195.7
202	R2	untreated	C	200.3
203	R2	treated	B	200.3
204	R2	treated	C	216.5
205	R2	untreated	A	185.0
206	R2	untreated	B	188.4
301	R3	treated	B	209.4
302	R3	treated	A	198.2
303	R3	untreated	A	185.6
304	R3	untreated	C	203.6
305	R3	untreated	B	194.2
306	R3	treated	C	220.4
401	R4	untreated	B	198.7
402	R4	treated	B	214.4
403	R4	treated	C	219.2
404	R4	treated	A	194.8
405	R4	untreated	A	191.0
406	R4	untreated	C	199.6

Our linear additive model is:

$$Y_{ijk} = \mu + B_i + F_j + H_k + FH_{jk} + BFH_{ijk}$$

where  $Y_{ijk}$  is the yield in the  $i$ th block with the  $j$ th level of fungicide and the  $k$ th level of hybrid,  $\mu$  is the population mean,  $B_i$  is the effect of the  $i$ th block,  $F_j$  is the effect of the  $j$ th level of fungicide,  $H_k$  is the effect of the  $k$ th level of hybrid,  $FH_{jk}$  is the interaction of the  $j$ th level of fungicide and  $k$ th level of hybrid, and  $BFH_{ijk}$  is the interaction of block, fungicide, and hybrid.

This translates to the following model statement and analysis of variance:

```
fung_hyb_model = aov(yield ~ block + fungicide + hybrid + fungicide*hybrid, data = fung)
summary(fung_hyb_model)
```

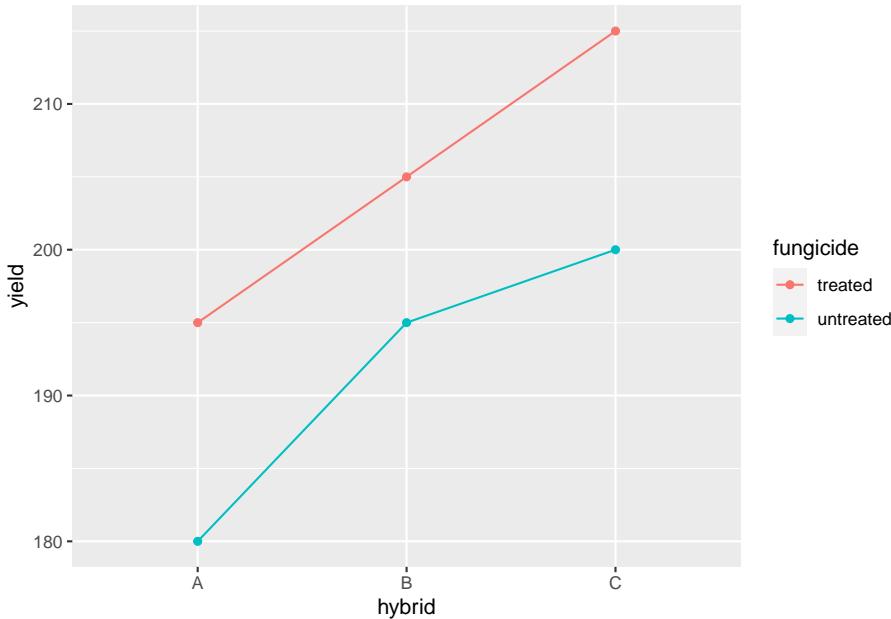
	Df	Sum Sq	Mean Sq	F value	Pr(>F)
## block	3	538.1	179.4	15.92	6.27e-05 ***
## fungicide	1	1038.9	1038.9	92.22	8.49e-08 ***

14.5. SCENARIO 5: HYBRID RESPONSE TO FUNGICIDE TRIAL (ANOVA FACTORIAL OR SPLIT PLOT) 559

```
## hybrid           2 1403.4   701.7   62.29 5.43e-08 ***
## fungicide:hybrid 2    23.2    11.6     1.03    0.381
## Residuals        15  169.0    11.3
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The Analysis of Variance results above, show that the main effects – fungicide and hybrid – are both highly-significant, but the interaction between fungicide and hybrid is insignificant. The line plot below allows us to further examine that interaction.

```
fung_hyb %>%
  group_by(fungicide, hybrid) %>%
  summarise(yield = mean(yield)) %>%
  ungroup %>%
  ggplot(aes(x=hybrid, y=yield, group=fungicide)) +
  geom_point(aes(color=fungicide)) +
  geom_line(aes(color=fungicide))
```



We can perform means separation on the data the same as we did for our analysis of variance in the previous example. Since fungicide only has two levels, its significance in the analysis of variance means the two levels (“treated” and “untreated”) are significant. To separate the hybrid levels, we can use the least significant difference test.

```
lsd_fung_hybrid = LSD.test(fung_hyb_model, "hybrid")
lsd_fung_hybrid
```

```
## $statistics
##      MSerror Df      Mean       CV t.value      LSD
##    11.26542 15 198.3625 1.692053 2.13145 3.576998
##
## $parameters
##           test p.adjusted name.t ntr alpha
## Fisher-LSD      none hybrid   3  0.05
##
## $means
##      yield     std.r      LCL      UCL     Min     Max     Q25     Q50     Q75
## A 188.7000 8.305420 8 186.1707 191.2293 172.1 198.2 185.450 189.10 195.025
## B 198.9875 9.388966 8 196.4582 201.5168 187.4 214.4 192.750 198.90 202.575
## C 207.4000 9.777817 8 204.8707 209.9293 195.5 220.4 200.125 203.85 217.175
##
## $comparison
## NULL
##
## $groups
##      yield groups
## C 207.4000     a
## B 198.9875     b
## A 188.7000     c
##
## attr(,"class")
## [1] "group"
```

Our means separation results suggest the three hybrids differ in yield.

## 14.6 Scenario 6: Foliar Rate-Response Trial (Linear or Non-Linear Regression)

*We want to model how the effect of a foliar product on yield increases with rate, from 1X to 4X.*

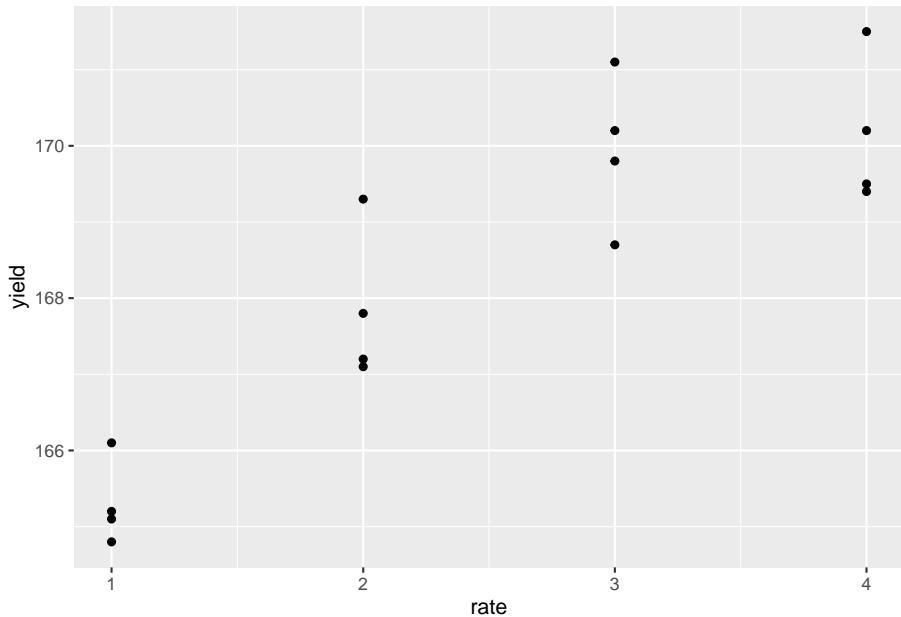
The data are below:

plot	block	rate	yield
101	1	2	167.1
102	1	3	168.7
103	1	4	169.5
104	1	1	165.2
201	2	2	167.2
202	2	1	164.8
203	2	4	169.4
204	2	3	170.2
301	3	2	167.8
302	3	4	170.2
303	3	1	166.1
304	3	3	169.8
401	4	2	169.3
402	4	3	171.1
403	4	4	171.5
404	4	1	165.1

We should start by plotting our data with a simple scatter plot so we can observe the nature of the relationship between Y and X. Do their values appear to be associated? Is their relationship linear or nonlinear?

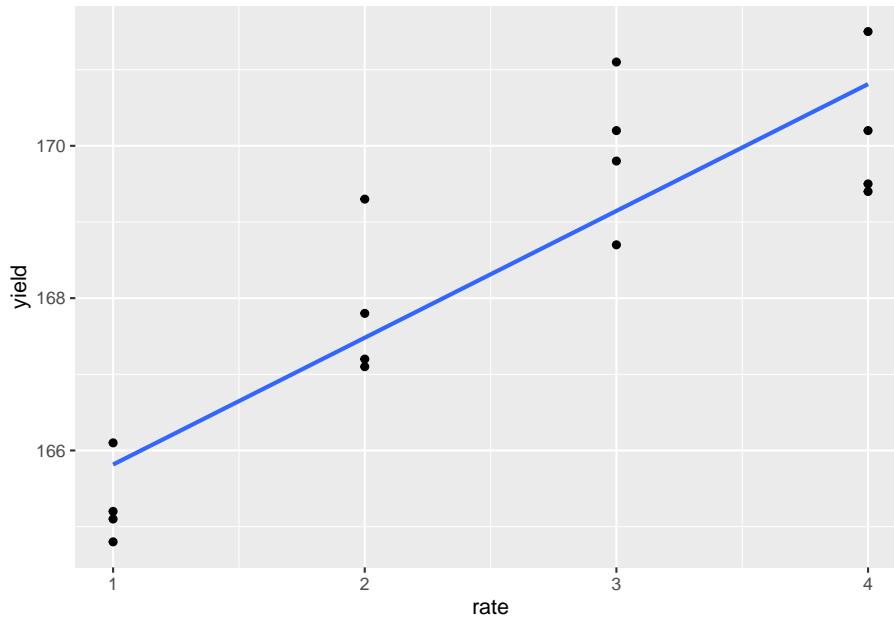
```
p = foliar_final %>%
  ggplot(aes(x=rate, y=yield)) +
  geom_point()

p
```



The response appears to be nonlinear, but we first try to fit the relationship with simple linear regression, as we learned in Unit 10. Our regression line is plotted with the data below:

```
p + geom_smooth(method = "lm", se=FALSE)
```



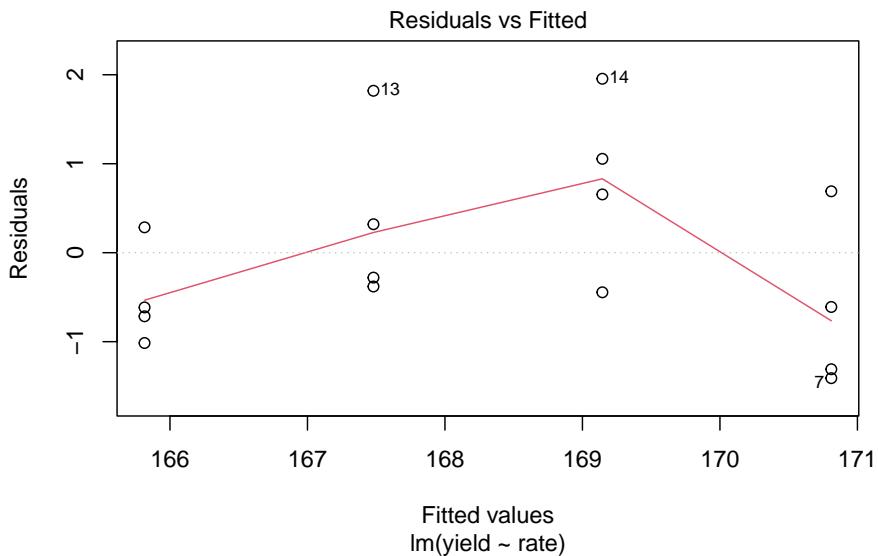
We also run an analysis of variance on the regression, modelling yield as a function of rate, which produces the following results:

```
foliar_linear_model = lm(yield~rate, data = foliar_final)
summary(foliar_linear_model)

##
## Call:
## lm(formula = yield ~ rate, data = foliar_final)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.4100 -0.6400 -0.3300  0.6637  1.9550
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 164.1500    0.6496 252.70 < 2e-16 ***
## rate         1.6650    0.2372   7.02 6.06e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.061 on 14 degrees of freedom
## Multiple R-squared:  0.7787, Adjusted R-squared:  0.7629
## F-statistic: 49.27 on 1 and 14 DF,  p-value: 6.057e-06
```

Not bad. The slope (rate effect) is highly significant and the  $R^2 = 0.75$ . To see whether the linear model was appropriate, however, we should plot the residuals.

```
plot(foliar_linear_model, which = 1)
```



We see, as we might expect, the residuals are not distributed randomly around the regression line. The middle two yields are distributed mostly above the regression line, while the highest and lowest yields are distributed mostly below the regression line.

A linear model is probably not the best way to model our data. Lets try, instead, to fit the data with a asymptotic model as we did in Unit 11. This model, in which the value of Y increases rapidly at lower levels of X, but then plateaus at higher levels of X, is often also referred to as a monomolecular function.

```
foliar_monomolecular = stats::nls(yield ~ SSasymp(rate, init, m, plateau), data=foliar_fi
summary(foliar_monomolecular)

##
## Formula: yield ~ SSasymp(rate, init, m, plateau)
##
## Parameters:
##             Estimate Std. Error t value Pr(>|t|)
## init      171.1636    1.2699 134.789   <2e-16 ***
##
```

#### 14.6. SCENARIO 6: FOLIAR RATE-RESPONSE TRIAL (LINEAR OR NON-LINEAR REGRESSION)565

```
## m      159.7711    3.0120  53.044   <2e-16 ***
## plateau -0.4221    0.4880 -0.865    0.403
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9104 on 13 degrees of freedom
##
## Number of iterations to convergence: 0
## Achieved convergence tolerance: 4.011e-06
```

We are successful in fitting our nonlinear model. To plot it with our data, however, we have to build a new dataset that models yield as a function of rate, using our new model.

To do this, we first create a dataset with values of rate from 1X and 4X, in increments of tenths.

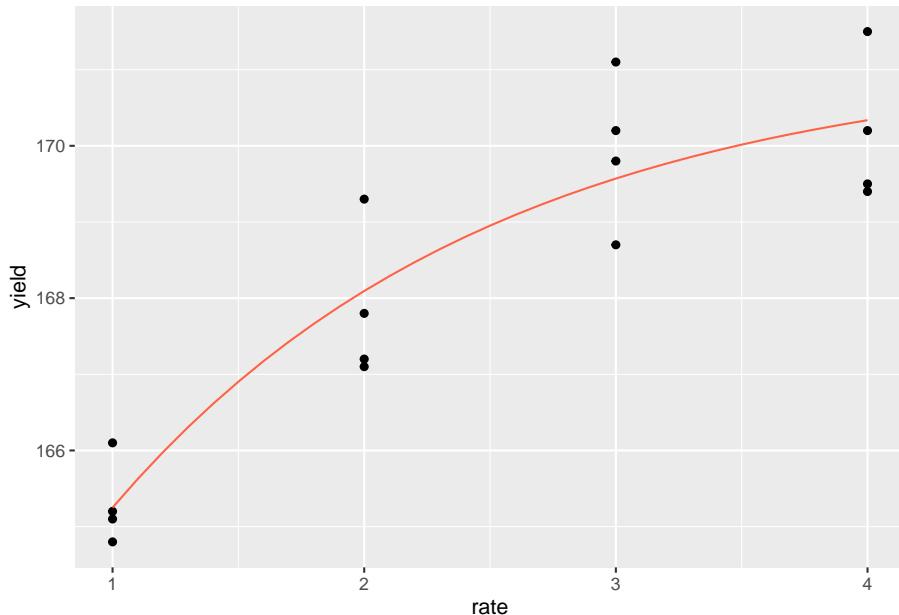
```
foliar_predicted = data.frame(
  rate =
    seq(from=1,to=4,by=0.1)
)
```

We then can predict yield as a function of rate, using the new model.

```
foliar_predicted$yield = predict(foliar_monomolecular, foliar_predicted)
```

Finally, we can plot our curve and visually confirm it fits the data better than our linear model.

```
p + geom_line(data = foliar_predicted, aes(x=rate, y=yield), color="tomato")
```



## 14.7 Scenario 7: Application Map (Shapefiles and Rasters)

We grid sample our field and want to visualize the results. We are particularly interested in our soil potassium results. We want to first visualize the point values, then create a raster map to predict potassium values throughout the field.

We start by reading in the shapefile with our results.

obs	Sample_id	SampleDate	ReportDate	P2	Grower	Field	attribute	measure
29	21	10/26/2018	10/30/2018	0	Tom Besch	Folie N & SE	Om	3
30	22	10/26/2018	10/30/2018	0	Tom Besch	Folie N & SE	Om	3
31	27	10/26/2018	10/30/2018	0	Tom Besch	Folie N & SE	Om	3
32	5	10/26/2018	10/30/2018	0	Tom Besch	Folie N & SE	Om	3
33	6	10/26/2018	10/30/2018	0	Tom Besch	Folie N & SE	Om	3
34	8	10/26/2018	10/30/2018	0	Tom Besch	Folie N & SE	Om	3

Our next step is to filter our results to potassium ("K") only.

```
k_only = folie %>%
  filter(attribute=="K")
```

We want to color-code our results with green for the greatest values, yellow for

intermediate values, and red for the lowest values. We can do this using the colorBin function.

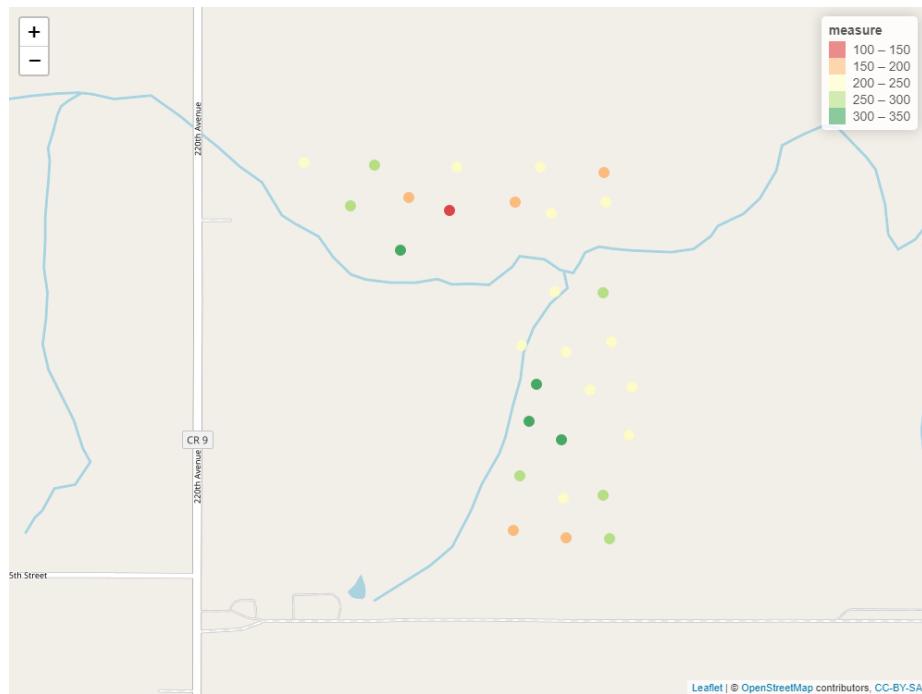
```
library(leaflet)
library(grDevices)

pal_k = colorBin("RdYlGn", k_only$measure)
```

We then create our map using the leaflet() function, just as we learned in Unit 12.

```
m <- k_only %>%
  leaflet() %>%
  addTiles() %>%
  addCircleMarkers(
    fillColor = ~pal_k(measure),
    radius = 6,
    weight = 0,
    fillOpacity = 0.8
  ) %>%
  addLegend(values = ~measure,
            pal = pal_k)

#above code won't knit -- created static image for markdown
saveWidget(m, "temp.html", selfcontained = FALSE)
webshot("temp.html", cliprect = "viewport")
```



Next, we want to create a raster map that predicts soil potassium levels throughout our field. We first need to define the field boundary, which we do by loading a shapefile that defines a single polygon that outlines our field.

```
boundary = st_read("data-unit-14/Folie N &SE_boundary.shp", quiet=TRUE)
```

We then use that boundary polygon to create a grid. Each cell of this grid will be filled in when we create our raster.

```
library(stars)

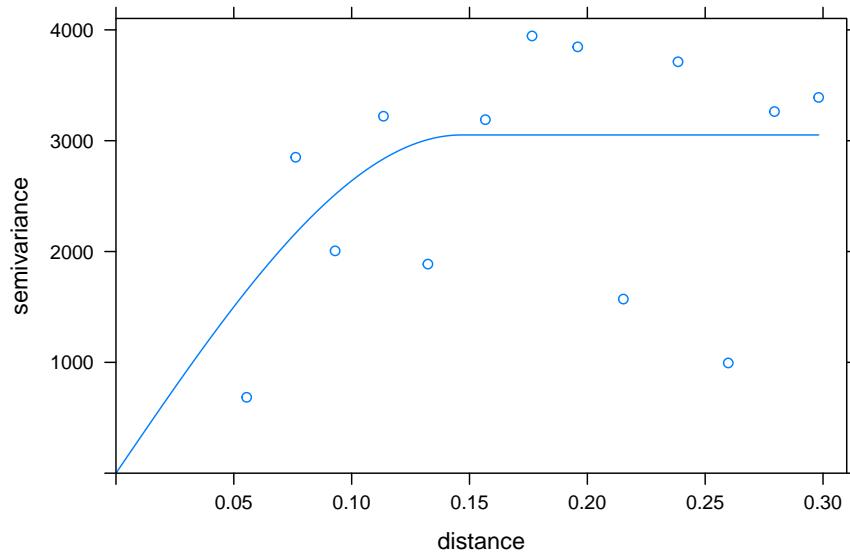
### make grid
grd = st_bbox(boundary) %>%
  st_as_stars() %>%
  st_crop(boundary)
```

Each cell in our raster that does not coincide with a test value will be predicted as the mean of the values of other soil test points. Since soils that are closer together are more alike than those that are farther apart, soil test points that are closer to the estimated cell will be weighted more heavily in calculating the mean than those more distant. How should they be weighted?

To answer this, we fit a variogram to the data. The variogram describes how the correlation between points changes with distance.

```
library(gstat)

v = variogram(measure~1, k_only)
m = fit.variogram(v, vgm("Sph"))
plot(v, model = m)
```



We are now able to interpolate our data with kriging, which incorporates the results of our variogram in weighting the effect of sample points in the estimate of a cell value.

```
kriged_data = gstat::krige(formula = measure~1, k_only, grd, model=m)
```

```
## [using ordinary kriging]
```

We finish by plotting our kriged data using leaflet().

```
library(RColorBrewer)
library(leaflet)

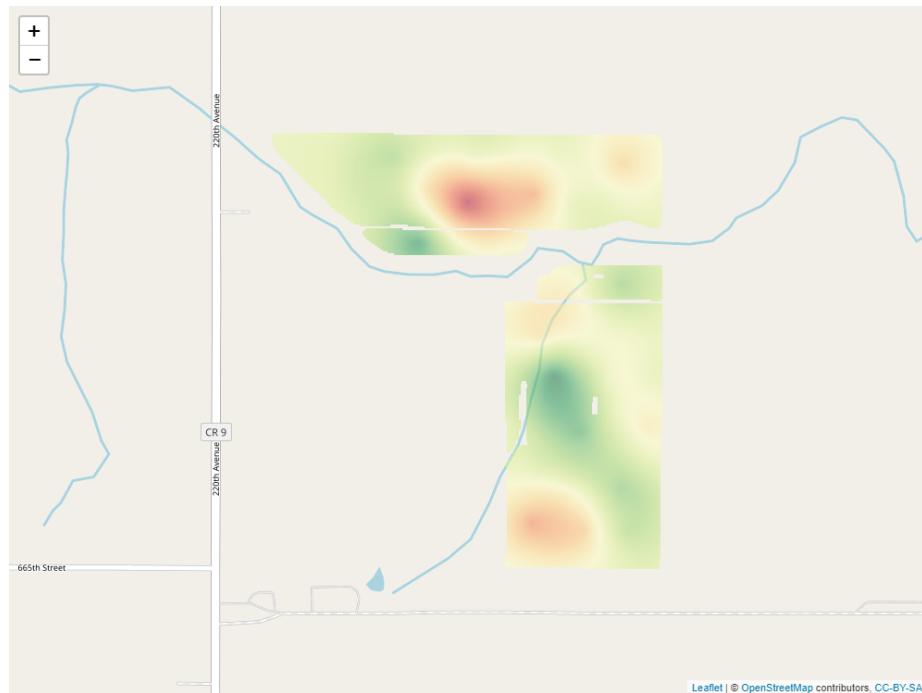
m <- kriged_data %>%
  leaflet() %>%
  addTiles() %>%
  addStarsImage(opacity = 0.5,
```

```

colors="RdYlGn")

#above code won't knit -- created static image for markdown
saveWidget(m, "temp.html", selfcontained = FALSE)
webshot("temp.html", cliprect = "viewport")

```



## 14.8 Scenario 8: Yield Prediction (Multiple Linear Regression and other Predictive Models)

*We want to predict the yield for a particular hybrid across 676 midwestern counties, based on over 300 observations of that hybrid.*

We start by reading in the shapefile with our hybrid data.

```

## Reading layer `hybrid_data' from data source
##   `C:\ds_ag_professionals\data-unit-14\hybrid_data.shp' using driver `ESRI Shapefile'
## Simple feature collection with 4578 features and 4 fields
## Geometry type: POINT
## Dimension:      XY

```

#### 14.8. SCENARIO 8: YIELD PREDICTION (MULTIPLE LINEAR REGRESSION AND OTHER PREDICTIVE MODELS)

```

## Bounding box: xmin: -104.8113 ymin: 37.56348 xmax: -81.08706 ymax: 46.24
## Geodetic CRS: WGS 84

## Simple feature collection with 6 features and 4 fields
## Geometry type: POINT
## Dimension: XY
## Bounding box: xmin: -99.80072 ymin: 42.54919 xmax: -89.77396 ymax: 43.77493
## Geodetic CRS: WGS 84
##   book_name year attribute      value      geometry
## 1 ADAMS-MN 2016 bu_acre 215.4100 POINT (-92.26931 43.57388)
## 2 Adams-WI 2016 bu_acre 257.0930 POINT (-89.77396 43.77493)
## 3 Ainsworth-NE 2016 bu_acre 236.5837 POINT (-99.80072 42.55031)
## 4 Ainsworth-NE 2017 bu_acre 244.3435 POINT (-99.80072 42.55031)
## 5 Ainsworth-NE 2019 bu_acre 220.0456 POINT (-99.79738 42.54919)
## 6 Algona-IA 2016 bu_acre 216.0600 POINT (-94.28736 42.97575)

```

The attributes are organized in the long form so we will want to “spread” or pivot them to the wide form first.

```

hybrid_wide = hybrid %>%
  spread(attribute, value)
head(hybrid_wide)

## Simple feature collection with 6 features and 16 fields
## Geometry type: POINT
## Dimension: XY
## Bounding box: xmin: -99.80072 ymin: 42.54919 xmax: -89.77396 ymax: 43.77493
## Geodetic CRS: WGS 84
##   book_name year bu_acre clay om prcp_0_500 prcp_1001_1500
## 1 ADAMS-MN 2016 215.4100 23.939049 149.7867 152 167
## 2 Adams-WI 2016 257.0930 9.899312 438.3402 140 112
## 3 Ainsworth-NE 2016 236.5837 11.701055 143.7150 97 63
## 4 Ainsworth-NE 2017 244.3435 11.701055 143.7150 148 16
## 5 Ainsworth-NE 2019 220.0456 11.701055 143.7150 234 22
## 6 Algona-IA 2016 216.0600 25.283112 480.4145 96 144
##   prcp_1501_2000 prcp_501_1000 sand silt tmean_0_500 tmean_1001_1500
## 1             84          122 16.96170 59.09925 14.32143 21.91071
## 2            152           39 78.38254 11.71815 16.74286 22.30357
## 3             79           50 69.87984 18.41910 15.72857 22.38095
## 4            212           12 69.87984 18.41910 15.96428 26.11905
## 5            142          116 69.87984 18.41910 16.32143 23.82143
## 6             55           72 37.38857 37.32832 16.19286 21.26191
##   tmean_1501_2000 tmean_501_1000 whc      geometry
## 1        22.05952     21.45238 27.98555 POINT (-92.26931 43.57388)
## 2       21.83333     20.13095 15.67212 POINT (-89.77396 43.77493)

```

```

## 3      23.77381    24.00000 18.41579 POINT (-99.80072 42.55031)
## 4      21.34821    20.64286 18.41579 POINT (-99.80072 42.55031)
## 5      22.46429    21.69048 18.41579 POINT (-99.79738 42.54919)
## 6      22.89286    23.32143 33.21381 POINT (-94.28736 42.97575)

```

Similarly, lets load in the county data. Like the hybrid dataset, it is in long form, so lets again spread or pivot it to the long form so that the attributes each have their own column.

```

county_climates = st_read("data-unit-14/county_climates.shp")

## Reading layer `county_climates' from data source
##   `C:\ds_ag_professionals\data-unit-14\county_climates.shp' using driver `ESRI Shapefile'
## Simple feature collection with 8788 features and 8 fields
## Geometry type: MULTIPOLYGON
## Dimension:      XY
## Bounding box:  xmin: -106.1954 ymin: 36.99782 xmax: -80.51869 ymax: 47.24051
## Geodetic CRS:  WGS 84

county_climates_wide = county_climates %>%
  spread(attribt, value)

head(county_climates_wide)

## Simple feature collection with 6 features and 19 fields
## Geometry type: MULTIPOLYGON
## Dimension:      XY
## Bounding box:  xmin: -105.6932 ymin: 38.61245 xmax: -102.0451 ymax: 40.26278
## Geodetic CRS:  WGS 84
##   stco stt_bbv stt_fps          cnty_nm fps_cls stat_nm    clay      om
## 1 8001     CO     08    Adams County      H1 Colorado 21.88855 102.0773
## 2 8005     CO     08  Arapahoe County      H1 Colorado 23.87947 102.4535
## 3 8013     CO     08  Boulder County      H1 Colorado 25.83687 106.2404
## 4 8017     CO     08 Cheyenne County      H1 Colorado 23.67021 105.8340
## 5 8039     CO     08   Elbert County      H1 Colorado 22.00638 117.1232
## 6 8063     CO     08 Kit Carson County      H1 Colorado 24.78930 159.3644
##   prcp_0_500 prcp_1001_1500 prcp_1501_2000 prcp_501_1000      sand      silt
## 1    72.05263        44.15789       43.84211      31.68421 47.83218 30.27927
## 2    58.20000        43.20000       33.80000      17.00000 41.83093 34.28960
## 3   100.38462        47.84615       67.46154      58.07692 42.52472 31.63841
## 4    57.42105        54.52632       61.73684      37.26316 25.24085 51.08894
## 5    82.35714        57.64286       47.71429      49.28571 45.78082 32.21280
## 6    68.05263        54.26316       76.52632      43.36842 26.70159 48.50911
##   tmean_0_500 tmean_1001_1500 tmean_1501_2000 tmean_501_1000      whc

```

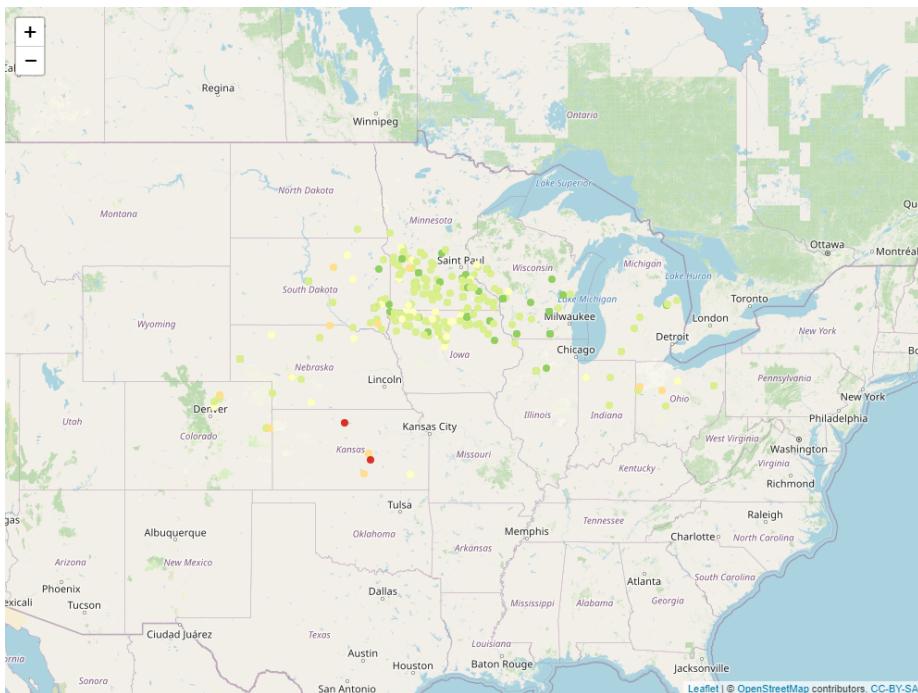
#### 14.8. SCENARIO 8: YIELD PREDICTION (MULTIPLE LINEAR REGRESSION AND OTHER PREDICTIVE MODELS)

```

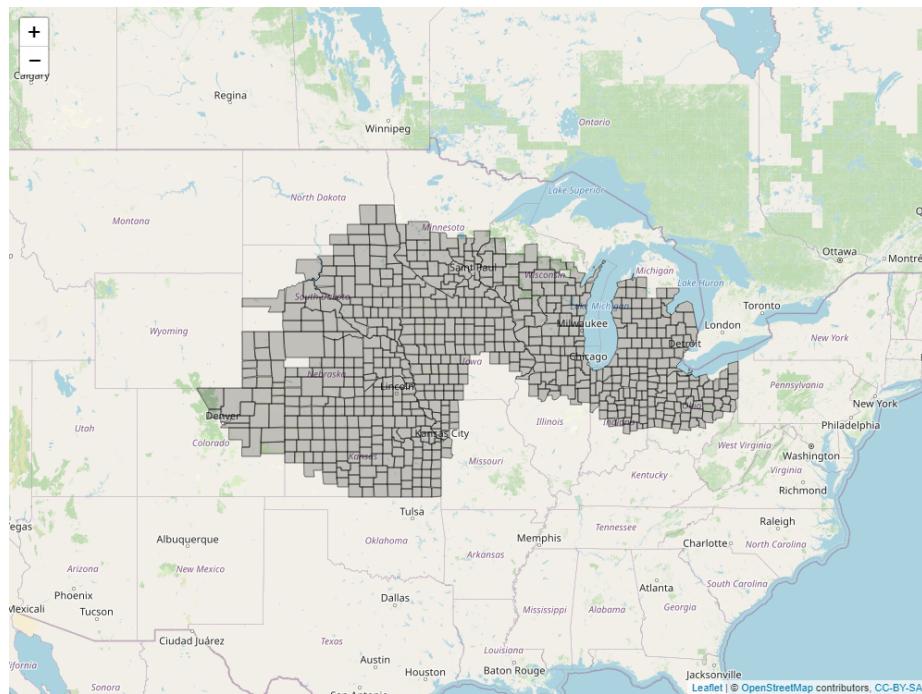
## 1 16.06967 24.17904 22.77083 21.65316 22.03449
## 2 15.64571 24.02857 22.46845 21.42024 21.34191
## 3 12.94062 18.60742 13.54932 19.00073 22.04795
## 4 17.02484 24.85072 24.18186 22.41526 30.14545
## 5 14.86390 22.27912 20.54928 20.41850 19.80896
## 6 16.74041 24.50439 23.56375 22.10329 26.92315
##
# geometry
## 1 MULTIPOLYGON (((-105.0529 3...
## 2 MULTIPOLYGON (((-105.0534 3...
## 3 MULTIPOLYGON (((-105.0826 3...
## 4 MULTIPOLYGON (((-103.1725 3...
## 5 MULTIPOLYGON (((-104.6606 3...
## 6 MULTIPOLYGON (((-103.1631 3...

```

First, lets plot the locations of our hybrid trials:



We see our trials were conducted mostly in northern Iowa and southern Minnesota, but also in several other states. We will want to constrain our predictions to counties in this general area. That requires a few steps that we didn't cover this semester (the very curious can look up "convex hulls"). Our county dataset has been limited to the appropriate counties for our predictions:



Our next step is to develop our random forest model to predict yield. Recall that the predictor variables in a random forest model are called features. Our data has the following features:

attribute	description
clay	% clay
om	% organic matter
prcp_0_500	precip from 0 to 500 GDD
prcp_1001_1500	precip from 1001 to 1500 GDD
prcp_1501_2000	precip from 1501 to 2000 GDD
prcp_501_1000	precip from 501 to 1000 GDD
sand	% sand
silt	% silt
tmean_0_500	mean temp from 0 to 500 GDD
tmean_1001_1500	mean temp from 1001 to 1500 GDD
tmean_1501_2000	mean temp from 1501 to 2000 GDD
tmean_501_1000	mean temp from 501 to 1000 GDD
whc	water holding capacity

GDD are growing degree days accumulated from the historical average date at which 50% of the corn crop has been planted in each county. For example, prcp\_0\_500 is the cumulative precipitation from 0 to 500 GDD after planting. This would correspond with germination and seedling emergence.

#### 14.8. SCENARIO 8: YIELD PREDICTION (MULTIPLE LINEAR REGRESSION AND OTHER PREDICTIVE MODELS)

We run our random forest the same as we learned in Unit 13. We have a couple of extraneous columns (`book_name` and `year`) in our `hybrid_wide` dataset. It is also a shapefile; we need to drop the geometry column and convert it into a dataframe before using it. We will do that first.

```
hybrid_df = hybrid_wide %>%
  st_drop_geometry() %>%
  dplyr::select(-c(book_name, year))
```

We will use 10-fold cross validation (indicated by the “`repeatedcv`” option in our `trainControl()` function below.)

```
library(caret)
library(randomForest)

ctrl = trainControl(method="repeatedcv", number=10, repeats = 1)
```

We can now fit our random forest model to the data.

```
hybridFit = train(bu_acre ~ .,
                   data = hybrid_df,
                   method = "rf",
                   trControl = ctrl)

hybridFit

## Random Forest
##
## 327 samples
## 13 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 295, 294, 295, 294, 295, 295, ...
## Resampling results across tuning parameters:
##
##   mtry   RMSE     Rsquared    MAE
##   2      34.52569  0.1436353  25.86010
##   7      34.95068  0.1426655  26.24718
##   13     35.42862  0.1309118  26.53165
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.
```

These values aren't great. The  $R^2$  of the optimal model (top line, mtry = 2) is only about 0.16 and the root mean square error is above 33 bushels. For simplicity in this example, we left out several additional environmental features. We might consider adding those back in and re-running the model.

Nonetheless, let's use our fit random forest model to predict yield across each county in our prediction space.

```
county_climates_wide$yield = predict(hybridFit, county_climates_wide)
```

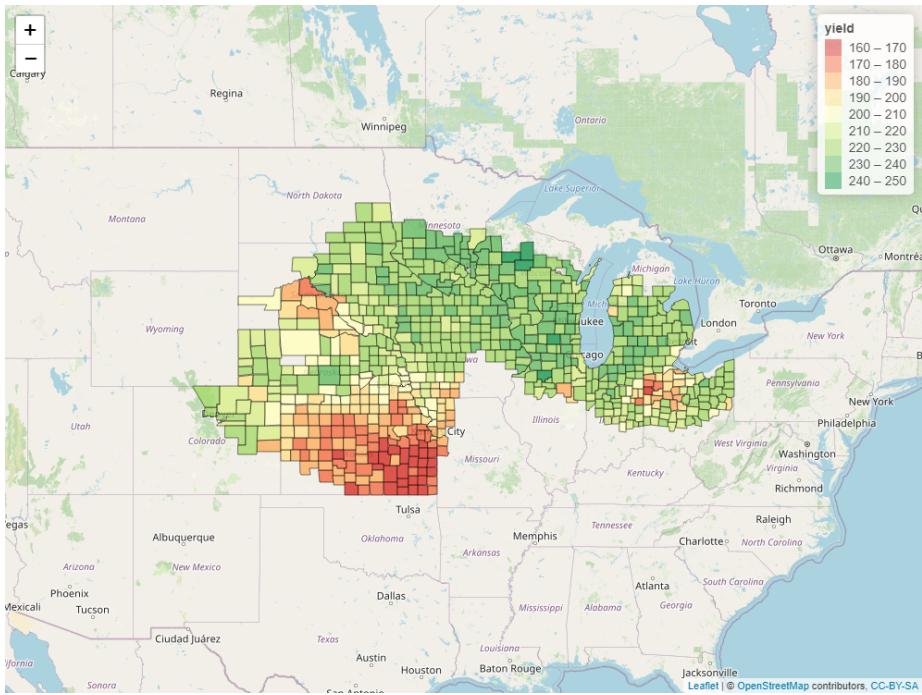
Finally, we can plot the results. We will again use a red-yellow-green scheme to code counties by yield.

```
pal_yield = colorBin("RdYlGn", county_climates_wide$yield)

m <- county_climates_wide %>%
  leaflet() %>%
  addTiles() %>%
  addPolygons(
    color = "black",
    fillColor=~pal_yield(yield),
    weight=1,
    fillOpacity = 0.8,
    popup = paste(county_climates_wide$cnty_nm, "<br>",
                 "Yield =", as.character(round(county_climates_wide$yield,1))), %>%
  addLegend(values = ~yield,
            pal = pal_yield)

#above code won't knit -- created static image for markdown
saveWidget(m, "temp.html", selfcontained = FALSE)
webshot("temp.html", cliprect = "viewport")
```

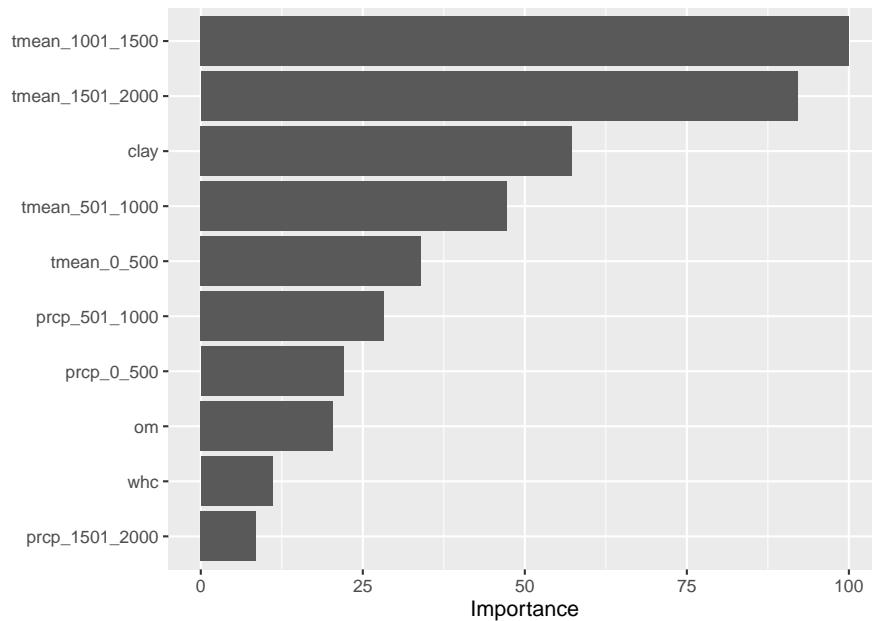
#### 14.8. SCENARIO 8: YIELD PREDICTION (MULTIPLE LINEAR REGRESSION AND OTHER PREDICTIVE MODELS)



Our hybrid is predicted to yield best in northern Iowa, southern Minnesota, Wisconsin, and Northern Illinois. It is predicted to perform less well in Western Ohio and Kansas.

One last question: of our features, which had the greatest effect on the yield of this hybrid? We can answer that by running the `vip()` function with our model.

```
library(vip)
vip(hybridFit)
```



We see that mean temperature during later-vegetative (1001-1500 GDD) and reproductive (1501-2000 GDD) phases had the most effect in our model, followed by clay content.

## 14.9 Summary

And that is it for our whirlwind review of *Data Science for Agricultural Professionals*. While each scenario is discussed briefly, it is my hope that seeing the major tools we have learned side-by-side will give you a better sense where to start with your analyses.

For the sake of brevity, we didn't cover every possible combination of these tools (for example, you should also inspect data distributions and perform means separation when working with factorial trials as well as simpler, single-factor trials). Once you have identified the general analysis to use, I encourage you to go back to the individual units for a more complete "recipe" how to conduct your analysis.

In fact, feel free to as a "cookbook" (in fact, several R texts label themselves as such), returning to it as needed to whip up a quick serving of t-tests, LSDs, or yield maps. Few of us ever memorize more than a small amount of material. Better to remember where to look it up in a hurry.

I hope you have enjoyed this text or, at least, found several pieces of it that can support you in your future efforts. I welcome your feedback and suggestions

how it might be improved. Thank you.