Introduction: Purpose and Goal

The purpose of this book is to provide an overview and tutorial for executing and critically interpreting various methods. A key contribution of this document is the ease to which you can switch between topics and search for concepts, this was its initial purpose in helping me study for exams and it may be usuful to you as a primer on graduate-level methods.

Beyond the more stream-lined methods courses, there are sections on causal inference, time series modeling, network analysis, and even brief sections devoted to courses on qualitative methods and survey design. This document includes hyperlinks for the table of contents and index for quick access to information. The scope of this book will be only in R, with rare mentions to other tools like Tableau, Power BI, Python, and SQL.

The book follows various data sets and topics from beer rankings to popular Reddit posts. I have tried my best to include copies of all data tables at the GitHub repository below. If the data is not available there the source will usually be mentioned in the text.

`https://github.com/KyleDavisGithub/Graduate_Methods_Handbook/tree/master/data`

Code for visualizations is included but sometimes hidden for readability. Ultimately this tutorial is less about replicating my results but to provide easily accessible tools for social scientists to use in their own research. That is, this document is only a first pass at a lot of these methods, to learn them better I would suggest using them and failing over and over until something is learned. This is also not an exhaustive list, I may reference something but never cover it later on. This only covers courses I've taken in my PhD program at Ohio State, but some of my insights from my work later on may be folded in to provide relevance for those in the business or government industry.

Code in this document will appear as follows:

```
> #  Example Code:
> X <- "String"
> inverse_logit <- function(x){
+      exp(x)/(1+exp(x))
+ }
```

Overall, I hope this document helps someone. Feel free to connect with me over any issues or questions you might have. Any questions or concerns can be sent to me via LinkedIn or GitHub issue reporting.

Created by Kyle Davis; https://www.linkedin.com/in/kyledavisln
Summer 2017 — August 17, 2022 (last updated)
For the most recent version:
`https://github.com/KyleDavisGithub/Graduate_Methods_Handbook`

Contents

1 Causal Inference and Assumptions

What is a *cause*? More specifically, what in social science can really "cause" anything, is it ever really measurable? These are the questions that have been plaguing scientists for years, and with increases in methodology and theory we have tried to answer causal questions — or at the very least, accurately report our assumptions and errors. A lot of the first few pages of this guide will be explicit in describing the assumptions we make with various methods, because these should be explicit for the audience when we find "causal effects."

Here we are using one language to define "cause," specifically that used by social scientists, which is the first theoretical assumption. A cause here, in the simplest terms that maximizes the utility of the word – **is a treatment**. Anytime you see the word "cause" you could also say treatment, and the reverse. Therefore, race, gender, any attribute is **not** a cause because we can not give people a new race or gender on a fundamental and societal level. But obviously these things matter! Here we would divide up our study, *stratifying* on gender, race, etc for all of those groups which we cannot assign a new race or gender. Being creative in design up front can save a lot of our analysis on the back-end too; for example, creating an online program that generates a name or image for you on some online messenger program may give you a new *perceieved race* or *perceieved gender* which we can randomize and identify. The key point here is that our definition of "causal" already introduces some problems, and a lot of what follows is attempting to advert them, understand them, and explain them accurately to the audience through *causal inference*. Causal inference, then, is the way we infer causality through many design and analytic choices. One take-away here is that by broadening our language we strengthen the assumptions we can make, it's a give-and-take that has become convention.

Symbol	**Meaning**
D	Indicating Treatment Group (1) or Control Group (0)
Y	Dependent Variable of Some Theoretical Interest
Y_0 and Y_1	Dependent Variable when Controlled or Treated
X or x_1 etc.	Independent Variables
V or σ^2	Variance
E[ ]	Expectation of Something, Typically the Mean
$E[Y\|D]$	Expectation of Y *Given Some* D
μ_{ATE}	The Average Treatment Effect
μ_{ATT}	The Average Treatment of the Treated Group
μ_{ATC}	The Average Treatment of the Control Group
$\xrightarrow{d}$	The Distribution Becomes
n or N	The Number of Observations
$\mathcal{N}$	Normally Distributed
$\hat{\theta}$	Estimated Parameters
X_{it}	All Rows Within X (i) and At Each Time (t)

1.1 Pearl vs. Rubin Models of Causal Inference

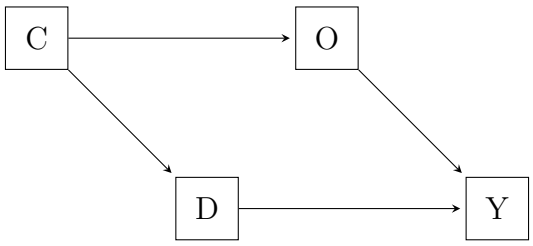
Working with similar definitions of "cause" Rubin and Pearl are two scholars that have had different approaches to tackling causal inference. Both focus on weighted averages (means) to make the jump between data to the "real world." Note that we could care about the distribution or variance instead of weighted means, but this won't be covered here. Rubin's approach is more algebraic, focusing on how attributes cannot be causal, and specifically stating when errors originate and cross between variables. Pearl leans on graphing theories, creating "DAG" plots (this will be covered later) to draw out for readers which variables influence others. Pearl's approach is intuitive and straightforward and seems to encapsulate theory just as well as Rubin's approach but has been criticized for not being explicit enough in error tracking between nodes of the theory plots. However, in some ways Pearl is more explicit algebraically, consider a fundamental mean difference in means between treatment and controls:

$$\text{Rubin approach:} \quad E[Y_1] - E[Y_0] \tag{1}$$

$$\text{Pearl approach:} \quad E[Y_1|\text{do}(D = 1)] - E[Y_0|\text{do}(D = 0)] \tag{2}$$

Pearl's approach, using the *do()* operator explicitly states what the researcher *did* and does not assume the reader has knowledge of what the researcher did in notation.

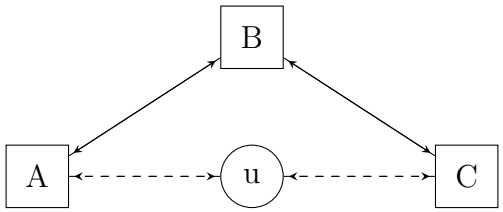
The DAG Plot:



Consider our first plot (above), and imagine that it is some causal "story" we wish to convey to the reader. Here C influences both O and D which both influence Y. If we care about the effect of D on Y then we ought be careful that we are not measuring any effect of O on Y as well. This is the backbone to more complex DAG plots, or dynamic acyclic graphs. As a few definitions with these going forward, a **back door path** is any possible influence that goes into our causal variable rather than coming *from* our causal variable to the dependent variable. Here C is a backdoor path, because Y could influence O (thus C) against the arrow, which in turn gives us false results if we merely measure D on Y. A **collider** is a node which has multiple variables running into it (in social science most variables will be colliders). Colliders hand-off information into these "hubs" which can greatly complicate our findings if not accounted for. Sometimes we may see broken lines or different shapes and subscripts to denote errors in these graphs (take THAT Rubin!) and how these are thought to exist between nodes. Note that DAG plots are made entirely from theory, and that qualitative and quantitative data alike can sculpt how we might model our causal phenomena of interest. Ultimately we care about a few things, (1)

identifying exogenous variation, (2) blocking backdoor paths, (3) sum up the front door paths. But how do we "block" these dangerous paths? Well there are many ways, the most common in experiments is to stratify the experiment by the node, but the **curse of dimensionality** tells us that as we do this more and more we will run out of respondents (countries, etc) to be able to afford all of the strata we may want. Thus, many methods such as matching, and design choices help us account for blocking difficulties.

Another great way to account for back-door-paths is to simply randomize, randomize everything in many different ways. By randomizing we shuffle who might get treated or not, with no interaction between them (SUTVA), and no selection into treatment. Although this will be mentioned in detail later. A final note is that sometimes researchers will put coefficients and effect sizes on the edges between nodes to show effects between and within variables of a study. Also when we see circles and dotted lines these are not easily observable variables that are not measurable, or observable.



1.2 Treatment (Causal) Effects, SUTVA, Confounding

In its simplest form, we care about the difference between the average treated effect and the average control effect. The difference between the two is the **Naive Differences in Means**, or the naively simple calculation of treatment effects. Why is it naive? Well, we haven't considered any assumptions surrounding the groups, and the fundamental problem of causal inference — the counterfactual. A counterfactual is the "what if" of being in the other group than the one assigned.

$$\hat{\mu}_{\text{Naive}} = E[Y|D=1] - E[Y|D=0] \tag{3}$$

Think for a second, given all of this, when might the $\hat{\mu}_{\text{Naive}} = \hat{\mu}_{\text{ate}}$? Or, when might the naive estimator be the same as our average treatment effect (ATE)? Remember that the reason why μ_{naive} is naive is because it never observes the counterfactual world.

Answer:

Note that the above equation has no decorations on Y, we are assuming that all respondents are not interacting with each other, even over time, (SUTVA) and that our design is absolutely complete (think of a complete DAG plot) such that we really are just getting $E[Y_1] - E[Y_0]$. So no error, and no SUTVA violation we get the *ate* to equal μ_{Naive}. Here SUTVA is doing a lot of work, we are assuming with SUTVA that there is no interference and treatment variation ($Y = Y_1$). This is a pretty monstrous assumption, countries, people, politicians, the media, *everyone* often communicates and shares what they have been "treated" with all the time! Beyond this, treatments often effect people very differently depending on affiliation, openness to feedback, or if you ate breakfast that

morning. Some methods like network analysis will embrace this fact instead of avoiding, although this method will not be covered here.

SUTVA, or the Stable Unit Treatment Value Assumption, follows two primary tenants:

1. That the treated and controlled units are not interacting between one another.
2. That the method of how someone gets treated does not vary between units, or is at least equivalent.

Some core assumptions are not only SUTVA (consistency) but also that no *confounding* occurs (also known as selection on observables, ignorability). This is when the people that were to benefit most from the treatment select into the treatment. No confounding formally is:

$$D \mid Y_1, Y_0 = P(D = 1 \mid Y_1, Y_0) \tag{4}$$

This language, by the way, is really annoying because: selection on observables = conditional independence = conditional ignorability = no confounding, conditional on covariates = blocking all back door paths. Open backdoor paths, is Pearlsian DAG-speak for having confounding. If we have more treated folk for each Y_0's than Y_1's then we have imbalance. If X is correlated with Y_1 and Y_0 for *any reason*, then even though we randomized, then Y_1 and Y_0 would be informative about X - And therefore Y_1 and Y_0 would be informative about our treatment D! Therefore we have a broken experiment.

Some other things to look out for is the plug-in estimator and conditional expectation functions, each of these are just examining average treatment effects around conditions (control variables). The *plug-in estimator* is when we condition on and control for the average treatment effect (ATE) summing up over our controlled X's.

$$E_X[E[Y|D = 1, X] - E[Y|D = 0, X]] \tag{5}$$

The conditional expectation function (CEF) is when we take the average treatment effect to see conditional causal effects.

$$E[Y|D] = E[Y_0] + \mu_{ate} D \tag{6}$$

We could think about taking into account all variables and possible conditions, this is the hope of Rubin's *saturated model*, which has it's roots in imputation for trying to find missing data (the missing data here in our case would be the counterfactuals). Saturated models do come with particular downsides however, mostly in the errors it produces, and our lack of knowing the exact causal pathway given that adding new variables can open up back-door-paths to our causal estimand.

1.3 Identification Through Randomization, Hypothesis Testing

Identification strategies are the attempt to find the missing data that we don't know due to the fundamental problem of causal inference. Two popular paths to do this are using controls and randomization.

There are different types of randomization to help us gain inference. Simple randomization is a probability of treatment per individual (D "treatment" being applied) in all possible cases; complete randomization is deciding what fraction of observations we need to fulfill for our total group and then curate our probability of treatment to this. Using randomization, we can hope to "block" on these observations (like gender, or X) which we cannot randomly assign.

$$D \mid Y_1, Y_0 | X \text{ or, } P(D = 1 \mid Y_1, Y_0; X) = p \tag{7}$$

Essentially, we could calculate the naive differences in means for each possible estimation of all of treatment and control groups: and that is called randomization inference (Rubin's causal model).

In hypothesis testing we could just take with and without treatment and see how our distributions converge (remember the t-test stuff)

$$\sqrt{n}(\hat{\mu} - \mu) \xrightarrow{d} \mathcal{N}(0, \sigma^2) \tag{8}$$

Cool, what's the p-values that we get from this? It is saying that conditional on the null hypothesis (i.e., conditional on $\mu_{ate} = 0$), what is the (frequentist) probability of observing this extreme an estimate of $\hat{\mu}$? This is typically in regards to a sharp null hypothesis of zero average effects, $Y_1 = Y_0$ always.

On hypothesis testing, in most cases we start off with testing against a *sharp null hypothesis of zero average effects*. That is, $\mu_{ate} = 0$. Note that this is probably the most boring, unrealistic, and uninteresting null hypothesis that we could ever test against. Does anything, in social science, really ever have a sharp zero effect? Probably not, but we use our handy friends, the z-test and t-test, to find the statistical significance of our finding compared to the zero effect. For small samples we often assume that, because of the large sample size used, the unknown population variance may be replaced by the sample variance. That is, instead of a z score we use a t score:

$$z = \frac{\bar{x} - \mu}{\sigma/\sqrt{n}} \tag{9}$$

one would use:

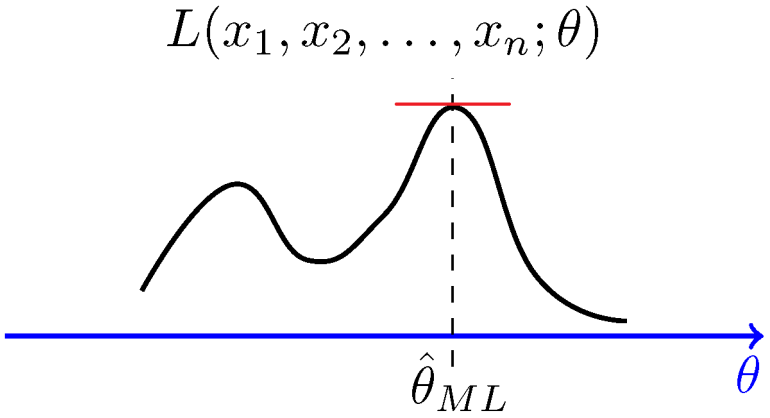
$$t = \frac{\bar{x} - \mu}{s_x/\sqrt{n}} \tag{10}$$

1.4 Frequentism and Bayesianism: What are we really doing here?

"Statistics" has its roots, interestingly, in social sciences as the origins of the word derives from "state-istics" or the study of the state. This methodology, uniquely bred for States to understand it's citizenry, statistics have become fascinatingly turned for the *people* to now understand their State. Following World War II this was more important than ever, a social science with deep philosophical, theoretical, and qualitative research began to explode with quantitative methodology in through the 1950's. These methodologies, deeply aided by fields like biology, computer science, and econometric, have both drawn us closer to understanding public and governmental behavior *and* has driven us farther from these concepts as methodology has been misused (e.g. p-hacking), misunderstood, and miscommunicated.

To begin to understand why we use statistics, general linear models, or machine learning, one must first understand a few basic underpinnings of frequentist theory versus Bayesian theory. Frequentism comes in social science largely from R.A. Fisher's idea of likelihood theory of inference. It argues that one true parameter (θ) exists in nature and all scientists are doing are getting multiple samples to guess the one-true-fixed θ. Our results are likelihoods that are close to measuring the one-true DGP but **always** with some uncertainty. In theory, we can map our estimated DGP $\hat{\theta}$ in two-dimensions pretty easily (think of a basic x–y coordinate grid). Yet, the social world is very complex, as we add more and more variables we begin to get to a point where we find ourselves in hyperspace, unable to map and visualize our surroundings (thus, machine learning helps here, but this will come up later). To help find interesting points when we do these regressions, we (in frequentist theory) seek the MLE or the Maximum Likelihood Estimation. Note this is purposefully not the Maximum *Probabilistic* Estimation, in likelihoodist theory these words mean very different things. The take-away here should be to be careful when speaking in likelihoods (a relative measure of uncertainty) versus probabilities (results only from randomized experiments) as these words have strict definitions in causal inference.

An MLE is interesting because it is the impasse between our estimated parameter $\hat{\theta}$ and our y dependent variable. This typically occurs at the "highest" or maximum point on whatever curve we have. Consider the image below, the red line indicates the plateau or maximum point, for the liklihood (L) of y (estimated by x_1, etc.) on our θ.



To find, mathematically, our MLE (which occurs under-the-hood in R) we compute the following:

1. We log the likelihood function for ease of computation.
2. We then take the derivative (also known as the score-function).
3. Then set this derivative equal to zero (to find our flat top part).
4. we then solve for $\hat{\theta}$.
5. We then make sure found the maximum and not some minimum peak (note we could have found the other peak in the above image). To do this we check the sign of the second derivative, if negative we know the curve is downward-facing. We finish by calculating the Fisher information (the second derivative) to find our confidence via variance and standard errors.

1.5 Running Assumptions

Typically we trade-off assumptions but can never completely eliminate them, only show or *prove* that we have fulfilled them – all to get closer to causation (causal inference). Some running assumptions are:

1. SUTVA
2. No confounding
3. Probability exists only between 0 and 1, but never zero nor 1.

If we attempt to "lean-into" one of these assumptions to be able to measure it, we may come across additional assumptions like "parallel paths" or various representation and error assumptions. Some of these will be discussed in the Regression section of this book.

1.6 Non-parametric Modeling

A few non-parametric models help us lean away from certain assumptions. This could be an entire class on it's own, but I have a few notes that I'll share here.

1.6.1 Multiple Comparisons

1.6.2 Marginal Structural Models (MSM)

If we are working with relational datasets, and do have downstream effects at t_1, t_2, t_n we cannot solve this problem by using fixed effects. Because we are assuming with fixed effects that our estimands do not change over time, this is actually a core assumption that is violated all the time. However, fear not, for marginal structural models can help us when fixed effects cannot.

Assuming time doesn't influence previous times (which may be an issue if we're studying some forecasting or strategic process that appears in our data). we can look at the outcome in the last time period and stack each time period on top of one another. This is very different from adding a bunch of dummy variables, it is stacking the time periods together; we can treat for the histories in this way and potential influence among them.[1]

So marginal structural models can work with different treatment histories (arms) than fixed effects. Let's fit a model for every "D," and just weight each of these down the road. We condition the history of past events, and if the model pukes at us then we drop the bad cases and redefine the subsample we are talking about, and how it's still interesting.

Note, we do not get out of the unobservables problem, we still need to set up this model with the same theory and careful design. *You cannot analyize your way out of a bad design.*

Some code to look into, (and notes):

```
> # GLM has an argument called "weights," don't trust it...
>
> library(survey) # this is better for a variety of reasons.
>      svydesign(ids = , weights = ~1, data)
>      svyglm(Y ~ D, unweighted_design, family = quasibinomial)
>      coef(The_Model)
```

These weighted coefficients can help us, but we can try to stabilize the weights we get, especially in cases of low probability events. For this we need non-dynamic probabilities. Assign these values with R's "mgcv" package, and function "bam" and some smoothing with `s()`. Get the t-specific weights, and there are ways to plot the improvements from doing this. Then, put the weights together (stabilized) so mean around 1, and multiply these easily with `prod(weights)`. Change out these weights in above, instead of `1` we can use `weights`:

```
> # after checking out:
> library(mgcv)
>      bam()
>      s()
>
> # Then we can edit things:
> library(survey) # this is better for a variety of reasons.
>      svydesign(ids = , weights = ~weights, data)
>      svyglm(Y ~ D, weighted_design, family = quasibinomial)
>      coef(The_Model)
>
>      # Might wan to check out package: cbmsm
```

[1] It may be a good idea to check out time series analysis here, just in case you problems can be solved with your data.

We can then get predicted probabilities from our marginal structural models. So essentially with marginal structural models:

1. Set up data so each outcome has multiple individuals and prior time periods.
2. Model treatment in each period without tie-varying covariates with them.
3. Assess balance (repeat 2 if necessary).
4. stabilize weights.
5. calculate weighted estimates of QOI's.
6. What is N anymore? Let's just always bootstrap the heck out of this: doing (2), (4), (5), over 1,000 times over.

1.7 Finding Causal Estimands:

1.7.1 Matching and Propensity Scores:

Matching is weighting each observation to all strata. Much of this is really the same thing, matching is pairing individuals and comparing causal estimands and weighting is forcefully multiplying a "weight" onto our variables to pair them to reality (say the Census). Subclassifcation is the same thing, in the long, as matching and weighting. If we don't have exact matches we can use a coarse matching approach where we break up a continuous variable into deciles. This is similar to K-Nearest Neighbor subclassification matching. And a propensity score is just the probability of treatment given some X.

$$e(X) = Pr(D = 1|X) \tag{11}$$

The idea is then to coarsen/subclassify or nearest-neighbor match on the **propensity score**. **Subclassification** is making the probabilities of treatment and control equivilant in observational studies. Doing this, we help create balance in our treated and controlled groups. This is similar to complete randomization in experimental trials. (What machine learning techniques can help us here?)

Some machine learning approaches that can help us here (see machine learning chapter for more) include regression trees and other classification approaches. These can help us identify groups to match on create more balance in our data.

1.7.2 Fixed Effects:

Fixed effects are dummy variables (are stratification) and are turning "on or off" a year, country, or some other theoretically interesting variable to "account" for the things that pour into a time or place. Using this we assume no carry over between years to previous or future years. Here are the assumptions with fixed effects:

1. no unobserved time-varying confounds exist
2. past outcomes do not directly affect current outcome
3. past outcomes do not directly affect current treatment
4. past treatments do not directly affect current outcome

One larger problem is that the linear fixed effect does not consistently estimate the ATE:

$$\beta_{LFE} \rightarrow \frac{E[V^i \mu^i_{naive}]}{E[V^i]} \neq \mu_{ate} \tag{12}$$

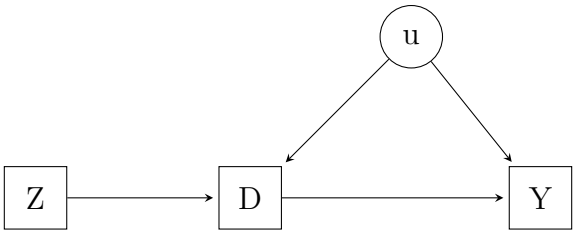
But marginal structural models allow for our times and events to influence one another (more on that later).

1.7.3 Differences in Differences

Differences in Differences is a way to estimate our causal estimand (ATT, ATE, etc) and map it when treatment rolls out. Say we have three time points 1, 2, and 3, where all are untreated at 1, some are treated at 2, and all are treated at 3. To get the estimand from differences in differences we first subtract (getting the difference) of the mean outcomes between times and 2 and time 1 for the treated and controlled then calculate the differences in those two groups between the two time groups. We could do the same thing for 3 and 2. An assumption for difference in differences designs is that things were to be on the same parallel paths, or that we need to prove to the reader (often through theory) that the treated and controlled group – in a world where no treatment occurred, actually would've been the same (parallel to one another). Further, that it is only our treatment group that caused a change between the control and treated group, not anything that would've happened anyways.

1.7.4 Instrumental Variables

Here imagine we are interested in the causal effect of D –¿ Y but our errors, or some unknown (u) covariate influences both our treatment and outcome. What we can do is add in some "garbage" variable (Z) into the mix, and see just how Z passes through to Y. It helps me think of "pipes" where we want to see how much [insert liquid here] passes through D to Y, neverminding covariate u.



1.8 Conclusion: Causal Inference

In the end, whether you agree or disagree that causality can truly ever be found in our social world, the efforts we take to ensure that theory is at the forefront of our models is never time wasted. Thinking through - seriously - the exact relationships between our variables and carefully crafting our models make them more convincing be it scientifically, in a business scenario, or in government work. It shows our hand, hides nothing, and clarifies our thinking. It encourages teamwork and collaboration around a shared thought and research agenda, and it helps in presentation by painting a clear picture of the work accomplished. Further chapters are not meant to be shortcuts to this chapter, but additions to it.

2 Introductory Statistics

While causal effects are certainly the cream-of-the-crop in terms of statistical findings; it's not necessary to examine our social world. In fact, in many cases it's not possible to get causal effects for human behavior. I cannot, for example, randomly assign a population to become black, female, or Hispanic to study the effects of discrimination. However, there is an intersection of causality and correlation: the likelihood. That is, how likely is X to happen, given a list of set assumptions?

There's an example involving famous statistician Ronald Fisher, someone we've heard of. Story has it, someone in his office one day claimed that she was a passionate tea lover, so passionate that she could tell from taste alone whether tea was added to milk or if milk was before the tea. Fisher, ever-fun at parties, questioned the lady's passion with statistical analysis. How would we set up a *causal* analysis of this however? Well, we don't have multiple universes of passionate tea-loving ladies to examine; yet, we can test the lady's abilities against random chance. Assuming random guessing would lend us a 50/50 chance of success if the lady were to defy those odds to such a great detail we could claim her successful. As the story goes, the lady could not correctly guess the difference against random chance after Fisher had her blindfolded and tested. Something deep inside me hopes this story is true.

This all begs the question, how do we statistically examine the degree to which someone "defies odds" or what can we call "statistically significant" success at something? Fisher, not leaving the world hanging, had other projects besides breaking old ladies hopes and dreams. In a study on manure on farming yields Fisher noted that "statistical significance" could be found at three separate thresholds which could be best communicated using associated p-values. These p-values are simply representations of the "probability of unusualness" found via transformation of our standard errors giving a set distribution. The thresholds are usually on the 10% tail (.10), 5% tail (.05), and the most unusual 1% tail observations (.01). If a set observation, or population distribution were to exist in these tail regions, passing these thresholds we could claim statistical significance. These thresholds were noted by Fisher to be "usual, and convenient" enough in the scientific enterprise to use them universally. This also means that the origins of the p-value are mired in farm manure.

In this section of the methods handbook I'll go over the fundamentals of t-tests, population distributions, z-scores, chi-squared testing, and other tests that all come back to the same notion of testing "unusualness" in the world. This may not be causal observations but can be powerful, these tests are deployed by cancer researchers to aerospace engineers - it has saved lives and it's perversion has conversely caused great harm. These examples and many more will be included along the way. Last note, there very well could be a section on our assumptions and tests for them, however I will just integrate these issues alongside the tests and direct specific attention to this in the introduction section on OLS regression.

2.1 T Testing

Directly, t-score is a ratio between the difference between two groups and the difference within the groups. The larger the t-score, the more difference there is between groups. The

smaller the t-score, the more similarity there is between groups. A t-score of 3 means that the groups are three times as different from each other as they are within each other. When you run a t-test, the bigger the t-value, the more likely it is that the results are repeatable. When we work with t-tests we are typically working with sample populations rather than whole population data (with a known σ error like we saw with z-testing). There are three ways we usually find a t-test:

1. An Independent Samples t-test compares the means for two groups.
2. A Paired sample t-test compares means from the same group at different times (say, one year apart).
3. A One sample t-test tests the mean of a single group against a known mean. Say, a sample statistic to a census statistic.

Each of these follow the same theory with slight variations to account for differences in known variance or not. Each of these test against a standardized null hypothesis of zero average effects ($H_0 : \mu_1 - \mu_2 = 0$) and, conversely, our alternative hypothesis can be that our sample is anything but zero ($H_A : \mu_1 - \mu_2 \neq 0$) or existing only in one tail ($H_A : \mu_1 - \mu_2 > 0$). However, our alternative hypothesis specification doesn't really change our test much - just our interpretation of the critical values and our multiplicative testing parameter, similar to the z test.

Formally, our t-test takes the following form when we are comparing our data to a known population mean:

$$t = \frac{\bar{x} - \mu_0}{\sqrt{s^2/n}} \sim t_{df=n-1}$$

Here μ_0 is our postulated value of the population where as $\bar{x}$ is the sample mean. Our test should, now, not follow a normal distribution but rather a student's t-distribution which is similar to a normal distribution but rather we take into account the degrees of freedom $(n-1)$. Why subtract one? Well this is because we "spend" one degree of freedom when finding our sample mean, with additional parameters we "use" more degrees of freedom in our estimation process. In the denominator we account for our sample variance and the quotient with of our sample size, all square rooted. To find s:

$$s = \sqrt{\frac{(X - \bar{x})^2 ... (X_n - \bar{x})^2}{n-1}}$$

Recall our degrees of freedom being calculated in s not in the actual t-test. Without speeding ahead too fast let's stop and consider an example before opening it up to two sample differences. Suppose we gave a small class an exam from a well-published textbook. The textbook company claims that, on average, students get an 85% on the exam. Is our class *significantly* better or worse than the national average?

```
> # Class scores:
> class1 <- c(60,70,80,78,98,86,78,82,93,99,86,86,79,0,77)
>
> # Finding s:
> s <- sqrt( sum(((class1 - mean(class1))^2)) / length(class1)-1)
> # Finding our t statistic:
> t <- (mean(class1) - 85) / (sqrt(s^2/length(class1)))
> t

## [1] -1.400315

> # p-value:
> pt(t, df = length(class1)-1) # using t-distribution! (pt)

## [1] 0.09159503

> # Testing this against R's function
> r_results <- t.test(class1, mu=85)
> r_results$statistic

##         t
## -1.35152
```

We see that according to Fisher's definition of statistical significance we pass the 10% threshold of "unusualness" but not the significance level of .05 or .01.

Suppose we were to compare exam scores for our class to a comparative school across a local river. Here we would need to use to two-sample t-test. Luckily, the formula doesn't change much except that we need a pooled sample standard deviation in the denominator:

$$t = \frac{\bar{x} - \bar{y}}{Sp\sqrt{1/m + 1/n}}$$

Where our Sp standardized deviation is:

$$Sp = \sqrt{\frac{(x_1 - \bar{x})^2 + ... + (x_m - \bar{x})^2 + (y_1 - \bar{y})^2 + ... + (y_n - \bar{y})^2}{m + n - 2}}$$

Note the pooled observation -2 because of the two sample mean estimations already used. This follows the same t-distribution (t_{m+n-2}). Is our class significantly better than the other schools?

```
> # Two samples:
> class1 <- c(60,70,80,78,98,86,78,82,93,99,86,86,79,0,77)
> class2 <- c(40,100,81,88,98,86,90,82,93,99,86,86,80)
>
>
> # Using Cohen 1988:
```

```
> sp <- sqrt( sd(class1)^2 + sd(class2)^2 / 2)
> # But wait:
> sd(class1) == sd(class2)

## [1] FALSE

> t <- (mean(class1) - mean(class2)) /
+      sqrt( sd(class1)^2/length(class1) + sd(class2)^2/length(class2))
> # p-value; x2 for both tails
> 2*pt(t, df = (length(class1) + length(class2) - 2) )

## [1] 0.259778

> r_results <- t.test(class1,class2)
> r_results$statistic

##         t
## -1.152065

> r_results$p.value

## [1] 0.2605557
```

Note that here we could not prove that our standard deviations are equal so we had to do a different standard deviation test where the denominator uses the standard deviations independently instead of pooling them. Our results are exactly what R's t.test function would have given us however it should be noted that there is some nuance here in selecting exactly how we account for standard deviations. For more information consider Welch's t-test versus choosing the smaller df as a conservative estimate against our Type 1 error.

Above I used a very sharp test to see if our standard deviations were equal or not, however we can add some nuance to this assumption and see if our standard deviations are significantly different from one another. Testing this assumption can be done with an F-test. Unsurprisingly, the "F" in F-test is in regards to Fisher who thought the F-test could be used to help researchers make better decisions in the variation of t-test they use.

$$F = \frac{s_x^2}{s_y^2} \sim F_{m-1,n-1,\alpha}$$

Using an F-test table we can find our critical value and test the null hypothesis that $s_x - s_y = 0$. That is, if we reject the null hypothesis using the F-test we would not use a pooled test and if we could not reject the null we would have to keep the standard deviations separated.

2.2 Z Testing

$$z = \frac{\bar{x} - \mu_0}{\sigma/\sqrt{n}}$$

This is distributed normal mean 0 standard deviation of 1.

$$z = \frac{\bar{x} - \bar{y} - E[\bar{x} - \bar{y}]}{SD(\bar{x} - \bar{y})}$$

Where $E[\bar{x} - \bar{y}]$ is just our null hypothesis statement, note our null hypothesis argues zero average (E) effects - so this usually just wipes out given the zero. So what we get is:

$$z = \frac{\bar{x} - \bar{y}}{\sqrt{\sigma_x^2/m + \sigma_y^2/n}}$$

We would take this result, compare it with a z-table depending on our α and either use our observed value to either reject or accept the null. These z-tables can be found online and basically act as the pass/fail for much of these scores.

If our standard deviations are equal then we can:

$$z = \frac{\bar{x} - \bar{y}}{\sigma\sqrt{1/m + 1/n}}$$

Finding confidence intervals for all of this are as simple as taking the upper and lower threshold desired (say 95 percent or 90 percent) and then reporting the upper/lower in a table. Some models in the regression segment will do this for variety.

I found a really good article online about confidence intervals if you are looking for more notes on this: `http://sphweb.bumc.bu.edu/otlt/MPH-Modules/BS/BS704_Confidence_Intervals/BS704_Confidence_Intervals_print.html`

2.3 Chi-Squared Testing

The chi-square test of independence is used to analyze the frequency table (i.e. contingency table) formed by two categorical variables. The chi-square test evaluates whether there is a significant association between the categories of the two variables.

Let's review an example of this in R. We'll pull R's native iris data and separate the sepal lengths into two categorical variables - big and small depending on if it's above or below the median.

```
> dat <- iris
>
> dat$size <- ifelse(dat$Sepal.Length < median(dat$Sepal.Length),
+   "small", "big"
+ )
>
> # here's our contingency table:
> table(dat$Species, dat$size)

##
##              big small
##   setosa       1    49
```

```
##   versicolor  29    21
##   virginica   47     3
```

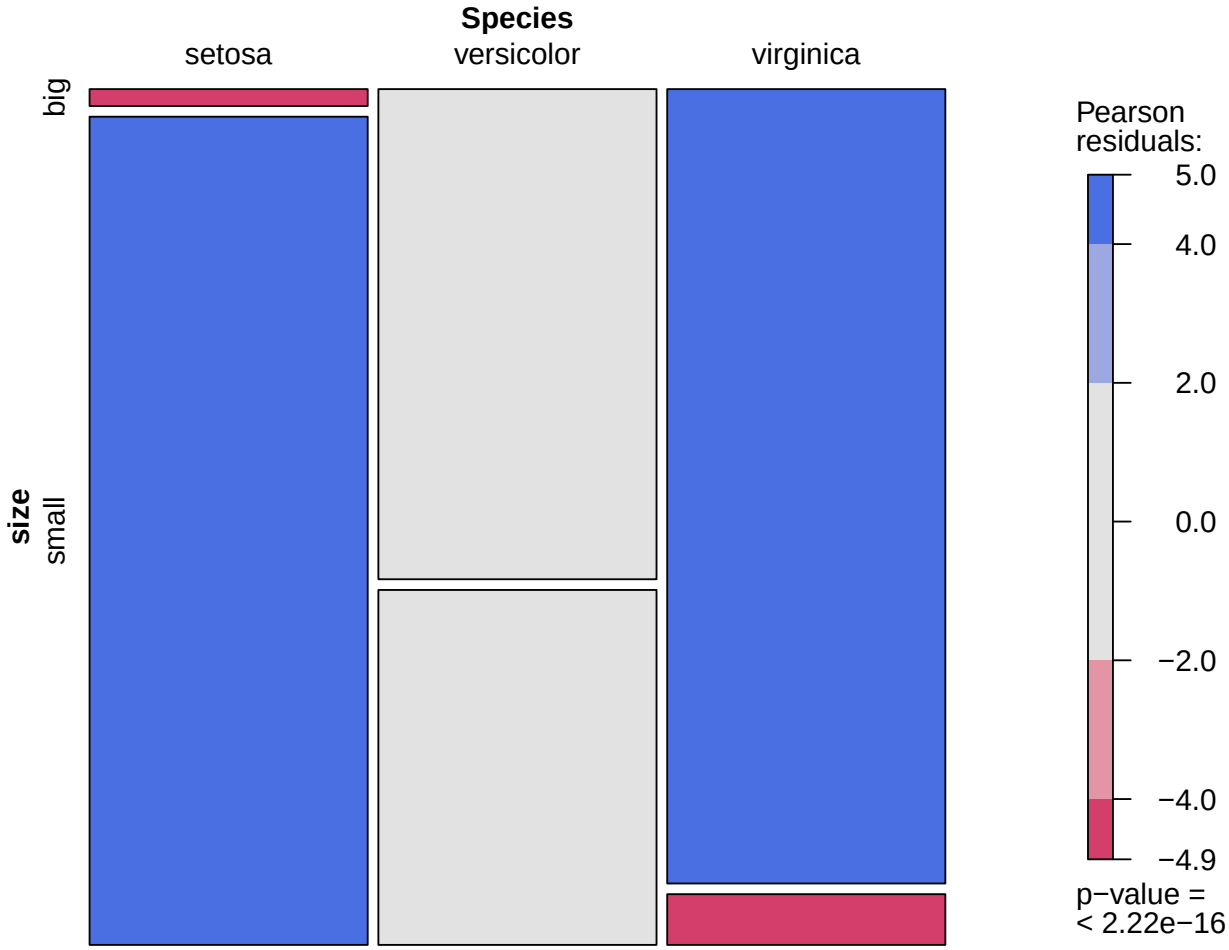
This is great, and below we will use R functions to calculate that chi-squared statistic:

```
> # calculate chi-squared (full results)
> test <- chisq.test(table(dat$Species, dat$size))
> test # full results

##
##  Pearson's Chi-squared test
##
## data:  table(dat$Species, dat$size)
## X-squared = 86.035, df = 2, p-value < 2.2e-16

> test$statistic # chi-squared result

## X-squared
##  86.03451

> test$p.value # p-value

## [1] 2.078944e-19
```

From the output and from test p.value argument we see that the p-value is less than the significance level of 5%. Like any other statistical test, if the p-value is less than the significance level, we can reject the null hypothesis.

Finally, we can combine the contingency table with the statistical results together in a concise visual:

```
> # This gives us the ability to make mosaic plots pretty easily.
> library(vcd)

## Warning:  package 'vcd' was built under R version 4.1.3
## Loading required package:  grid

> mosaic(~ Species + size,
+   direction = c("v", "h"),
+   data = dat,
+   shade = TRUE
+ )
```



This mosaic plot with colored cases shows where the observed frequencies deviates from the expected frequencies if the variables were independent. The red cases means that the observed frequencies are smaller than the expected frequencies, whereas the blue cases means that the observed frequencies are larger than the expected frequencies.

2.4 Permutation Testing

Much of this section on permutation testing comes straight from Thomas Leeper upon me meeting him at Ohio State in 2016. I highly recommend following him on Github, or checking out the relevant article for this section linked at the end of this section.

Thomas writes that an increasingly common statistical tool for constructing sampling

distributions is the permutation test (or sometimes called a randomization test). Like bootstrapping, a permutation test builds - rather than assumes - sampling distribution (called the "permutation distribution") by resampling the observed data. Specifically, we can "shuffle" or permute the observed data (e.g., by assigning different outcome values to each observation from among the set of actually observed outcomes). Unlike bootstrapping, we do this without replacement.

Permutation tests are particularly relevant in experimental studies, where we are often interested in the sharp null hypothesis of no difference between treatment groups. In these situations, the permutation test perfectly represents our process of inference because our null hypothesis is that the two treatment groups do not differ on the outcome (i.e., that the outcome is observed independently of treatment assignment). When we permute the outcome values during the test, we therefore see all of the possible alternative treatment assignments we could have had and where the mean-difference in our observed data falls relative to all of the differences we could have seen if the outcome was independent of treatment assignment. While a permutation test requires that we see all possible permutations of the data (which can become quite large), we can easily conduct "approximate permutation tests" by simply conducting a vary large number of resamples. That process should, in expectation, approximate the permutation distribution.

For example, if we have only n=20 units in our study, the number of permutations is:

```
> factorial(20)

## [1] 2.432902e+18
```

That number exceeds what we can reasonably compute. But we can randomly sample from that permutation distribution to obtain the approximate permutation distribution, simply by running a large number of resamples. Let's look at this as an example using some made up data:

```
> set.seed(1)
> n <- 100
> tr <- rbinom(100, 1, 0.5)
> y <- 1 + tr + rnorm(n, 0, 3)
```

The difference in means is, as we would expect (given we made it up), about 1:

```
> diff(by(y, tr, mean))

## [1] 1.341389
```

To obtain a single permutation of the data, we simply resample without replacement and calculate the difference again:

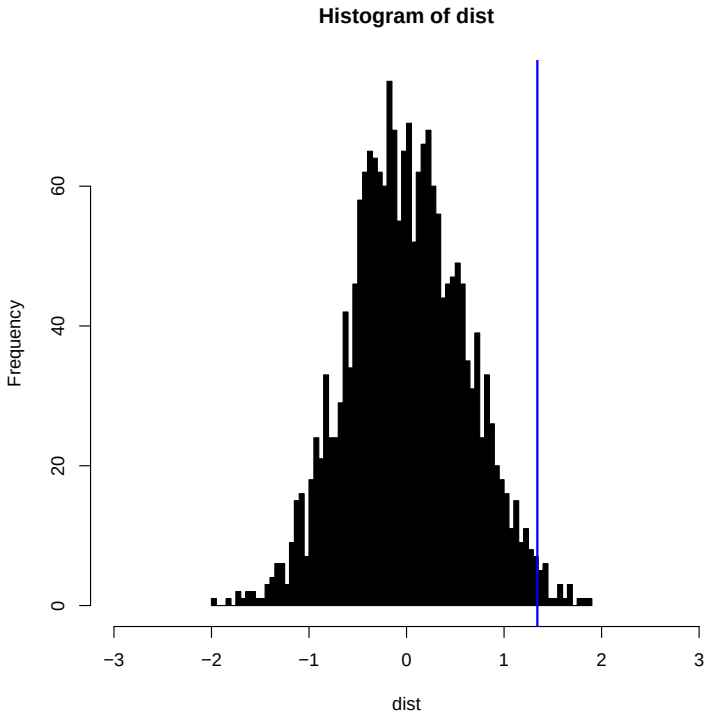
```
> s <- sample(tr, length(tr), FALSE)
> diff(by(y, s, mean))

## [1] 1.283456
```

Here we use the permuted treatment vector s instead of tr to calculate the difference and find a very small difference. If we repeat this process a large number of times, we can build our approximate permutation distribution (i.e., the sampling distribution for the mean-difference). We'll use replicate do repeat our permutation process. The result will be a vector of the differences from each permutation (i.e., our distribution):

```
> dist <- replicate(2000, diff(by(y, sample(tr, length(tr), FALSE), mean)))
```

We can look at our distribution using hist and draw a vertical line for our observed difference:

```
> hist(dist, xlim = c(-3, 3), col = "black", breaks = 100)
> abline(v = diff(by(y, tr, mean)), col = "blue", lwd = 2)
```



At face value, it seems that our null hypothesis can probably be rejected. Our observed mean-difference appears to be quite extreme in terms of the distribution of possible mean-differences observable were the outcome independent of treatment assignment. But we can use the distribution to obtain a p-value for our mean-difference by counting how many permuted mean-differences are larger than the one we observed in our

actual data. We can then divide this by the number of items in our permutation distribution (i.e., 2000 from our call to replicate, above):

```
> sum(dist > diff(by(y, tr, mean)))/2000  # one-tailed test

## [1] 0.012

> sum(abs(dist) > abs(diff(by(y, tr, mean))))/2000  # two-tailed test

## [1] 0.021
```

Using either the one-tailed test or the two-tailed test, our difference is unlikely to be due to chance variation observable in a world where the outcome is independent of treatment assignment.

There are many packages that assist in permutation testing and sampling, a few are used in future chapters. One Thomas Leeper uses in his article (quoted nearly verbatim in this chapter) is the library "coin". Visit his article linked below to review this package to see if it may help.

`https://github.com/leeper`
`https://thomasleeper.com/Rcourse/Tutorials/permutationtests.html`

3 Regression:

$$y = \beta_0 + \beta X + \epsilon \tag{13}$$

The General Linear Model (GLM) consists of a *stochastic and systematic* components. The stochastic component is the section in the equation which changes ("randomly") and has error our error term (ϵ), whereas the systematic component remains constant with data values (X) and is used to test our dependent variable (y) and give us interpretive values in the form of coefficients (β). To "generalize" this to other models (as we'll see in this document) we need a distribution and a link function, which will come up later.

It should be mentioned that this GLM formula has it's genesis in basic geometry's equation for a line $y = mx + b$ the root of this should be intuitive as we attempt to fit a line to our data and estimate it using maximum likelihood.

No matter what the model we're specifying throughout this document, be it logit, probit, Poisson, or survival models the linear predictor will always be the same:

$$\eta = X\beta \quad \text{or, } \eta_i = \beta_0 + \beta_1 x_{i1} ... \beta_q x_{iq} \tag{14}$$

One example of when we might use regression is when trying to determine the best quality of beer based upon mean judged ratings, these are the questions that matter the most. Beer ratings range from -1: "Fair", to 0: "Good", and 1: "Great." Consider our first model where we consider how expensive the beer is:

$$\text{Beer Rating} = \beta_0 + \beta \text{Beer Price} + \epsilon \tag{15}$$

Given the information from our data regressed in R: (This is shown on the following page)

$$\text{Beer Rating} = (-0.76) + (.26)\text{Beer Price} + \text{error} \tag{16}$$

We can solve this to find our beer rating with the substitution of a beer price, we remove the error term because we assume it's normally distributed and centered at zero (an assumption that will come up in the following section). Let's assume we are going out with friends and only have $2.00:

$$\text{Beer Rating} = (-0.76) + (.26) \cdot 2.00 = -.24 \tag{17}$$

This means, given that our average ("Good") beer rating is zero, that if we only had two dollars to spend on a beer we would get a slightly below average beer ($-.24$). This is assuming the price of beer and rating of beer is a linear relationship, this isn't an assumption we always have to make but one we will for now.

Showing this in R, let's load and set up our data and run the model that we used to get the numbers above:

```
> ## Initial pass: reading data, looking at variables.
> # Data available, email: MailKyleDavis@gmail.com
>
> library(foreign) # For reading data.
> beer_data <- read.dta('C:/Users/mailk/OneDrive/Documents/R/Graduate_Methods_Handbook/d
>
> library(car) # for ease of recoding variables

## Loading required package:  carData

> # Our Beer ratings could be coded more intuitively:
> beer_data$rating <- as.numeric(beer_data$rating)
> head(beer_data) # Check out the variables we have available to us:

##   rating         beer  origin avail price cost calories sodium alcohol
## 1      1 Anchor Steam     USA     1  7.19 1.20      154     17     4.7
## 2      1      Labatts  Canada     1  3.15 0.53      147     17     5.0
## 3      1       Molson  Canada     1  3.35 0.56      154     17     5.1
## 4      1    Budweiser     USA     0  2.59 0.43      144     15     4.7
## 5      1     Heineken Holland     0  4.59 0.77      152     11     5.0
## 6      1  Kronenbourg  France     1  4.39 0.73      170      7     5.2
##      class    light us quality good verygood
## 1 Super-pr NONLIGHT  1       3    1        1
## 2  Premium NONLIGHT  0       3    1        1
## 3  Premium NONLIGHT  0       3    1        1
## 4  Premium NONLIGHT  1       3    1        1
## 5 Super-pr NONLIGHT  0       3    1        1
## 6 Super-pr NONLIGHT  0       3    1        1

> nrow(beer_data) # For 35 beers:

## [1] 35

> # Let's recode our rating dependent variable to be more intuitive:
> # reassign values: where 3 is now 0, etc..
> beer_data$rating <- recode(beer_data$rating, "3=0; 2=1; 1=2")
> class(beer_data$rating) # numeric and coded intuitively, could be better...

## [1] "numeric"

> # From -1 to 1
> beer_data$rating <- recode(beer_data$rating, "0=-1")
> beer_data$rating <- recode(beer_data$rating, "1=0")
> beer_data$rating <- recode(beer_data$rating, "2=1")
>
> beer_data$rating # Setting median to zero will help our interpretation
```

```
##  [1]  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0
## [25]  0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

```
> set.seed(12345) # Setting our seed will fix the randomness for replication
>
> #    Let's do the basic linear model thing:
> ##!  lm(Y ~ X, data selection)
> mod1 <- lm(rating ~ price, data = beer_data)
> # just list our coefficients (to see everything use summary(mod1))
> mod1$coefficients

## (Intercept)       price
##  -0.7561396   0.2592005
```

```
> # Texreg Tables:
> library(texreg)
>
> texreg(l= list(mod1), stars = numeric(0),
+        custom.coef.names    = c("Intercept", "Beer Price"),
+        caption.above        = T, float.pos = "h!",
+        custom.note          = "Dependent variable: Beer Rating")
```

Table 2: Statistical models

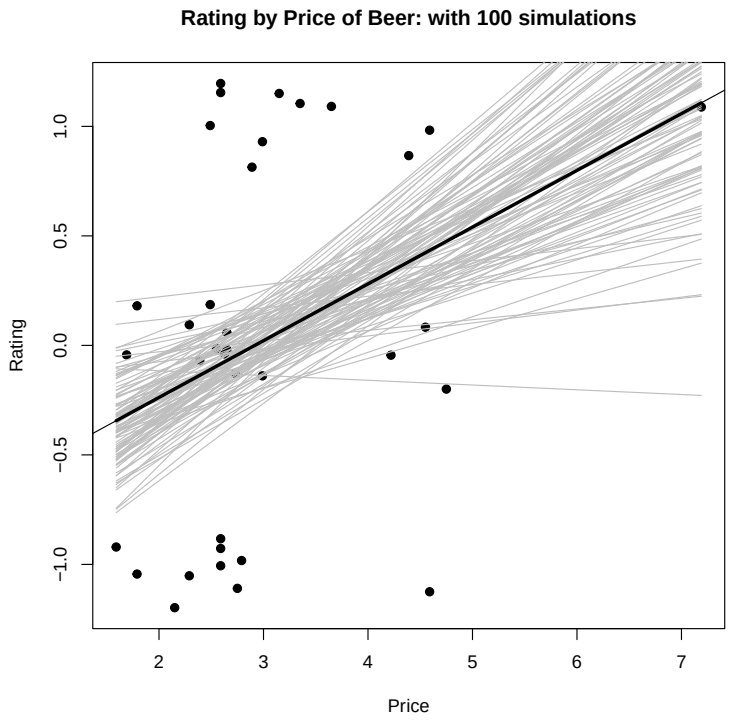
	Model 1
Intercept	−0.76
	(0.36)
Beer Price	0.26
	(0.11)
R^2	0.14
Adj. R^2	0.11
Num. obs.	35
RMSE	0.74

Dependent variable: Beer Rating

The above table shows the standard errors (in parentheses) being two standard errors away from our coefficients, therefore being significant at the 95% confidence level. We have 35 observations and our R^2 explains 14% of the variation in our dependent variable. The adjusted R^2 accounts for the amount of variables added into a model and is slightly more robust, prevent against "kitchen sink" models which add many variables to get a high R^2. It explains 11% of the variation.

Our intercept is when all else is held at zero (β_0) so the most average priced beer has a rating of $-.76$, which is overall pretty poor quality beer. Yet, if we spend one dollar more, our ranking jumps by .26 (a quarter rank) each time. This can be visually represented alongside simulations in a plot here:

```
> library(arm) # For sim()
> # Let's plot a simulation:
> plot(beer_data$price, jitter(beer_data$rating), pch=19,
+      xlab= "Price",
+      ylab= "Rating",
+      main= "Rating by Price of Beer: with 100 simulations")
> abline(mod1)  # Add our model line
> # Simulate 100 possible outcomes, take their lines, plot them on top
> m1sim <- sim(mod1)
> for (i in 1:100){
+   curve(coef(m1sim)[i,1] + coef(m1sim)[i,2]*x, add=T, col="grey")
+ }
> curve(mod1$coef[1] + mod1$coef[2]*x, add=T, lwd=3)
```



The grey lines here are simulations which display our confidence intervals around the black regression line for our model. They occupy a low amount of the variation in our dots so intuitively we can see how this would yield a low R^2 (our model only had .14). See below:

```
> mod2 <- lm(d$rating ~ d$price + d$alcohol)
>
> texreg(l= list(mod1, mod2), stars = numeric(0),
+        custom.coef.names = c("Intercept", "Beer Price", "Alcohol Content"),
+        caption.above = T, float.pos = "h!",
+        custom.note = "Dependent variable: Beer Rating")
```

Table 3: Statistical models

	Model 1	Model 2
Intercept	−0.76	−2.89
	(0.36)	(0.91)
Beer Price	0.26	0.21
	(0.11)	(0.11)
Alcohol Content		0.50
		(0.20)
R^2	0.14	0.28
Adj. R^2	0.11	0.23
Num. obs.	35	35
RMSE	0.74	0.69

Dependent variable: Beer Rating

We could consider a model where alcohol content is additionally important, Model 2. Here our R^2 increases to .28; and our variables remain within two deviations of the error, with the slight exception of beer price. Our intercept loses it's interpretation, we are now looking at where alcohol content is at zero. Our beer rank jumps by .50 every one increase in alcohol content. Both of these effects are controlling for the other.

3.1 Modeling Assumptions, Diagnostics

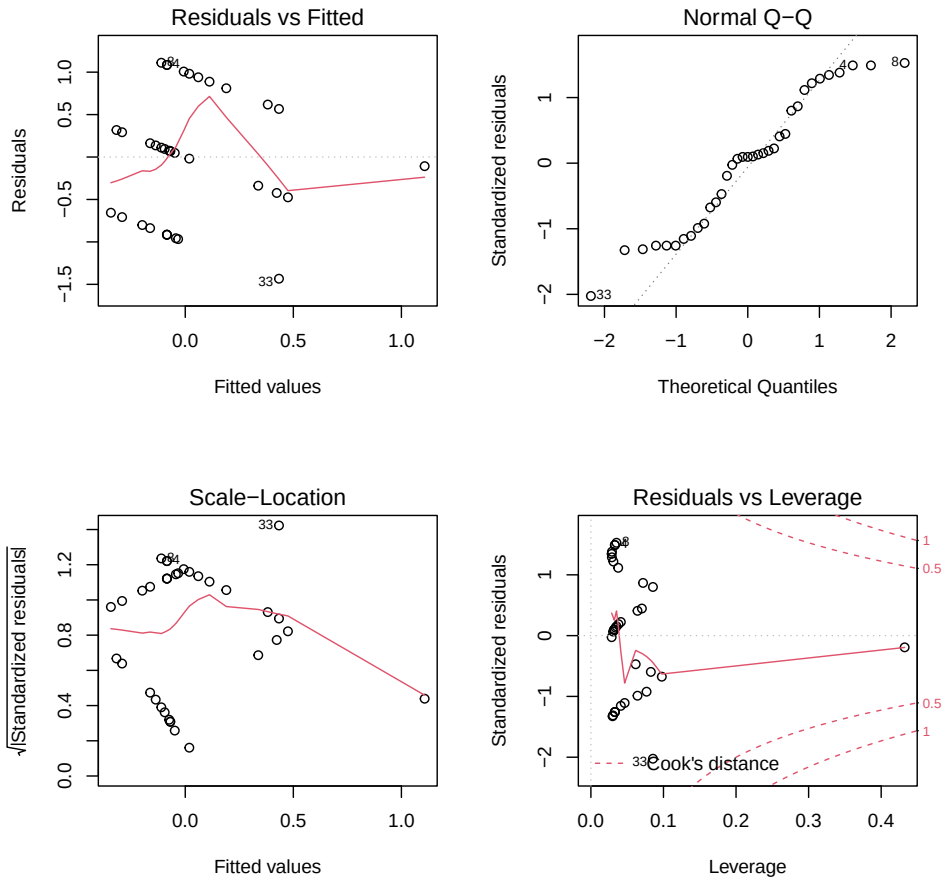
In social science empirical research there is no dearth of modeling errors. The following diagnostics and error-checking may not seem worthwhile because this work rarely makes it into the actual publication; but rather into a supplemental appendix. This work too helps a lot to ease skeptical readers about the claims we make (helping us get published).

$$\epsilon_i \sim \mathcal{N}(0, \sigma^2) \tag{18}$$

Errors are Normally Distributed

We assume that the errors are normally distributed, to check for this histograms can give us a first pass, but one can also map the residuals and (hopefully) see their "random" distribution with no foreseeable trends. the `plot(model)` command in R can give a few of these plots in a row pretty easily too. Consider our beer rankings model from earlier:

```
> par(mfrow=c(2,2)) # plot multiple alongside each other:
> plot(mod1)        # plot our quick model checks
```



```
> par(mfrow=c(1,1)) # Reset plotting option
```

Bias in Error

Next, we assume that there is no bias in our errors, meaning that our expectation of our errors in zero: $\mathrm{E}(\epsilon_i) = 0$. It is rare to get exactly zero, but we should report exactly what our errors are and be sure to mention any outliers, leverage points, or otherwise influential points to the reader so they are not mislead. We can handle these situations in various ways:

Never outright delete data points. Theoretically, the social world is vastly complex and rarities can (and do) exist in nature. If we have an influential point, where our data point(s) are not necessarily malicious or supremely rare, but rather pull our analysis in one way or another we can transform all of the data points (log or rescale the variable) to decrease the influence of one point. Observing points outside of "Cook's Distance" (in plot R command) can be a good way to observe influential points or outliers.

Uniform Variance

Our σ^2 is not subscripted, meaning we expect a uniform variance in our error term, where there is no increasing or decreasing in our errors. homoscedasticity here is a good thing, we can detect this by plotting our residuals and seeing random variance of our points across the mid-point. We do not want to see heteroscedasticity or where errors funnel outward or have a certain trend in themselves.

Autocorrelation, iid

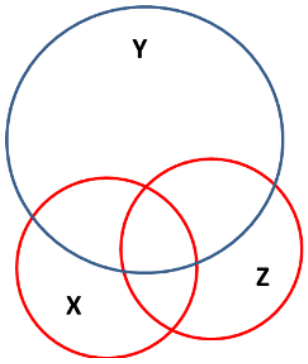
We do not want our observations to correlate with one-another, meaning our individuals or observations are not influencing another individual respondent's results. Each data point should be independent draws from one another.

Measurement Error (X)

We need to make sure the phenomenon we wish to measure is actually being measured by our data. A question such as "What do you think about healthcare" can be responded by a battery of cognitive processes; but a question like "To what degree of anger do you feel about the current healthcare policy?" is far more precise to the theoretical process we care to study. Beyond theory, the scale of our variable and our operationalization can be changed and shifted to answer different questions. In Section 3 that follows, for example, we will dichotomize a variable to observe a different question about how we think about our beer rankings.

Omitted Variable Bias (OVB)

OVB occurs when both a a variable excluded from analysis (Z) both theoretically overlaps some of the variance in our dependent variable (Y) and another included variable (X). If Z only overlapped with Y excluding it would just lower our R^2 and we would lose some of our potential ability to explain our DV. If Z only overlapped our X variable then we need only acknowledge Z in theory. It is only when Z overlaps both that we run into problems, for both we lose explanation of our DV and we over-inflate the significance of X's influence on Y. This can (and always should) be corrected by building a model slowly, one variable at a time, alongside well thought theory to ensure accuracy and purpose in our problem solving.



Parametric Linearity

If we run a standard linear regression we should be sure that our theory informs the decision that this phenomenon is linear, but also check that our relationship between our variables is linear in nature too. Other modifications of the GLM framework can include non-linear distributions. Linearity in our variables can be aided via transformation, with a drawback being (potentially) in interpretation.

Mathematical Concerns

We should ensure that X and Y have enough variance to provide enough change for analysis. Also, we should ensure (as often as we can) that our number of observations (N) is larger than our parameters (K). While having too high of an N will be more likely to report significant results in nearly any scattering of points, too low of an N can either show sharply significant results or no results at all when the *truth* can be the opposite. Typically having an N of 2,000 has been an accepted threshold.

In Structure

We also assume that our model is fully specified, and that our perameters (β) are "fixed" where our variables don't have a change-time or some external influence at a point in the data collection.

Other Notes on Stochastic Robustness

Note that much of the explainability and diagnostics of our model can be partially examined (at least as a first pass) through our R^2. Our R^2 is the amount of variance captured in Y from our model. More formally:

$$R^2 = 1 - \frac{\text{RSS}}{\text{TSS}} = 1 - \frac{\Sigma \epsilon_i^2}{\Sigma \left(Y_i - \bar{Y}\right)^2} \tag{19}$$

This shows our amount of variance explained by our model, over what's unexplained. This quotient is subtracted by one, we get a pretty good interpretation from .00 to 1 where we explain nothing to explaining everything. Other forms significance tests can penalize models for including many variables both linearly (AIC) or nonlinearly (BIC) these will be introduced later alongside logistic regression.

3.1.1 Missing Data

Often is the case that we have missing data and still models are ran as usual. Yet, missing data very often occurs for a reason, be it a question offends a respondent, a respondent doesn't know how to answer, or a unit of observation intentionally does not respond because the response may not be acceptable to the researcher. That is, the missing data is very often not missing completely at random (MCAR). Rather, data can be missing with surrounding answered variables that we can use to predict the missing response (imputation). If we have no variables to attempt to predict the missing data (NI) then there is nothing we can do. With MCAR data we would go on as normal, or impute because it wouldn't really hurt anything.

Assuming we can use multiple imputation, here we want to run 5 to 10 models of the predicted missing response using any variables we have (here a "kitchen sink" approach to missing data is a good thing because we want to cover the highest amount of variance). With these 5-10 theoretical data sets that could exist where the respondent answered the otherwise missing question, we can regress each of these hypothetical worlds and then take the mean of all of them to have the most likely responses included in our final result.

Two R packages that make this very easy is both "mice" and "Zelig." The functions to run multiple imputation from these packages can usually only take one or two lines of code and will print each individual regression and the mean regression. See the "mice" help file example example below:

```
> library(mice)
> ## Refering to the mice help file on imputation on medical data:
> #  Note that all head() is just showing the top row results.
>
> # do default multiple imputation on a numeric matrix
> imp <- mice(nhanes)

##
##  iter imp variable
##   1   1  bmi  hyp  chl
##   1   2  bmi  hyp  chl
##   1   3  bmi  hyp  chl
##   1   4  bmi  hyp  chl
##   1   5  bmi  hyp  chl
##   2   1  bmi  hyp  chl
##   2   2  bmi  hyp  chl
```

```
##   2   3  bmi  hyp  chl
##   2   4  bmi  hyp  chl
##   2   5  bmi  hyp  chl
##   3   1  bmi  hyp  chl
##   3   2  bmi  hyp  chl
##   3   3  bmi  hyp  chl
##   3   4  bmi  hyp  chl
##   3   5  bmi  hyp  chl
##   4   1  bmi  hyp  chl
##   4   2  bmi  hyp  chl
##   4   3  bmi  hyp  chl
##   4   4  bmi  hyp  chl
##   4   5  bmi  hyp  chl
##   5   1  bmi  hyp  chl
##   5   2  bmi  hyp  chl
##   5   3  bmi  hyp  chl
##   5   4  bmi  hyp  chl
##   5   5  bmi  hyp  chl

> head(imp)  # Checking, lots of useful information in here

## $data
##    age  bmi hyp chl
## 1    1   NA  NA  NA
## 2    2 22.7   1 187
## 3    1   NA   1 187
## 4    3   NA  NA  NA
## 5    1 20.4   1 113
## 6    3   NA  NA 184
## 7    1 22.5   1 118
## 8    1 30.1   1 187
## 9    2 22.0   1 238
## 10   2   NA  NA  NA
## 11   1   NA  NA  NA
## 12   2   NA  NA  NA
## 13   3 21.7   1 206
## 14   2 28.7   2 204
## 15   1 29.6   1  NA
## 16   1   NA  NA  NA
## 17   3 27.2   2 284
## 18   2 26.3   2 199
## 19   1 35.3   1 218
## 20   3 25.5   2  NA
## 21   1   NA  NA  NA
## 22   1 33.2   1 229
```

```
## 23   1 27.5   1 131
## 24   3 24.9   1  NA
## 25   2 27.4   1 186
##
## $imp
## $imp$age
## [1] 1 2 3 4 5
## <0 rows> (or 0-length row.names)
##
## $imp$bmi
##       1    2    3    4    5
## 1  28.7 27.4 30.1 27.2 30.1
## 3  30.1 30.1 35.3 30.1 30.1
## 4  21.7 24.9 22.5 21.7 24.9
## 6  25.5 27.4 21.7 20.4 20.4
## 10 22.5 26.3 27.5 29.6 24.9
## 11 30.1 30.1 35.3 22.0 22.7
## 12 22.5 27.4 27.4 27.4 22.7
## 16 28.7 27.4 20.4 27.2 30.1
## 21 27.2 22.0 20.4 22.5 35.3
##
## $imp$hyp
##    1 2 3 4 5
## 1  1 1 1 1 1
## 4  2 2 2 2 1
## 6  1 1 2 2 1
## 10 2 2 1 2 1
## 11 1 1 1 1 1
## 12 2 1 2 1 1
## 16 1 1 1 1 1
## 21 1 1 1 1 1
##
## $imp$chl
##      1   2   3   4   5
## 1  186 187 229 187 187
## 4  187 206 206 186 206
## 10 187 229 218 206 187
## 11 218 187 187 118 131
## 12 113 229 218 206 187
## 15 218 187 229 187 187
## 16 229 187 118 187 187
## 20 187 206 218 204 218
## 21 187 113 238 118 229
## 24 187 218 206 204 218
```

```
##
##
## $m
## [1] 5
##
## $where
##       age   bmi   hyp   chl
## 1   FALSE  TRUE  TRUE  TRUE
## 2   FALSE FALSE FALSE FALSE
## 3   FALSE  TRUE FALSE FALSE
## 4   FALSE  TRUE  TRUE  TRUE
## 5   FALSE FALSE FALSE FALSE
## 6   FALSE  TRUE  TRUE FALSE
## 7   FALSE FALSE FALSE FALSE
## 8   FALSE FALSE FALSE FALSE
## 9   FALSE FALSE FALSE FALSE
## 10 FALSE  TRUE  TRUE  TRUE
## 11 FALSE  TRUE  TRUE  TRUE
## 12 FALSE  TRUE  TRUE  TRUE
## 13 FALSE FALSE FALSE FALSE
## 14 FALSE FALSE FALSE FALSE
## 15 FALSE FALSE FALSE  TRUE
## 16 FALSE  TRUE  TRUE  TRUE
## 17 FALSE FALSE FALSE FALSE
## 18 FALSE FALSE FALSE FALSE
## 19 FALSE FALSE FALSE FALSE
## 20 FALSE FALSE FALSE  TRUE
## 21 FALSE  TRUE  TRUE  TRUE
## 22 FALSE FALSE FALSE FALSE
## 23 FALSE FALSE FALSE FALSE
## 24 FALSE FALSE FALSE  TRUE
## 25 FALSE FALSE FALSE FALSE
##
## $blocks
## $blocks$age
## [1] "age"
##
## $blocks$bmi
## [1] "bmi"
##
## $blocks$hyp
## [1] "hyp"
##
## $blocks$chl
```

```
## [1] "chl"
##
## attr(,"calltype")
##    age    bmi    hyp    chl
## "type" "type" "type" "type"
##
## $call
## mice(data = nhanes)
```

```
> # Our data: note the NA missing values
> head(nhanes$bmi)

## [1]   NA 22.7   NA   NA 20.4   NA

> # list the actual imputations for BMI, with predicted imputed values
> head(imp$imp$bmi)

##       1    2    3    4    5
## 1  28.7 27.4 30.1 27.2 30.1
## 3  30.1 30.1 35.3 30.1 30.1
## 4  21.7 24.9 22.5 21.7 24.9
## 6  25.5 27.4 21.7 20.4 20.4
## 10 22.5 26.3 27.5 29.6 24.9
## 11 30.1 30.1 35.3 22.0 22.7

> # first completed data matrix, can save this for analysis
> head(complete(imp))

##   age  bmi hyp chl
## 1   1 28.7   1 186
## 2   2 22.7   1 187
## 3   1 30.1   1 187
## 4   3 21.7   2 187
## 5   1 20.4   1 113
## 6   3 25.5   1 184

> ## imputation on mixed data with a different method per column:
> # mice(nhanes2, meth=c('sample','pmm','logreg','norm'))
> ## This will give you all information per column like before.
```

3.1.2 Model Specificiation, Distributional Functions

Some important continuous distributions:

Name	**Density**	**Parameters**	**Mean**	**Variance**
Uniform	$\frac{1}{b-a}$ for $a < x < b$	a,b	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$
Triangular	$\frac{1}{a}\left(1 - \frac{\lvert x-b\rvert}{a}\right)$, for $\lvert x-b\rvert < a$	a,b	b	$\frac{a^2}{6}$
Laplace (Exponential)	ke^{-kx}, for $x > 0$	k	$\frac{1}{k}$	$\frac{1}{k^2}$
Normal (Gaussian)	$\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$	μ, σ^2	μ	σ^2
Chi-Square	$\frac{x^{r/2-1}e^{-x/2}}{(v/2-1)!2^{v/2}}$	v	v	$2v$
Cauchy	$\frac{1/\pi}{1+(x-m)^2}$	m	(none)	(none)

In the following sections each of these should be illuminated as we use them in regression and think about how these relate to the natural world.

3.2 Logit and Probit Modeling:

Our dependent variable can take two values in logistic and probit regression; giving us a distribution such that:

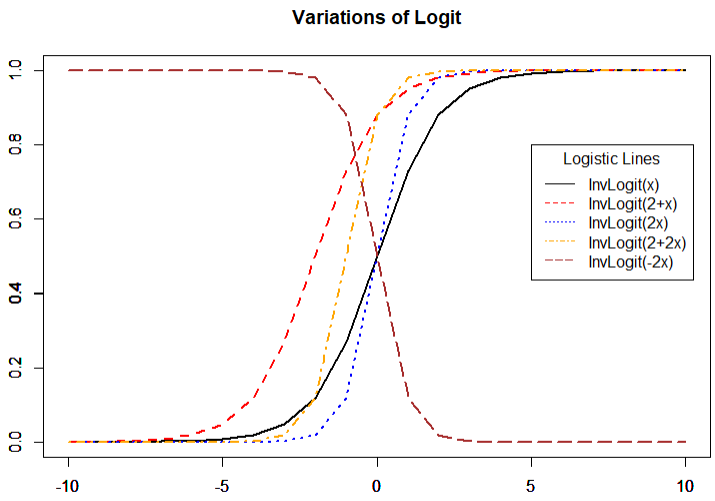
$$\begin{cases} 1 = \text{for event occuring} \\ 0 = \text{for event not occuring} \end{cases}$$

This can be used in a GLM framework where previously we had any continuous dependent variable available, we can have the dichotomous variable because of the necessary "link function" for our distribution. A link function is useful because it can be swapped out to fit any number of models we will see here in this document. This is absolutely important in GLM theory because it builds a connection between the mean of the data generating process to our linear predictors that otherwise would not pass our many assumptions if we were to just use standard OLS. Our two link functions for the logit and probit are as follows:

$$\text{logit: } \eta = p\left(\frac{p}{1-p}\right) \tag{20}$$

$$\text{probit: } \eta = \Phi^{-1}(p) \text{ where } \Phi^{-1} \text{ is the inverse normal distribution CDF} \tag{21}$$

Although probit and logit analyses are very similar in results, they differ in the distributions they allow. Probit regression works for a inverse normal distribution whereas logistic regression works for the logistic distribution (CDF's below). These are for the different theoretical DGP's, but the logit model is more likely to converge (data overlaps and varies enough) so we'll use that more often for accurate results, and often in social sciences we see the logit regression more often so given its canonical nature it is it more helpful at conveying important findings without confusion to the reader.



Below is an example of a model logit and probit regressions, reconsidering our beer data as the only real ranking of beer is "Very Good" or "Fair"; after all a "Good" ranking may not make sense to be any different from "Fair." Rather, these two rankings are combined to make a zero-one variable ("very good" versus "good or fair").

```
> logmod1 <- glm(beer_data$verygood ~ beer_data$price,
+                family = binomial(link=logit))
> ## summary(logmod1)  # Check your output
>
> promod1 <- glm(beer_data$verygood ~ beer_data$price,
+                family = binomial(link=probit))
> ## summary(posmod1)
>
> # Plot:   Using library(texreg)
> texreg(l= list(logmod1, promod1), stars = numeric(0),
+        custom.model.names = c("Logit Model", "Probit Model"),
+        custom.coef.names = c("Intercept", "Price"),
+        caption.above = T, float.pos = "h!",
+        custom.note = "Dependent variable: Very Good Beer (1)" )
```

Table 5: Statistical models

	Logit Model	Probit Model
Intercept	−3.12	−1.96
	(1.29)	(0.75)
Price	0.75	0.48
	(0.40)	(0.23)
AIC	42.96	42.79
BIC	46.07	45.90
Log Likelihood	-19.48	-19.39
Deviance	38.96	38.79
Num. obs.	35	35

Dependent variable: Very Good Beer (1)

From the table we see things to be very similar between the two models, the error reports (AIC, BIC, Log Likelihood, Deviance) are all similar, our price coefficient is still in the same sign, the logit model barely avoids significance whereas the probit model barely makes significance standards. Both coefficients are in the same direction however. Our intercepts vary between each other, but both are negative and significant.

Reading dichotomous outputs like these can be difficult, for the logit model a coeficient is the **changes in logged odds from moving from zero to a one**. Therefore, our price of .75 is the logged odds change from moving from a zero (a "Fair or Good" beer)

to a "Very Good" beer (1). But we often don't think in terms of logged odds, therefore it is always helpful to calculate the multiplicative effect of each variable.

We can try to understand more about what makes "Very Good" beer by building a model one block at a time until we think we have a complete model for the data we have - guided by our theory about very good beer in this case.

```
> # Another texreg example: using library(texreg)
> texreg(l= list(logmod2, promod2), stars = numeric(0),
+        bold = T, ci.force = T,   # Note some of these changes here
+        custom.model.names = c("Logit Model", "Probit Model"),
+        custom.coef.names = c("Intercept", "Price", "Availability",
+                              "Alcohol Content", "Caloric Content"),
+        caption.above = T, float.pos = "h!",
+        custom.note = "Dependent variable: Very Good Beer (1)" )
```

Table 6: Statistical models

	Logit Model	Probit Model
Intercept	**−23.43**	**−14.03**
	[−43.03; −3.83]	[−24.99; −3.07]
Price	1.11	**0.67**
	[−0.10; 2.33]	[0.00; 1.33]
Availability	**−3.53**	**−2.08**
	[−6.35; −0.71]	[−3.63; −0.54]
Alcohol Content	3.33	1.94
	[−3.01; 9.67]	[−1.59; 5.47]
Caloric Content	0.04	0.02
	[−0.10; 0.17]	[−0.05; 0.10]
AIC	32.60	32.20
BIC	40.37	39.97
Log Likelihood	-11.30	-11.10
Deviance	22.60	22.20
Num. obs.	35	35

Dependent variable: Very Good Beer (1)

The more specified table above includes the availability of the beer (is it often seen and promoted or more of an unknown beer to most bars), the alcohol, and caloric content; both of these are (at least chemically) to bring some form of pleasure and should be controlled. To switch up the data representation a little, confidence intervals are now shown and significance is indicated in bold.[2]

[2]Texreg package customization, p.24 there is an example.

Between the logit and probit models we still see a few differences, the intercept is again very different, likely because the difference in these models is the very distribution and link function they take which will shift the intercept; yet both are negative and significant. The price of the beer is insignificant for both models; yet, availability is significant for both models and in the negative direction (where 0 was a more frequent, popular beer). Alcohol and calorie levels are not significant for either model in predicting very good beers.[3] Finally, the model fitting reports are all very similar between the two models.

Going beyond sign and significance testing, let's first revisit our link function, if we take the inverse of the link function we can generate the "mean function" where instead of wrapping our link function around our linear predictor we can instead transform our DGP output itself. This can make more intuitive sense, and is again needed because then our values for η will be from the zero to one interval.

$$\text{logit}^{-1}(\eta_i) = \frac{e^{\eta_i}}{1 + e^{\eta_i}} \text{ to or from: } \eta_i = p\left(\frac{p}{1-p}\right) \tag{22}$$

Note from earlier that now in the logistic regression we are not dealing with a mere linear "one unit in x causes an increase in y" explanation. Rather, the logit regression is curved, and where we are at on the curve will determine how we read the regression coefficients. Generating predictive probabilities are always helpful, but require selecting a point of analysis per variable; this usually is set to the mean.

Consider our logit model from Table 4:

$$\text{logit}^{-1}(-23.43 + 1.11(\text{Price Value}) + \ldots + 0.04(\text{Caloric Content})) \tag{23}$$

Here we could enter in mean numbers for each variable, and change the value we add per variable for interesting predicted probabilities in various scenarios that may be interesting to us. The number received is the probability of y happening given the parameters x given the data. This is both intuitive and interesting for readers.[4]

The "divide by four" rule can be applied instead, were the beta coefficient is divided by four to yield the predictive probability *at it's maximum in the middle of the curve.* At this point, directly in the middle (intercept = 0), the derivative of our function becomes equal to $\beta/4$ allowing for an additional easy point estimate. Keep in mind this is best used for coefficients that are near the midpoint and not those on the tails because the logit distribution is much different at the tails than in the mid point. For example, in Table 4 our logit model showed the beer's availability had a large effect on the beer's rating.

---

[3]Interestingly caloric count is a low coefficient, likely because much of the variation is taken up by the alcohol content - which largely contributes to the calorie content in beer in the first place. An interaction between the two variables could be done, but this only inflates the alcohol content coefficient and all three variables become even further away from passing the significance threshold.

[4]Note this can also be done at the median of the data (intercept near zero), or at the middle category more generally. This is a case-by-case decision on whatever is most interesting given the question and data one is analyzing.

Taking it's availability $-3.53/4$ gives us a predictive probability of -60.5% That is, although this coefficient is away from the mean of the distribution, in example going from a widely available beer to a more sparse beer is likely to decrease the beer's rating (to the "Good/Fair" category) to a (approximate) 60.5% probability.

Since it is usually better to represent these data in graphic form, consider a probability plot of our logistic curve for price now by the probability of receiving a very good beer:[5]

```
> library(arm) # for inverse logit function
>              # (It may actually be in the MASS package)
>
> # Recall our logit model:
> logmod1 <- glm(verygood ~ price, data = beer_data,
+                family = binomial(link=logit))
>
> # The inverse logit gets us our predicted probabilities:
> #   All we're doing here is using some base-R plotting operations
> #   with creating our predicted probabilities per our coefficents
> curve(invlogit(logmod1$coef[1] + logmod1$coef[2]*x), 1, 5, ylim=c(-.01,.9),
+       xlim=c(0,7.5), xaxt="n", xaxs="i", mgp=c(2,.5,0),
+       ylab="Pr(Very Good Beer)", xlab="Price",
+       main="Probability of Very Good Beer, by Price", lwd=4)
> curve(invlogit(logmod1$coef[1] + logmod1$coef[2]*x), -2, 8, lwd=4.5, add = T)
> axis(1, 1:7, mgp=c(2,.5,0))
> mtext ("(two dollars)", 1, 1.5, at=2, adj=.5) # just adding text to x axis
> mtext ("(six dollars)", 1, 1.5, at=6, adj=.5)
> points(jitter(beer_data$price, 1.5), jitter(beer_data$verygood, 0.1),
+        pch=20, cex=.9) # Jittered points so we can see them
```

[5]The end section of this document, in quick overview, includes code for creating this.

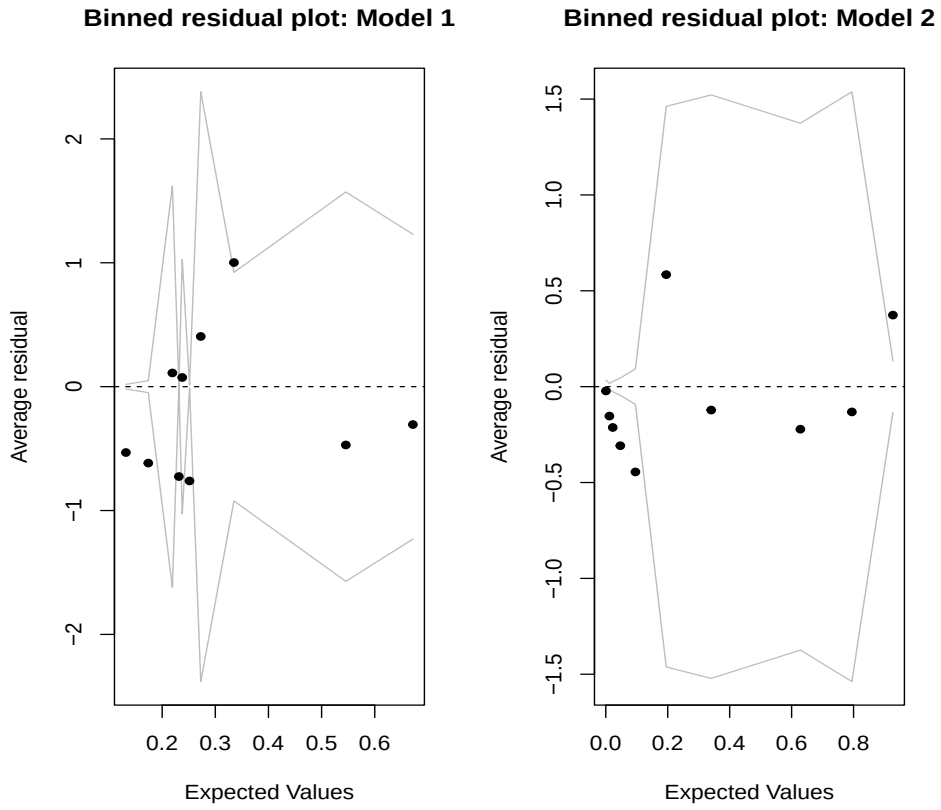
We can run some diagnostics on this similar to linear regression in that we're using residuals which are calculated similarly but for logit regression we have to "bin" these. I've done so for the initial model (only including price) and the full model (including calories and alcoholic content) below.[6]

Here, ideally, we would want to see "random noise" around the dotted zero mark more average residuals. Additionally, we want about 95% of the data to be encapsulated within the light grey lines. Thus, interpreting the graphs, we see a lot of data that stretches outside of the confidence bounds within both graphics. If one were to collect more data in the beer dataset we may be able to analyze this further, but as it is there seems to be not enough samples of very good beer to create a complete image of the ranking process in this dichotomous way.

```
> library(arm)
>
> # Recall our logit models:
```

[6]For more on why binned residual plotting: see Andrew Gelman (et. al's) article: `http://www.stat.columbia.edu/~gelman/research/published/dogs.pdf`

```
> logmod1 <- glm(verygood ~ price, data = beer_data,
+                family = binomial(link=logit))
>
> logmod2 <- glm(verygood ~ price + avail + alcohol + calories,
+                data = beer_data,
+                family = binomial(link=logit))
>
> ## Binned Residual plotting ("arm" package)
> par(mfrow=c(1,2))
> x <- predict(logmod1, type = "response")
> y <- resid(logmod1)
> binnedplot(x, y, nclass=NULL,
+            xlab="Expected Values", ylab="Average residual",
+            main="Binned residual plot: Model 1",
+            cex.pts=0.8, col.pts=1, col.int="gray")
>
> # Plot 2:
> x <- predict(logmod2, type = "response")
> y <- resid(logmod2)
> binnedplot(x, y, nclass=NULL,
+            xlab="Expected Values", ylab="Average residual",
+            main="Binned residual plot: Model 2",
+            cex.pts=0.8, col.pts=1, col.int="gray")
```

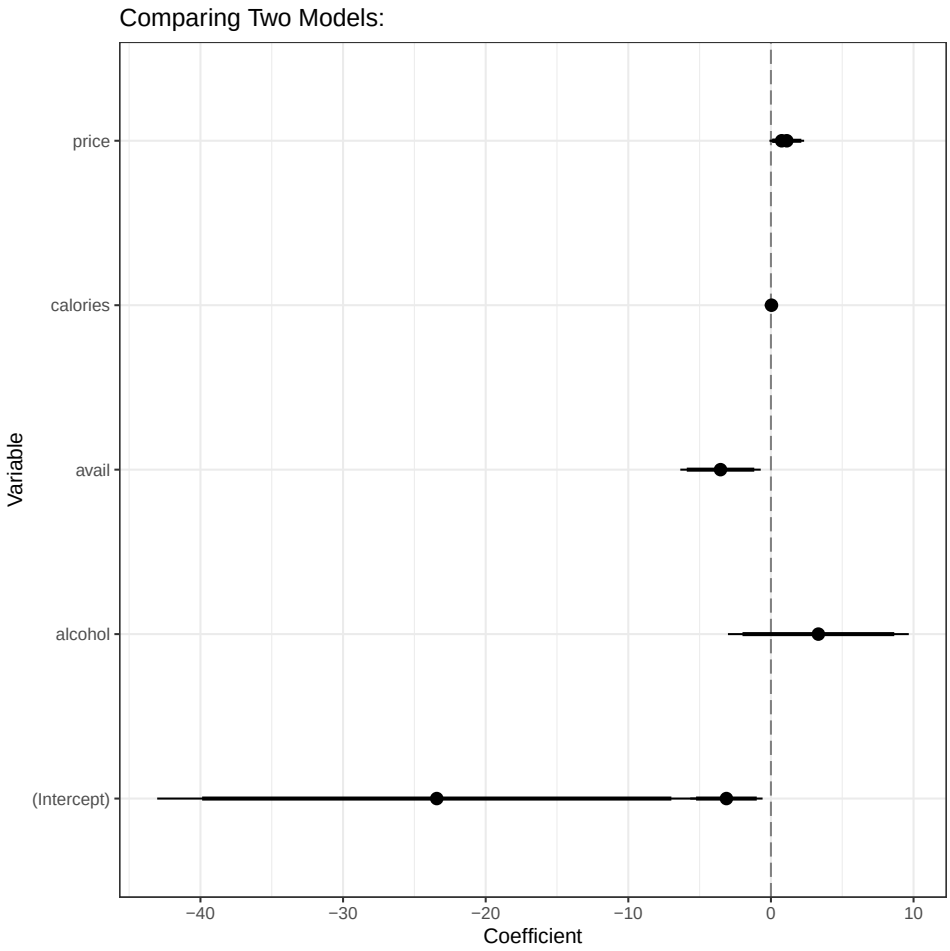



```
> par(mfrow=c(1,1)) # Reset graphing
```

The below graphic compares the initial model with our full model in another great way, oftentimes readers prefer a visualization rather than tables. Our rope ladder plot shows just how little our two models differ, except in the wide-spread intercept, and few variables falling outside of the zero mark. Availability of the beer has a notable effect and is perhaps the main take away, along with price, of the logit models. High priced, widely available beer is very good in ranking (with some modeling concerns and error).

```
> library(ggplot2)
>
> # Rope Ladder Plot (ggplot2)
> f1<-data.frame(Variable=rownames(summary(logmod1)$coef),
+                Coefficient=summary(logmod1)$coef[,1],
+                SE=summary(logmod1)$coef[,2],
+                modelName= "Model 1")
>
> f2<-data.frame(Variable=rownames(summary(logmod2)$coef),
```

```
+                 Coefficient=summary(logmod2)$coef[,1],
+                 SE=summary(logmod2)$coef[,2],
+                 modelName= "Model 2")
>
> combinedframe <- data.frame(rbind(f1, f2))
> interval1 <- qnorm((1-0.9)/2)
> interval2 <- qnorm((1-0.95)/2)
>
> rl1 <- ggplot(combinedframe)
> rl1 <- rl1 + geom_hline(yintercept = 0, color = grey(1/2), lty = 5)
> rl1 <- rl1 + geom_linerange(aes(x = Variable, ymin = Coefficient -
+ SE*interval1,
+                 ymax = Coefficient + SE*interval1),
+                 lwd = 1, position = position_dodge(width =1/2))
> rl1 <- rl1 + geom_pointrange(aes(x = Variable, y = Coefficient,
+ ymin = Coefficient - SE*interval2,
+                 ymax = Coefficient + SE*interval2),
+                 lwd = 1/2, position = position_dodge(width = 1/2),
+                 color=("Black"),
+                 shape = 21, fill = "BLACK")
> rl1 <- rl1 + coord_flip() + theme_bw()
> rl1 <- rl1 + ggtitle("Comparing Two Models:")
> print(rl1)
```


Comparing Two Models:
price
calories
avail
alcohol
(Intercept)
Variable
−40
−30
−20
−10
0
10
Coefficient

3.3 Poisson Modeling:

Poisson data uses a new link function which allows us to model count-level data, the probability mass function (pmf) now being:

$$\frac{\lambda^k e^{-\lambda}}{k!} \tag{24}$$

Where the available numbers the model will take are all real numbers greater than zero $(0; 1; 2; 3; 4; 5; ...)$. This normal Poisson model assumes the variance is equal to it's mean, but often we see the variance to be greater (over-dispersed) or lower (under-dispersed) than the mean. This assumption is oftentimes broken in social sciences, for this reason a *quasi-Poisson* or *negative binomial* is able to let the variance change.

$$\text{Quasi-Poisson:} \quad var(Y) = \theta\mu \quad \text{Where theta is the overdispersion parameter} \tag{25}$$

$$\text{Negative Binomial:} \quad var(Y) = \mu(1 + K\mu) \quad \text{Where K is the shape parameter} \tag{26}$$

How we handle the differences between the variance and mean relations in each of these three forms of count-data modeling may effect our regression coefficients fit differences between negative binomial (also known as gamma-Poisson), quasi-Poisson, and regular Poisson modeling. This is because fitting these models uses weighted least squares, and these weights are inversely proportional to the variance. Thus, negative binomial and quasi-Poisson will weight observations differently. Ultimately, the decision to choose the quasi-Poisson or another variation the negative binomial regression should depend on the theory rather than the data-driven method.[7]

Let's build a regular Poisson model, adding one variable at a time, to predict the likelihood of a successful military coup (R's faraway package). Let's assume that we theorize that the percent of a country voting, the years of oligarchy, number of elections, and political liberalization all make sense towards predicting the number (count) of successful military coups. Results are shown in Table 5 below, however here is how these models are ran in R:

```
> library(faraway)

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

> library(MASS)
> data(africa)
>
> ## Examine Dataset ##
> str(africa)  # There are some NAs in the data to consider
```

[7] Ver Hoef, Jay M. and Boveng, Peter L., "QUASI-POISSON VS. NEGATIVE BINOMIAL REGRESSION: HOW SHOULD WE MODEL OVERDISPERSED COUNT DATA?" (2007). Publications, Agencies and Staff of the U.S. Department of Commerce. Paper 142.

```
## 'data.frame': 47 obs. of  9 variables:
##  $ miltcoup : int  0 5 0 6 2 0 1 3 1 2 ...
##  $ oligarchy: int  0 7 0 13 13 0 0 14 15 0 ...
##  $ pollib   : int  2 1 NA 2 2 2 2 2 2 2 ...
##  $ parties  : int  38 34 7 62 10 34 5 14 27 4 ...
##  $ pctvote  : num  NA 45.7 20.3 17.5 34.4 ...
##  $ popn     : num  9.7 4.6 1.2 8.8 5.3 11.6 0.361 3 5.5 0.458 ...
##  $ size     : num  1247 113 582 274 28 ...
##  $ numelec  : int  0 8 5 5 3 14 2 6 4 6 ...
##  $ numregim : int  1 3 1 3 3 3 1 4 3 2 ...

> # Poisson
> mod4 <- glm(africa$miltcoup ~ africa$pctvote + africa$oligarchy +
+  africa$numelec + africa$pollib, family = poisson)
>
> # Quasi Poisson
> qmod4 <- glm(africa$miltcoup ~ africa$pctvote + africa$oligarchy +
+  africa$numelec + africa$pollib, family = quasipoisson)
```

More on Dispersion Problems:

In the Poisson distribution we make a rather strong assumption that our mean is equal to the variance due to the Poisson link function only having one parameter. We rarely ever fulfill this assumption in social sciences, but are able to report and correct for our dispersion problems pretty well. If our model is *over-dispersed* this means that our mean is a certain distribution with a peak and the variance is larger than that actual mean distribution (not being equal as our assumption posits). The inverse is defined as under-dispersion although this is less common. A few ways to control for dispersion would be to check the F-statistic to report model-fit more accurately per variable as we're building our model. Also, we can check the typical robustness checks from the linear model including the QQ plot, Cook's Distance, and residual plotting. It may be the case that a logarithmic transformation (or similar transformation) of a variable will better-fit our model without actually changing and values given that they are all scaled together. Note how the regular Poisson assumption is a special case of the Quasi-Poisson regression (where θ equals one we arrive at the Poisson assumption $var(Y) = 1\mu$. Often using Quasi-Poisson regression isn't a bad idea considering we rarely fulfill the Poisson dispersion assumption.

We can include and check the chi-squared of our model since it is compatible to our Poisson distribution:

$$\hat{\phi} = \frac{\chi^2}{n-p} = \frac{\Sigma_i (y_i - \hat{\lambda}_i)^2 / \hat{\lambda}_i}{n-p} \tag{27}$$

More easily computed in R:

Table 7: Statistical models

	Model 1	Model 2	Model 3	Model 4
Intercept	0.29	−0.52	−1.34	−0.17
	(0.28)	(0.37)	(0.61)	(0.73)
Percent Voting	0.00	0.00	0.00	0.00
	(0.01)	(0.01)	(0.01)	(0.01)
Years of Oligarchy		0.11	0.13	0.11
		(0.02)	(0.02)	(0.02)
Number of Elections			0.10	0.05
			(0.05)	(0.06)
Political Liberalization				−0.40
				(0.24)
AIC	149.55	120.73	118.61	111.67
BIC	152.98	125.87	125.47	119.59
Log Likelihood	-72.77	-57.36	-55.31	-50.83
Deviance	80.74	49.92	45.80	36.86
Num. obs.	41	41	41	36

Dependent variable: Successful Military Coups

```
> # ChiSquared of Model 4
> dp <- sum(residuals(mod4, type="pearson")^2)/mod4$df.residual
> dp

## [1] 1.155917
```

We can just run a quasi-Poisson family regression to get similar results initially. The follow is the two models comparatively:

Table 8: Statistical models

	Q-Model 4	Model 4
Intercept	−0.17	−0.17
	(0.78)	(0.73)
Percent Voting	0.00	0.00
	(0.01)	(0.01)
Years of Oligarchy	0.11	0.11
	(0.03)	(0.02)
Number of Elections	0.05	0.05
	(0.06)	(0.06)
Political Liberalization	−0.40	−0.40
	(0.25)	(0.24)
AIC		111.67
BIC		119.59
Log Likelihood		-50.83
Deviance	36.86	36.86
Num. obs.	36	36

Dependent variable: Successful Military Coups

Notice that the coefficients don't change much at all, but reconsidering our measure of the variance has changed some significance levels slightly. Note we cannot see our AIC, BIC, or Log Likelihood for the quasi-Poisson; this is because of how the variance is manipulated, a Quasi-AIC measure would be needed to report that finding. Also, we could include a negative binomial model next to the others:

```
> # Poisson
> mod4 <- glm(africa$miltcoup ~ africa$pctvote + africa$oligarchy +
+  africa$numelec + africa$pollib, family = poisson)
>
> # Quasi Poisson
> qmod4 <- glm(africa$miltcoup ~ africa$pctvote + africa$oligarchy +
+  africa$numelec + africa$pollib, family = quasipoisson)
>
> # Negative Binomial
> nbmod4 <- glm.nb(africa$miltcoup ~ africa$pctvote + africa$oligarchy +
+  africa$numelec + africa$pollib)
```

Table 9: Statistical models

	Model 4	Q-Model 4	Negative-Binomial Model 4
Intercept	−0.17	−0.17	−0.17
	(0.73)	(0.78)	(0.73)
Percent Voting	0.00	0.00	0.00
	(0.01)	(0.01)	(0.01)
Years of Oligarchy	0.11	0.11	0.11
	(0.02)	(0.03)	(0.02)
Number of Elections	0.05	0.05	0.05
	(0.06)	(0.06)	(0.06)
Political Liberalization	−0.40	−0.40	−0.40
	(0.24)	(0.25)	(0.24)
AIC	111.67		113.67
BIC	119.59		123.17
Log Likelihood	-50.83		-50.83
Deviance	36.86	36.86	36.85
Num. obs.	36	36	36

Dependent variable: Successful Military Coups

Ultimately, for publication I would include all of these in an appendix or supplemental appendix. Our results themselves are largely insignificant and our coefficients, using e^{β} we can find the multiplicative change of each variable's coefficient. The most significant finding (consistently) is the years of oligarchy a country experiences predicts the multiplicative likelihood of a military coup. For interpretation, our $\beta = .11$ so $e^{.11} = 1.116278$ or an 12% increase in the likelihood of a military coup as years of oligarchy increases. In Table 8 I do what is most intuitive for readers, report a model (in this case the negative binomial) and it's associated percentage likelihood estimate.

Table 10: Statistical models

	Negative-Binomial Model	$1 - e^{\beta}$
Intercept	−0.17	
	(0.73)	
Percent Voting	0.00	0%
	(0.01)	
Years of Oligarchy	0.11	12%
	(0.02)	
Number of Elections	0.05	5%
	(0.06)	
Political Liberalization	−0.40	−33%
	(0.24)	
AIC	113.67	
BIC	123.17	
Log Likelihood	-50.83	
Deviance	36.85	
Num. obs.	36	36

Dependent variable: Successful Military Coups

Likelihood Ratio Test, Deviance, AIC, and BIC:

In the likelihood ratio test we have two models where one is a subset (or nested) within another (following is using hypothetical data). Here we have the likelihoods of both of the restricted model ($\hat{L_R}$) divided by the unrestricted model ($\hat{L_U}$) thus taking their ratio. Using hypothetical data our restricted model logged-likelihood is -469.174 and our unrestricted model is a logged-likelihood of -406.0718. So assume here we have a restricted model with a higher logged-likelihood than the unrestricted model which isn't ideal of a likelihood ratio test. Further, our logged-likelihoods are negative which is also not ideal in creating an interpret-able likelihood ratio test. Exponentiating each of these:

$$\lambda = \frac{\hat{L_R}}{\hat{L_U}} = \frac{e^{-469.174}}{e^{-406.0718}} = 3.9360695\text{e} - 28 \tag{28}$$

Given that this doesn't really work, and our results would be helpful but we would have to just interpret arbitrary an arbitrary "cut-off" point between zero and one to determine if we can reject the null or not, we can actually just use a **logged-likelihood ratio test** which is similar but not exactly a likelihood ratio (although this is more common):

$$\lambda = -2 \ln \left(\frac{\hat{L_R}}{\hat{L_U}} \right) = -2 \left[(-469.174) - (-406.0718) \right] = 126.2044 \tag{29}$$

We could run this number in R:

```
> dchisq(126.2044, df=1)

## [1] 1.39777e-29
```

Similarly we can calculate the deviance, or the difference between a perfect fitting model and our models. Given that a perfect fitting model (for general linear models at least) are by definition perfect, the math becomes easier because perfect models can be understood as zero:

$$D = 2(0 - \ln L(\hat{\theta}|y) \tag{30}$$

$$D = -2(\ln L(\hat{\theta}|y) \tag{31}$$

Although the calculation doesn't involve pitting the models against each other we can compare model's distance away from the ideal models. We can also look at the AIC and BIC, defined similarly:

$$AIC = -2\ln L(\hat{\theta}|y) + 2p = D(\hat{\theta}) + 2p \tag{32}$$

$$BIC = -2\ln L(\hat{\theta}|x) + plog(n) = D(\hat{\theta}) + p\ln(n) \tag{33}$$

These are important to apply because if we arbitrarily add more and more variables into a model, our deviance will continue to decrease although this may not be honest to our actual understand of a "better" model (kitchen-sink models). AIC adds $2p$ where p is the number of parameters; thus, this adds a linear check against our deviance. BIC adds $p \cdot log(n)$ or a non-linear check on our deviance. Together than can help us check our model fit (and is often automatically calculated for you in tables in R)

Overall, the AIC, BIC, Log Likelihood, and Deviance are all great indicators for model fit from the ideal model and checking for many parameters (both linearly and non-linearly).

3.4 Ordered Logistic Modeling:

Ordered models take data that has a scale in responses, where a "1" response is less than or greater than a "2" response which is in similar direction to a "3" response. These ordered responses, in order to work in a GLM framework, must be formulated to be measurable at each disjointed segment between each ordered response. It may be helpful to think of this as a series of logitstic regressions between the segments, in the end bonded together. Therefore, we get the link function for the ordered logistic:

$$\frac{1}{1 + e^{-(\theta_i - w \cdot x)}} \tag{34}$$

and (optionally) the ordered probit could be used, or a sometimes attractive third option which is the exponential function to create a proportional hazards model, although survival models will be covered at the end of this document these two models are relatives of one another:

$$exp(-exp(\theta_i - w \cdot x)) \tag{35}$$

Intuitively, with a series of logistic regressions we could subtract these categories to find the effects of each category; or add these categories up to equal 1. If these categories do not add up to one the remaining number is the coefficient for a left out category. Subtracting these categories requires subtracting the inverse logit of each: $logit^{-1}(\theta_{k+1}) - logit^{-1}(\theta_k)$. AIC and BIC, note, will be including all disjointed segments. Also, by doing a regression we are not finding the areas between these "cut points" but rather the label (or response) at each cut point itself. We can take the derivative of the line estimate between cut points to find the nuanced area in between cut points, but subtracting the inverse logits (above) is essentially the same result.

Revising the data on beer, we should realize that our ranking variable is directly suited for ordered logistic regression (not OLS). Let's see if our findings differ in any way from the OLS or logit models we ran. Consider an ordinal logit of the beer ranking:

Table 11: Statistical models

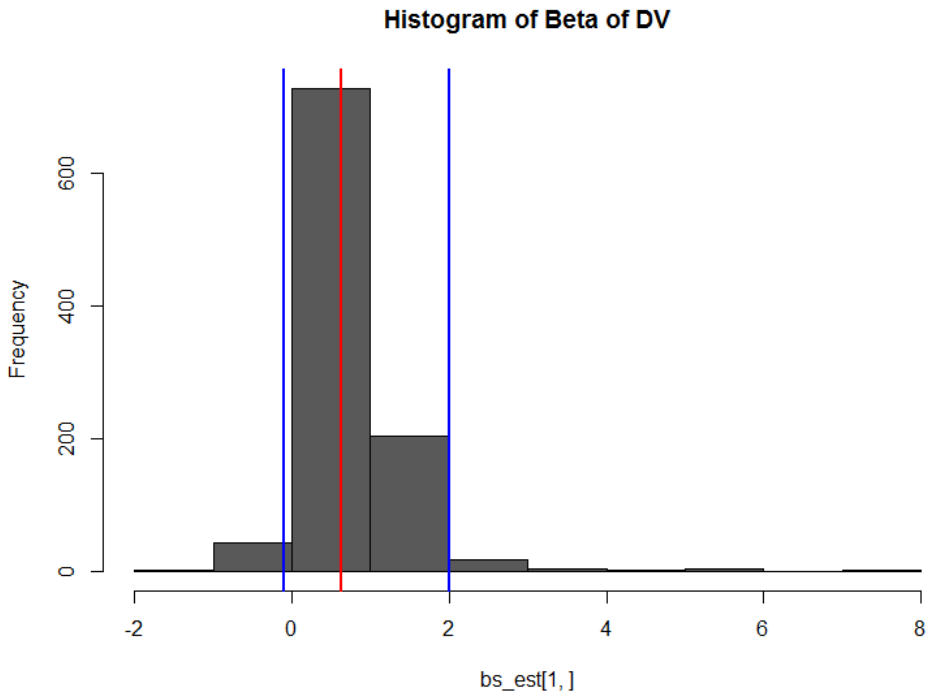
	Model 1	Model 2
Price	0.74	0.63
	(0.36)	(0.36)
Alcohol Content		1.75
		(0.82)
AIC	76.76	72.04
BIC	81.42	78.27
Log Likelihood	-35.38	-32.02
Deviance	70.76	64.04
Num. obs.	35	35

Dependent variable: Quality of Beer Ranking

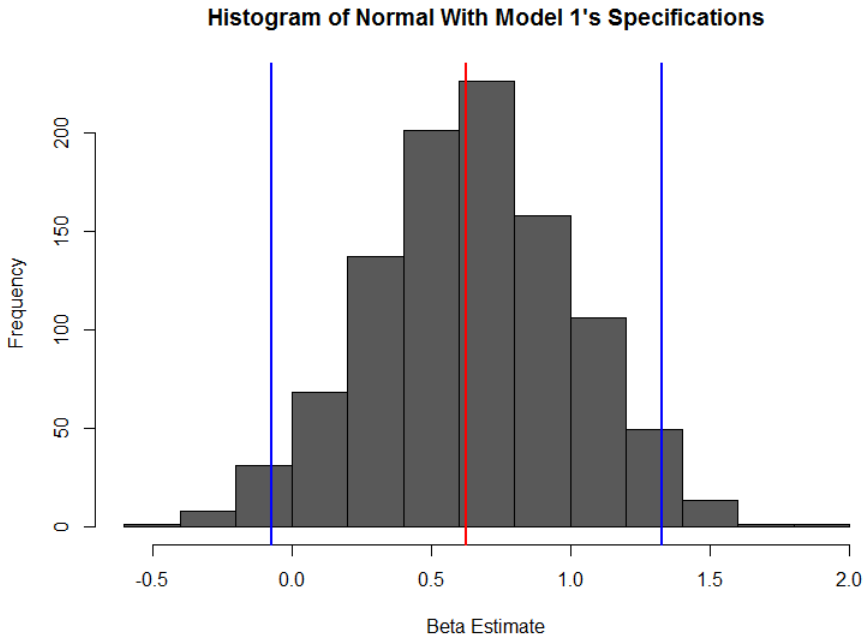
Interpretation of the results are similar to logit regression in that they are the logged odds change from one cut point to another cut point. That is, from a "2" quality to a "3" we see a 1.75 logged percentage increase per unit increase in alcohol content. This effect is statistically significant.

Bootstrapping:

One issue to consider is the low observation number (N) but high demand of the model (varying categorical demands of our dependent variable). One robustness check of this is to *bootstrap* the model to use simulation to build up the number of observations to fulfill our assumption of normal distribution which may be a hard assumption without doing this:



Above, we see our beta estimate is not a very good normal distribution even though our model assumptions assume that we should treat it like one. In order to make the assumption of normal distribution we can bootstrap the model by replicating it's draws. Figure 2 maps a normal distribution and marks the same model's beta coefficients information to show an ideal model:



Using these new-found thresholds, including a red mean line ideally where it should between the blue lower and upper bounds, we can re-run our model based upon the new information - creating new high and low confidence intervals, we find that significance still remains in Table 10:

Table 12: Statistical models

	Model 1
Price	0.63
	[−0.08; 1.95]
Alcohol Content	1.75*
	[0.62; 4.96]
AIC	72.04
BIC	78.27
Log Likelihood	-32.02
Deviance	64.04
Num. obs.	35

Dependent variable: Quality of Beer Ranking

This will typically yield more conservative estimations and evaluations of our standard errors, commonly in cases with low N. Code for this process is here but not ran, an example follows:

```
> set.seed(10)
> # This seed is used, it assures replication and convergence for this model
>
> # For Bootstrapping:
> bs.sample <- function(dat)
+ {
+   idx <- sample(1:nrow(dat), nrow(dat), replace = TRUE)
+   dat[idx,]
+ }
>
> bs.routine <- function(dat)
+ {
+   sample.dta <- bs.sample(dat)
+   coef(MASS::polr(as.factor(quality) ~ price + alcohol,
+                   data = sample.dta))
+ }
> bs_est <- replicate(1000, bs.routine(dat = beer_data))
> # replicate is actually doing the bootstrap^
>
> bs_mean <- rowMeans(bs_est)
> bs_ci <- t(apply(bs_est, 1, quantile, probs = c(0.025, 0.975)))
> bs_results <- cbind(bs_mean, bs_ci)
> colnames(bs_results) <- c("Est", "Low", "High")
>
> # Model build
> mod1 <- MASS::polr(as.factor(quality) ~ price + alcohol, data=beer_data)
> # Quick Results
> texreg::screenreg(mod1,
+                   override.ci.low = c(bs_results[, 2]),
+                   override.ci.up = c(bs_results[, 3]))
>
> # Table, to show overriding the confidence intervals with the bootstrap
> # library(texreg) # For this:
> # texreg(l= list(mod11), stars = numeric(0),
> #        custom.coef.names = c("Price", "Alcohol Content"),
> #        override.ci.low = c(bs_results[, 2]),
> #        override.ci.up = c(bs_results[, 3]),
> #        caption.above = T, float.pos = "h!",
> #        custom.note = "Dependent variable: Quality of Beer Ranking" )
```

Model Interpretation

Using a different dataset, assume R gives us the following raw output involving giving aid to Katrina relief:

`Model:`

	Coefficient	Std. Error	0.95 CI Lower	0.95 CI Upper
alt	1.2024	0.6335	-0.0393	2.4441
age01	3.4481*	1.1723	1.1504	5.7458
female	0.9407*	0.4333	0.0915	1.7899
inc8cat	-0.0188	0.7181	-1.4262	1.3886
advdeg	-0.9940*	0.4894	-1.9531	-0.0348
nonwhite	-0.5211	0.4990	-1.4991	0.4569
pid7cat	0.7343	0.9204	-1.0697	2.5384
0 | 0.5	0.9497	0.7954	-0.6092	2.5086
0.5 | 1	1.5801	0.8065	-0.0007	3.1608
Log Likelihood:	-89.63342 (df=9)			

Three variables are within two standard errors, age, if the respondent is female, and their average degree held. Interpreting ordered coefficients, since our dependent variable is ordered and many of our independent variables are ordered we will often care about the inverse logged difference between variables to find the rate of change between one ordered "cut-point" to the next. The difference between these two cut points recall:

$$logit^{-1}(\theta_{k+1}) - logit^{-1}(\theta_k) \tag{36}$$

Where "k" is the comparable category across the variable of interest. For example, we can compare our β from our zero to .5 cut point to our .5 to 1 cut point in our dependent variable: donating to Katrina relief.

$$logit^{-1}(.9497) - logit^{-1}(1.5801) \approx (8.9) - (38) \approx -29.1 \tag{37}$$

The rate of change between our .5 cut point (considering donating) and our cut point of 1 (has already donated) is -29.1. This predictive estimation is **all in reference to the omitted category**: zero or those who do not plan to donate given that all other variables are set at one. We find that having a low altruism score, being a minority race, and having a low average degree but highly liberal are all characteristics of someone that is far less likely to donate - or for us to observe a shift from a .5 to a 1 in the donation categories.

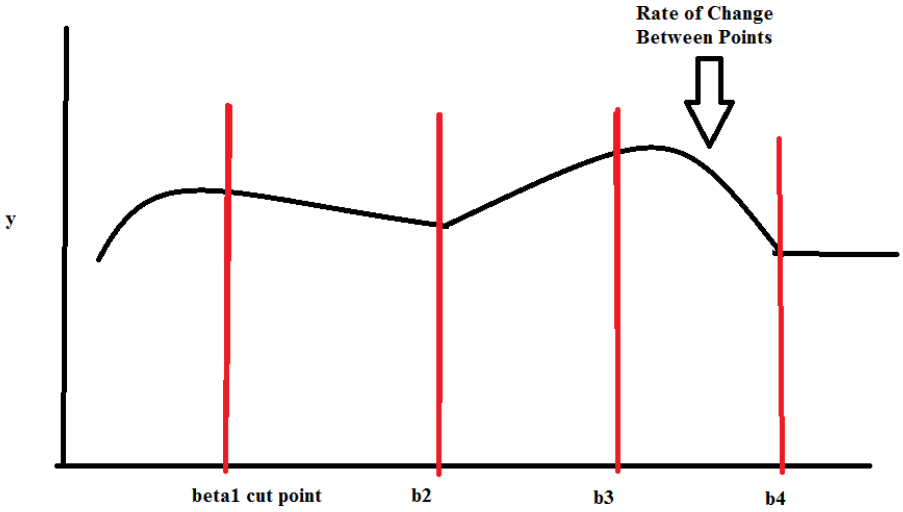
However, predicting a category like altruism (which ranges from 1-10) can be far different. Given the range of categories the variable could just be treated just as a variable would in a standard linear model, but the interpretation again is the **relative change from the removed category 0** (not considering a donation). In this instance we see a 1.2 percentage increase in the logged likelihood of donating for every unit increase in altruism.

Latent Spaces

This question has come up in the previous model's interpretation. What we obtain from a categorical variable is various "cut points" or snapshots of some longer distribution drawn out between these cuts in our distribution. These thresholds only tell us the associative values at each of these points, it is on the researcher to either integrate or use the difference in inverse logs to find the rate of change between points - or the latent spaces between the thresholds.

The more categories we add in initial data collection the easier it becomes to measure these latent spaces, for example age (a continuous variable) could be categorized as Teen,

Young Adult, Adult, Elderly. Or, we could collect more data and have it by year - or by day! Measuring the difference between groups, say form Adult to Elderly, would take the probability at the stage of being an adult and everything greater than it $(k+1)$ minus being Elderly and everything less than it (k) and then we find the effect from one threshold to the next. As this crude sketch shows, this is important because our distribution *between* cut points could be interesting to understand.



3.5 Multinomial Nominal Modeling:

In multinomial modeling we have categories that have no particular order among them (religious identification for example, or other nominal variables). This is another extension of the binomial regression, except in 0-1 we have 0 all the way up to the jth group. The probability of all of these categories have to sum to 1. To use the GLM framework we use the following link function:

$$\eta_{ij} = x_i^T \beta_j = log \frac{p_{ij}}{p_{i1}} \quad \forall \quad j = 2, ..., J \tag{38}$$

Since these categories have to equal to 1, we can manipulate and set the baseline category (usually whatever is coded 1 in R) to come up with a link that looks very similar to the logistic link, only with a summation operator for the categories starting beyond the omitted category, at $j = 2$ to the final category J:

$$p_{ij} = \frac{exp(\eta_{ij})}{1 + \Sigma_{j=2}^{J} exp(\eta_{ij})} \tag{39}$$

The only difference between this and the logistic regression is the summation operator for multiple categories rather than just the two in logistic regression. To run multinomial data in `R` we use the `nnet` package, which includes a lot on neural networks but also has a great multinomial function:

```
> library(nnet)
> object <- multinom(formula)
```

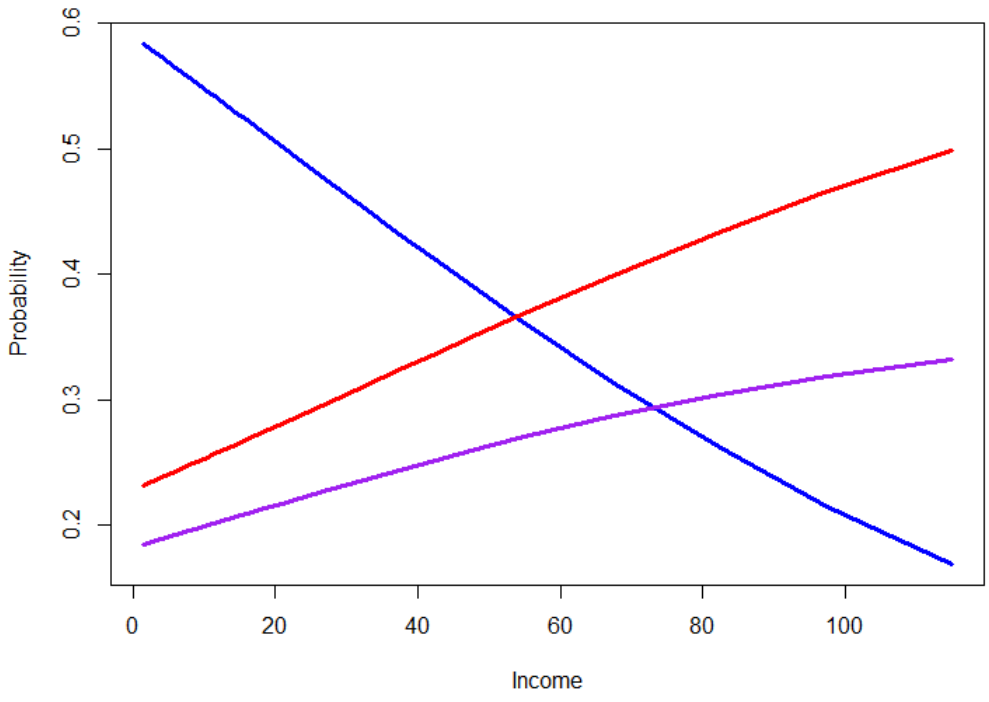
The following table is what a multinomial output looks like, created using the `R faraway` package, utilizing the NES 1996 data. Assume, for some reason, all we needed to add in our model was age and the amount of TV someone watched to predict their partisan leanings.

The raw coefficient we get is the logged odds from moving from the deleted category (Democrat here) to either Independent or Republican. Logged odds are pretty unituitive however, so making predictive probabilities from multinomial coefficients (and indeed much of regression output) is helpful to readers:

Table 13: Statistical models

	Independent	Republican
Intercept	**−2.71***	−0.36
	[−3.73; −1.69]	[−0.76; 0.04]
Age	0.01	0.01
	[−0.02; 0.03]	[−0.00; 0.02]
TV News	−0.06	−0.05
	[−0.19; 0.08]	[−0.10; 0.01]
AIC	1571.79	1571.79
BIC	1600.89	1600.89
Log Likelihood	-779.90	-779.90
Deviance	1559.79	1559.79
Num. obs.	944	944

* 0 outside the confidence interval



The above plot is modeling income along the x axis and the probability of being Democrat, Republican, or Independent (purple) for the NES 1996 data. This is using the "predict" command in R, which generates predictive probabilities out of our coefficients.

```
> predict(mmodi,data.frame(nincome=il),type="probs") # include type="probs"
> predict(mmodi,data.frame(nincome=il))
>
> # Create Object with this probability matrix
> prmat <- predict(mmodi,data.frame(nincome=sort(unique(nincome))),
+          type="probs")
>
> # Plotting:
> plot(unique(nincome), prmat[,1], type="l", col="blue", lwd=3,
+        xlab="Income", ylab="Probability") # Labels
> lines(unique(nincome), prmat[,3], type="l", col="red", lwd=3)
> lines(unique(nincome), prmat[,2], type="l", col="purple", lwd=3)
>
> summary(mmodi)
```

3.6 Survival Modeling:

Survival models (event history models) take data that occurs over a duration of time and maps the likelihood of surviving until the next "step". This could be, for example, the effect of a treatment, the lifeline of a patent in therapy, the tenure of a officeholder, or the length that a bill or law stays active over time. Consider the table:

```
> library(survminer)
> library(survival)
>
> # Exponential:
> expmod <- survreg(Surv(te, censor) ~ wc + dem + pvote + south,
+                   data = dat, dist = "exponential")
>
> # Weibull:
> weibullmod <- survreg(Surv(te, censor) ~ wc + dem + pvote + south,
+                   data = dat, dist = "weibull")
```

Table 14: Survival Models

	Exponential	Weibull	exp(Cox)
Intercept	3.60	3.59	
	(0.10)	(0.10)	
Investiture	−0.36	−0.34	
	(0.13)	(0.12)	
Polarization	−0.039	−0.04	1.04
	(0.01)	(0.005)	(0.01)
Crisis Duration	0.01	0.01	0.99
	(0.002)	(0.002)	(0.002)
Log(Scale)		−0.12	
		(0.05)	
Scale	Fixed 1	0.884	
Log Likelihood	-1056.7	-1053.8	56.6
χ^2	88.08	93.6	
df	3	3	2
Num. obs.	314	314	314

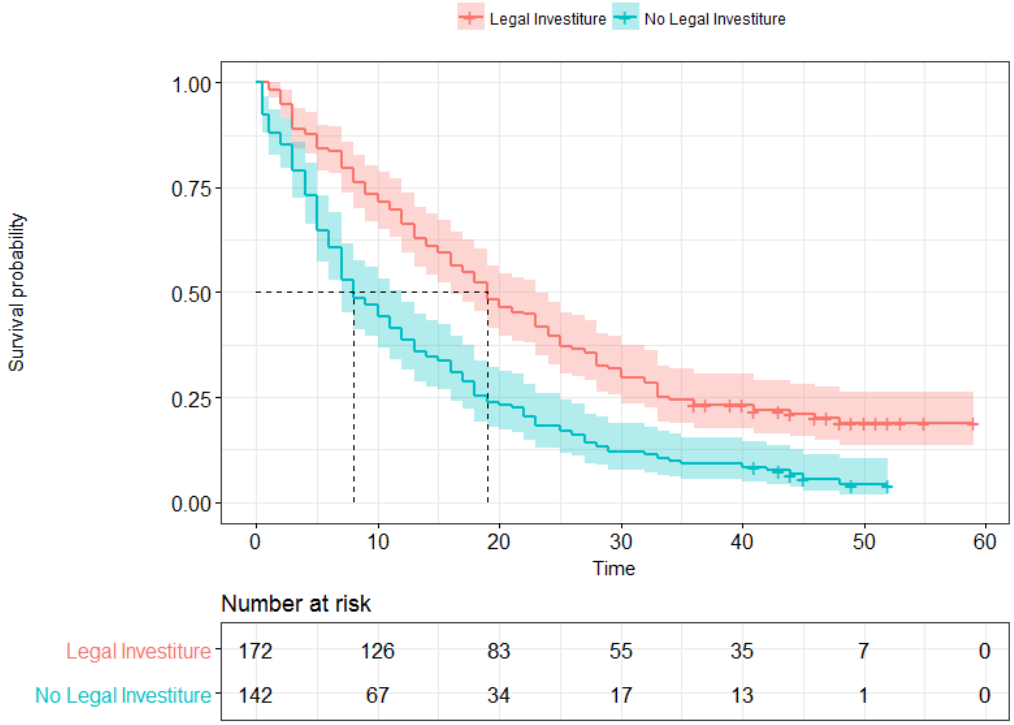
Dependent variable: Duration

The above table is the coefficients and standard errors for two models (and an expoentiated Cox model), the exponential and Weibull of a survival modeling of duration given legislative investiture, polarization measures, and crisis duration (measured in days). Our variable of interest here is the presence of legislative investiture and Gary King's claim

that they prevent against failure in international national cabinets (their duration).[8]

In interpretation, The exponential model assumes the scale parameter to be uniform for our hazard model (uniform at 1). This, in a basic sign-and-significance interpretation yields negative relationships for investiture policy and polarization which are both significant but weakly influential. The Weibull model gives us the actual scale parameter (.884) and similar results although slightly weaker when accounting for distributional changes additionally the interpretation changes via in absolute value terms from the exponential to the Weibull model. The Cox model assumes that the two strata within the survival curves will be proportional to each other, which could be an easy assumption to fulfill in some theory. Finally, the exponentiation of the Cox model can be read similarly to the Poisson model and is just a measure *of the effect of the hazard rate*, which again is the risk of failure given that the participant has survived up to a specific time. The Cox model output here does not include our strata (investiture).[9]

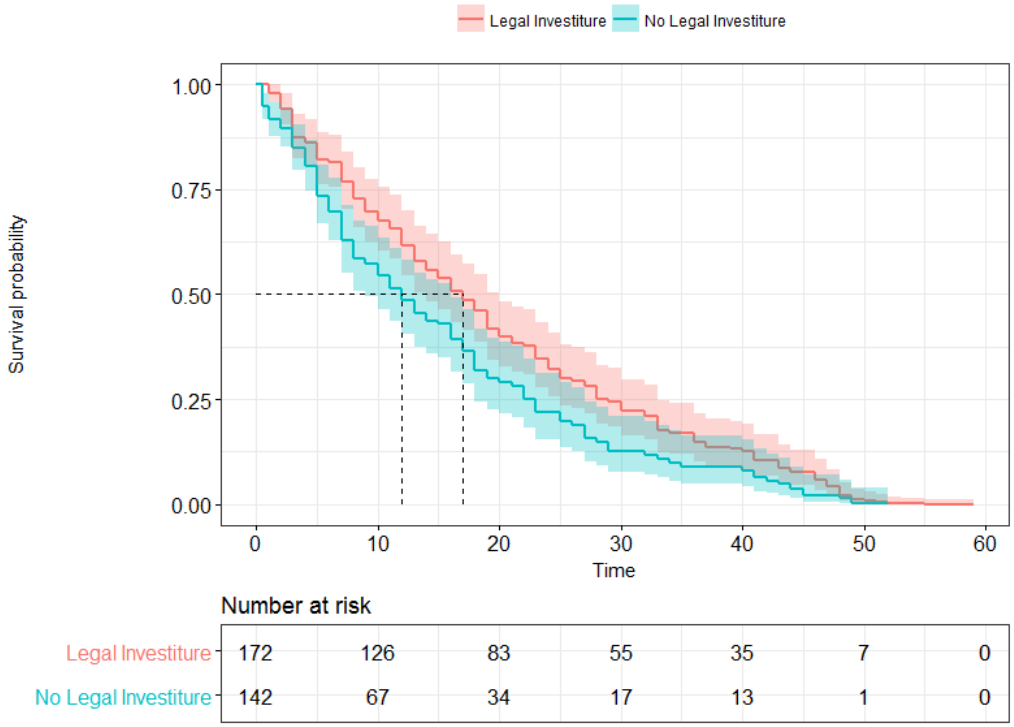
Figure 1: Exponential Survival Model of Legislative Investiture



[8]King, Gary, James E. Alt, Nancy Elizabeth Burns and Michael Laver (1990). "A United Model of Cabinet Dissolution in Parliamentary Democracies," American Journal of Political Science, vol. 34, no. 3, pp. 846-870.

[9]`coxph(Surv(duration, ciep12) ~ strata(invest) + polar + crisis, data = dat, method = "efron")`

Figure 2: Cox Survival Model of Legislative Investiture




```
> #Survival GGplot Example (not used to replicate above, code as template)
>
> ggsurvplot(fit, data = dataset)
>
> ggsurvplot(fit, data = dataset,
+                 surv.median.line = "hv",  # Add medians survival
+                         # Change legends: title & labels
+                 legend.title = "Title",
+                 legend.labs = c("One Category", "Another"),
+                         # Add p-value and confidence intervals
+                 pval = TRUE,
+                 conf.int = TRUE,
+                         # Add risk table
+                 risk.table = TRUE,
+                 tables.height = 0.2,
+                 tables.theme = theme_cleantable(),
+                 #Load "GrandBudapest" color scheme
+                 palette = c(wes_palette("GrandBudapest", 2,
+                 type = "discrete")[1],
+                 wes_palette("GrandBudapest", 2,
+                 type = "discrete")[2]),
+                 ggtheme = theme_bw()  #Change ggplot2 theme
```

```
+                    )
```

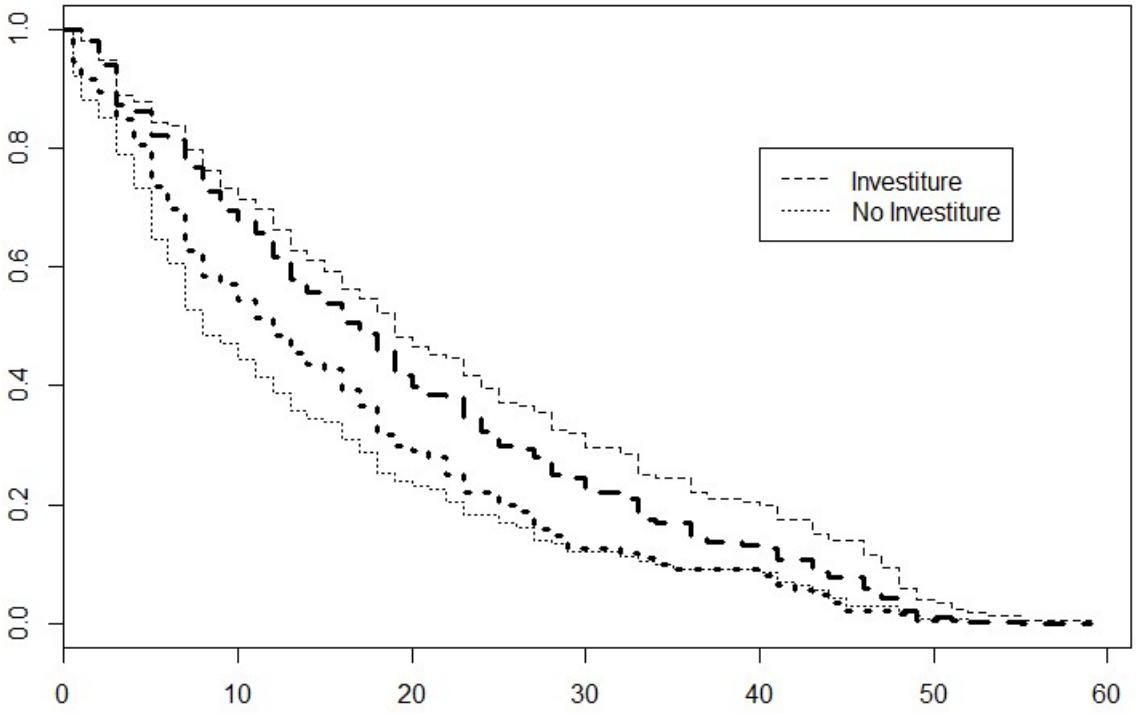
Survival modeling is a variation of the generalized linear model which accounts for predicted probabilities as a phenomenon or actual actor dies in the course of study. Each depression in the figures is the instance of death in the dependent variable. Exponential modeling holds the hazard as uniform (1) giving us an interpretation of the predicted chance of dying. Weibell, in contrary, is the predicted probability of living and does not make an assumption on the scale parameter and reports this hazard parameter to the researcher (resulting in a loss of one degree of freedom).[10] Confidence intervals for both Figure 1 and 2 are set at 95% and the number at risk are the remaining participants in the study at each time location. The dotted lines report the median of the data for each strata (Investiture). Finally, given the Kaplan-Meier modeling here each depression is robust in that it controls for a multiple-drop off scenario to where $1/n$ can be calculated. For example, if three people were to die off at the same time point a $1/10, 1/9, 1/8$ would be calculated instead of all three at the same functional moment. The number remaining at risk is shown descriptively below each figure.

Interpretation of the actual coefficients is very similar to Poisson modeling given that our general linear model is following similar distribution as the Poisson (non-negative). Yet, the Poisson uses count-integer level data whereas survival modeling can use continuous data. Because of the similarities, the beta coefficient reported is similar, it is the multiplicative effect holding all else constant of a one unit increase in X onto Y.

I have been able to plot both of these together in base-R; but I don't like how it looks compared to `ggplot`.

Figure 3: Survival Models of Legislative Investiture in Base R

[10] Recall that this is very similar to the scale/variance problem we occurred in Poisson regression. Similarities are very similar here as well, and the math is similar in this respect too.


1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
60
Investiture
No Investiture

Further Notes: Comments

Censoring Problem in Survival Models:

Perhaps as an additional note, a unique problem to survival modelings is the issue of censoring. Censoring is where participants (or the observed phenomenon)lives beyond the end point of the study. This is an example of rightward-censoring, but leftward-censoring would just be the inverse (beginning a study beyond the start of the phenomenon). This is a unique and difficult threat to survival models, thus we have had to use much of the tools listed in this section to account for these kinds of data. [11]

iid:

The GLM nature of the survival model does not give it a free pass on the independent and individually distributed (iid) assumption. In fact, much of the time in dealing with survival-type theories (bill-survival, authoritarian regimes, cancer patients) we deal with responses that are not independent of one-another. A doctor seeing cancer patients, for example, may see twenty patients and as they become fully treated or pass away he may have more time to distribute to other patients. These patients may even meet and have group-therapy among themselves (just as bills may co-vary and authoritarians interlock in strategy)! Further methodology (networks for instance) can help circumvent these concerns.

[11] Princeton article here includes mathematically how this and much of survival analysis works: http://data.princeton.edu/wws509/notes/c7.pdf

3.7 Conclusion, Quick Reference

The GLM

The general linear model framework consists of a stochastic and systematic component, in order to use other distributions in the exponential family we need a *link function* and to change our distriubtion. These build connections (links) between the mean of the DGP to our linear predictors.

Linear Model

It's origins in geometry, the standard linear model is initially pretty flexible to most social science data. In geometry, $y = mx + b$ can be rewritten for data: $y = \alpha + \beta X$ or $y = \beta_0 + \beta X$. where α or β_0 is the y-intercept, β_i is the ith coefficient for each variable onto our dependent variable y and X is our observed independent variable data.

In R this is executed simply:

```
object <- lm(formula)
#formula: (DV ~ IV + IV + IV)
```

Coefficients received in a standard linear model table are interpreted as a one unit increase in X as an effect on our DV (pg. 4 for more, and probability plotting).

Logit Regression:

Logit (and the alternative probit) regression(s) examine dependent variables that are coded dichotomous (0;1). This is accomplished by the link function $p\left(\frac{p}{1-p}\right)$.

If coded correctly, logit regression can give you the logged odds from moving from a zero (event not happening) to one (event happening). Yet, we often don't think in terms of logged odds, so we can calculate predictive probabilities by dividing the coefficient by four - this will yield a maximum predictive probability when the coefficient rests in the middle of the logistic curve.

More helpful for readers, perhaps, is to generate a probability plot, including data points along the x axis and the probability(y) along the logistic curve on the y axis.

Poisson Regression:

Poisson, Quasi-Poisson, and Negative Binomial regression all take count level data. Poisson assumes the variance to be held equal to the mean (a "1" in Quasi-Poisson); whereas Quasi-Poisson and Negative Binomial allow the variance to differentiate between models. In R:

```
> # Poisson
> object <- glm(formula, family = poisson)
>
> # Quasi-Poisson
> object <- glm(formula, family = quasipoisson)
>
> # Negative Binomial/ Gamma-Poisson
> object <- glm.nb(formula)
```

For interpretation, it is best to exponentiate the coefficients (e^{β}) to receive the multiplicative change exerted from each variable (keeping in mind significance). Tables can be created that show readers these effects (pg. 18-19).

Ordinal Regression:

Ordinal data requires the interpretation of cut points (k) within our regression and is interpreted similarly to logistic regression. Ordinal data between cut points can be thought of as a series of logistic regressions between cut points.

```
> # Ordinal Regression
> object <- polr(formula)
```

It is important to make note of the ordinal categories, one category is omitted as a reference category within regression and the information received is in refference to the omitted category. Inverse logit subtraction can yield predictive probabilities: $logit^{-1}(\theta_{k+1}) - logit^{-1}(\theta_k)$ these latent space estimates are important to note as, when controlling for all else, the relative change between ordinal responses (feeling "okay" to feeling "great" for example).

Multinomial Nominal Modeling:

```
> library(nnet)
> object <- multinom(formula)
```

Multinomial modeling includes data that is non-ordered, for example religious identity. Mathematically the link function (considering all probabilities for each category need add up to one) is very similar to the logistic link.

With the raw coefficients we get from multinomial data we receive the logged odds (like logistic regression) from moving from the omitted category (usually the first category in R) to the following categories. Again, we don't typically think in logged odds (another

similarity to logistic regression) so making predictive probability plots are helpful (again, similar to logistic regression).

Survival Modeling:

Survival models (as the name suggests) measures and indeed explicitly addresses the length of time a unit dies or survives across a length of time. The hazard parameter is held constant in the exponential model (at 1) whereas in the Weibull model it is allowed to vary (losing one degree of freedom, but this choice would depend on theory). To model survival data:

```
> library(survminer)
> library(survival)
>
> expmod <- survreg(Surv(te, censor) ~ wc + dem + pvote + south,
+                   data = dat, dist = "exponential")
>
> weibullmod <- survreg(Surv(te, censor) ~ wc + dem + pvote + south,
+                   data = dat, dist = "weibull")
```

In interpretation of these models' output, we should consider it similar to the Poisson modeling: the multiplicative effect, holding all else constant, of a one unit increase in X onto Y. Although predictive probability plots are (again) superior to present these results.

Mapping graphs as shown in full in Section 7, are a intuitive and attractive way to communicate results to the reader. In these we separate two interesting strata and map the survival data accordingly, controlling for what we deem important with our theory. Kaplan-Meier checks are also important because multiple "deaths" could occur at once and this will scale each of these instead of all at once - which is better.

4 Machine Learning

Machine learning has come to be popular among data scientists, chiefly for its ability to remove the researcher and have more streamlined ways to empirically examine categorization and estimands. By definition there are two types of machine learning, supervised and unsupervised. The prior includes researcher influence, classifications and penalizing regressions. The latter includes algorithmic detection of classifications and estimands through a (dare I say) machine-like process.

Types of Machine Learning

$$\begin{cases} \text{Supervised learning} \begin{cases} \text{Classification} \\ \text{Regression} \end{cases} \\ \text{Unsupervised learning} \begin{cases} \text{Discovering clusters} \\ \text{Discovering latent factors} \\ \text{Discovering graph structure} \\ \text{Matrix completion} \end{cases} \end{cases}$$

A large benefit of this section will come in making synthetic data to test some fundamental assumptions of the GLM, all before introducing machine learning as a way to help us reach similar or better conclusions without the threat of researcher bias and ensured reproducability. For this reason, even an novice to graduate methods may find helpful the code and testing that exists in this section, as I have.

4.1 Simulation and Replication

Let's begin by setting up a simulated regression, which will have similar architecture throughout this chapter. This particular regression will have two random variables (x_1 and x_2) for N observations. Then, we create our own coefficent values (β) and errors (e). A vector of ones is our intercept ($alpha$) and this is all neatly placed into a matrix. From this random data we can test some intial regressions.

```
> set.seed(12345)
>
> N <- 1000
> # Two random x variables, with same mean and same error. Distributed Normally
> x_1 <- rnorm(N, mean=5, sd=sqrt(16))
> x_2 <- rnorm(N, mean=5, sd=sqrt(16))
> # beta values to multiply with the Intercept, X_1, and X_2 matrix.
> b <- c(2, -1, 3)
> # Perfect-world errors: e~Normal(0, sig^2). set pretty low, Number=1000
> e <- rnorm(N, 0, sd=sqrt(4))
> # This vector of ones is to generate our intercept along with the "2" in beta
> ones <- matrix(1, nrow=1000, ncol=1)
> X <- matrix(c(ones, x_1, x_2), nrow=N)
```

```
> # head(X)  # Checked that the Matrix comes out right, and is intuitive
> # y = intercept + b1*X_1 + b2*X_2 + some error
> y <- b[1]*X[,1]+ b[2]*X[,2]+ b[3]*X[,3] + e
> my_data <- data.frame(x_1, x_2, y)
> # After having y, we can run our formula, OLS using lm()
> m1 <- lm(y ~ x_1 + x_2, data=my_data)
> summary(m1)$coef

##              Estimate Std. Error   t value     Pr(>|t|)
## (Intercept)  1.992134 0.12163219  16.37834 1.420043e-53
## x_1         -1.005716 0.01517949 -66.25491 0.000000e+00
## x_2          2.997224 0.01503489 199.35128 0.000000e+00
```

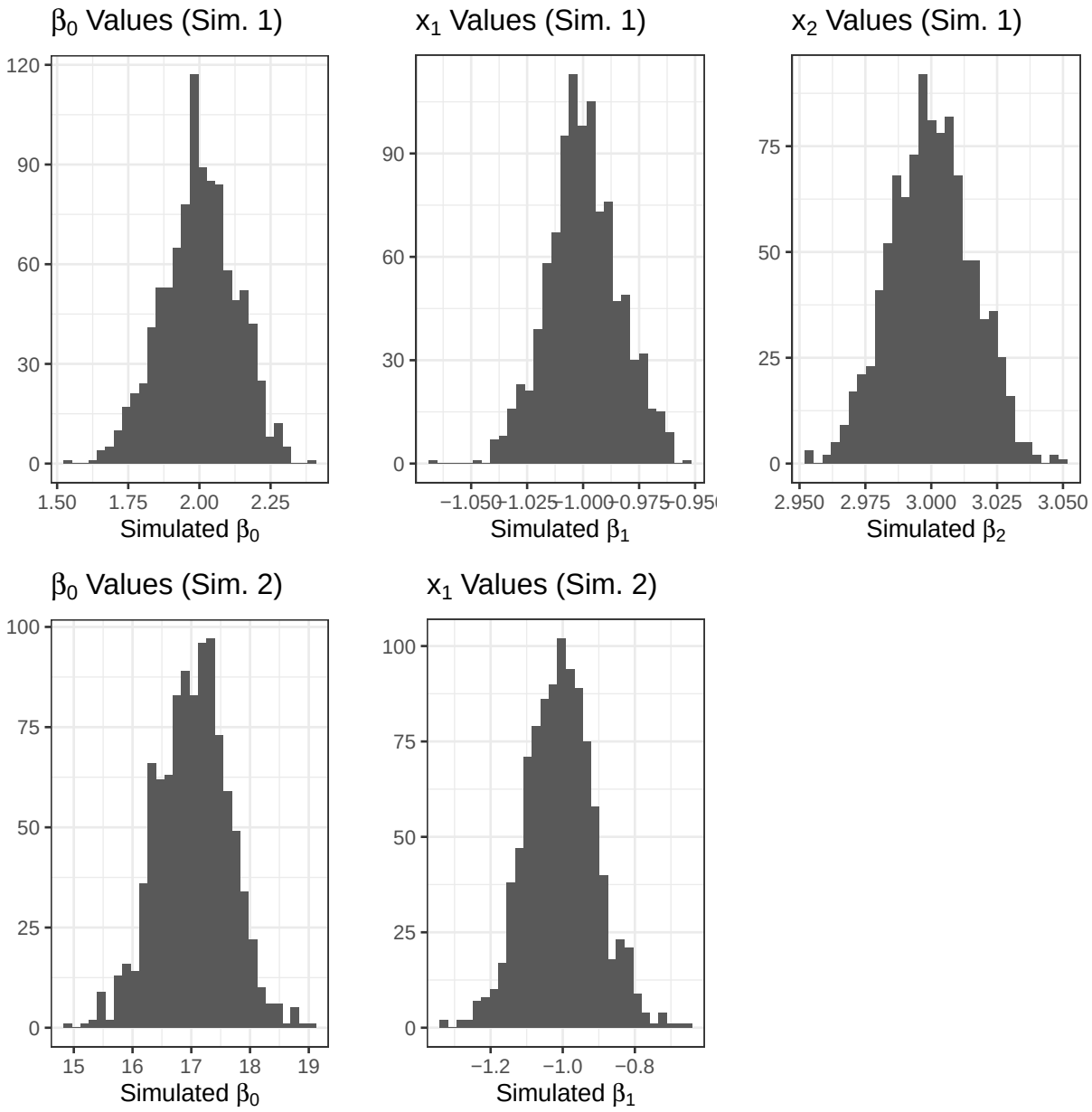
We find that our estimates in our model come very close to our defined beta estimates, this makes sense since our betas were pre-defined this way. Each variable is statistically significant given we entered a pretty low error value into the model. Let's take the above regression and simulate it 1,000 times through a loop:

```
> # Let's set up an Empty matrix
> simulation1 <- matrix(nrow = 1000, ncol = 6)
> # head(simulation1)  # Matrix of NA's to fill in simulated values later.
>
> for (i in 1:1000)
+ {
+   N <- 1000
+   x_1 <- rnorm(N, mean=5, sd=sqrt(16))
+   x_2 <- rnorm(N, mean=5, sd=sqrt(16))
+   b <- c(2, -1, 3)
+   e <- rnorm(N, 0, sd=sqrt(4))
+   ones <- matrix(1, nrow=1000, ncol=1)
+   X <- matrix(c(ones, x_1, x_2), nrow=N)
+   y <- b[1]*X[,1]+ b[2]*X[,2]+ b[3]*X[,3] + e
+   my_data <- data.frame(x_1, x_2, y)
+   m1 <- lm(y ~ x_1 + x_2)

+   sm1 <- summary(m1)  # summary table is used to easily locate standard errors

+   simulation1[i,1] <- m1$coef[1]    # storing models first coef: Intercept
+   simulation1[i,2] <- m1$coef[2]    # x_1 coef estimate
+   simulation1[i,3] <- m1$coef[3]    # x_2 coef estimate
+   simulation1[i,4] <- sm1$coefficients[1,2]   # standard error for Intercept
+   simulation1[i,5] <- sm1$coefficients[2,2]   # se x_1
+   simulation1[i,6] <- sm1$coefficients[3,2]   # se x_2
+ }
>
```

```
> # head(simulation1)  # To check our now-full matrix to see if values make sense
```

Next, we will do the same simulation but removing x_2 in our model. Then we will plot each of the variables from the two simulations. The code for plotting has been witheld to save space on page, but it is done using ggplot and a function to make the plots appear next to one another.

```
> # Removing x_2 now
> #  New empty matrix
> simulation2 <- matrix(nrow = 1000, ncol = 4)
> # head(simulation2)
>
> for (i in 1:1000)
+ {
+   N <- 1000
+   x_1 <- rnorm(N, mean=5, sd=sqrt(16))
+   x_2 <- rnorm(N, mean=5, sd=sqrt(16))
+   b <- c(2, -1, 3)
+   e <- rnorm(N, 0, sd=sqrt(4))
+   ones <- matrix(1, nrow=1000, ncol=1)
+   X <- matrix(c(ones, x_1, x_2), nrow=N)
+   y <- b[1]*X[,1]+ b[2]*X[,2]+ b[3]*X[,3] + e
+   m1 <- lm(y ~ x_1)                #! no X_2 regressed, but included in DGP

+   sm1 <- summary(m1)

+   simulation2[i,1] <- sm1$coef[1,1]     #Storing models first coef, Intercept
+   simulation2[i,2] <- sm1$coef[2,1]     #x_1 coef estimate
+   simulation2[i,3] <- sm1$coef[1,2]      #standard error for Intercept
+   simulation2[i,4] <- sm1$coef[2,2]      #se for x_1
+ }
>
> # Full Model Means
> #   mean(simulation1[,1]) # mean estimate for b_0 (= 2)
> #   mean(simulation1[,2]) # mean estimate for b_1 X_1 estimate (= -1)
> #   mean(simulation1[,3]) # mean estimate for b_2 x_2 estimate (= 3)
>
> # Limited Model Means
> #   mean(simulation2[,1]) #mean b_0 (=2)
> #   mean(simulation2[,2]) #mean b_1 of x_1 (=-1)
> # This makes sense given our initial model data b=c(2, -1, 3)
```



As is expected, most things remain the same between the two simulations, however in simulation two removing a variable (imagine this being race, education, or some economic variable) biases our estimate of the intercept. More importantly, this will balloon the errors (hopefully because that's what errors are supposed to tell us):

```
> mean(simulation1[,5]) # Simulation 1's X_1 se's (0.0158)

## [1] 0.01581834

> mean(simulation2[,4]) # Simulation 2's X_1 se's (0.0962)
```

```
## [1] 0.09632281

> mean(simulation1[,4])  # Simulation 1's Intercept se (.128)

## [1] 0.1285995

> mean(simulation2[,3])  # Simulation 2's Intercept se (.615)

## [1] 0.6168927

> # Errors are larger when we take x_2 out of our analysis
> #   (less variation captured from DGP)
```

One advantage of using simulated data is the complete controll and education available from messing around with things without any consequences (most of the time). Now that we have tested some of the fundementals of the linear model and reporting estimates, let's use simulation to try to fit a linear model to binary data.

Let us start with just making a fun function that creates inverse logit estimands.

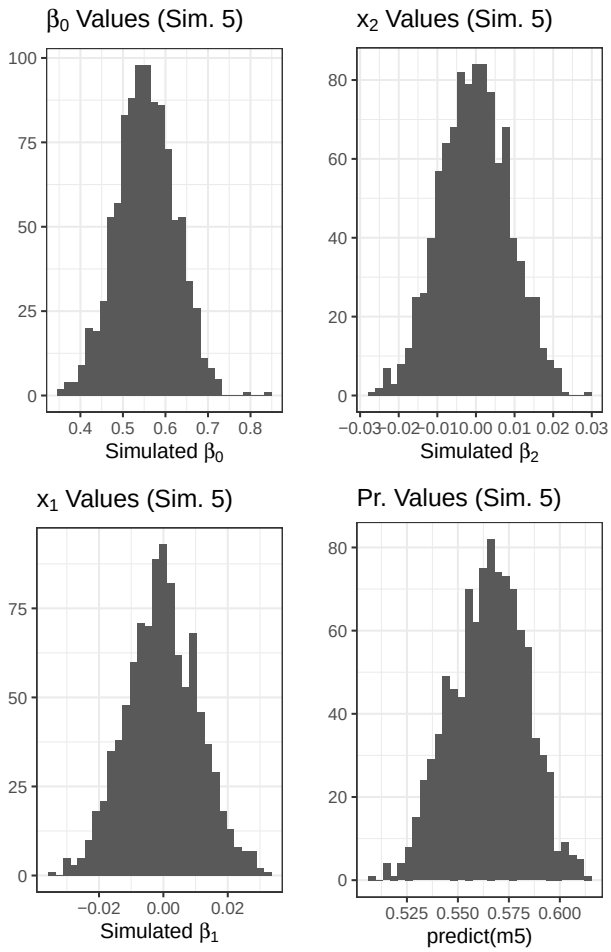
```
> inv.logit = function(x){
+   if(!is.numeric(x)){return("Error 404: Numbers Not Found")}
+   exp(x)/(1+exp(x))
+ }
>
> inv.logit(.95)

## [1] 0.7211152

> inv.logit("Normative")

## [1] "Error 404: Numbers Not Found"
```

Now, like before, we will construct a DGP via simulation:

```
> library(Rlab) # for rbern() function

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

> simulation5 <- matrix(nrow = 1000, ncol = 6)
> # head(simulation5)
>
> for (i in 1:1000)
+ {
+   N <- 1000
+   mu <- c(5,5)
+   Sigma <- matrix( c(2, .5, .5, 3 ), nrow = 2, ncol= 2)
```

```
+   mvr <- mvrnorm(N, mu, Sigma) # using MASS package
+   b <- c(.2, -1, 1.2)
+   e <- rnorm(N, 0, sd=sqrt(4))
+   ones <- matrix(1, nrow=1000, ncol=1)
+   X <- matrix(c(ones, mvr[,1], mvr[,2]), nrow=N)
+   y <- b[1]*X[,1]+ b[2]*X[,2]+ b[3]*X[,3] + e
+   y <- as.numeric(y >= median(y)) # Setting y 0-1 based around median.

+   # Inverse Logits
+   # Intercept
+   inv.logit.a <- inv.logit(b[1]*X[,1])
+   b0 <- matrix(inv.logit.a, nrow=N, ncol= 1)

+   #X_1
+   inv.logit.x2 <- inv.logit(b[2]*X[,2])
+   b1 <- matrix(inv.logit.x2, nrow=N, ncol= 1)

+   #X_2
+   inv.logit.x3 <- inv.logit(b[3]*X[,3])
+   b2 <- matrix(inv.logit.x3, nrow=N, ncol= 1)

+   pi <- matrix(c(b0, b1, b2), nrow=N, ncol=3)
+   y <- rbern(N, pi)

+   m5 <- lm(y ~ X[,2]+ X[,3]) # linear model to binary data...
+   sm5 <- summary(m5)

+   simulation5[i,1] <- sm5$coef[1,1] #Intercept estimate
+   simulation5[i,2] <- sm5$coef[2,1] #x_1 coef estimate
+   simulation5[i,3] <- sm5$coef[3,1] #x_2 coef estimate

+   simulation5[i,4] <- sm5$coef[1,2] #standard error for Intercept
+   simulation5[i,5] <- sm5$coef[2,2] #se x_1
+   simulation5[i,6] <- sm5$coef[3,2] #se x_2
+ }
```

That perhaps seemed too complicated, but I wanted to show the MASS and RLab packages for what they offer in creating simulated data. Similar to before I will plot the results:




```
## [1] 2
```

Here we see the three estimates from our simulation and our model's prediction based upon the R `predict()` command. Our estimates are definately off and our predictions seem to be off because of this. Perhaps running a proper logit or probit model will help. We could run the same DGP but with the proper modeling using the `glm()` function, and the proper link function if we so desired.

4.2 Penalized Regression: LASSO, Ridge, Elastic Net

Let's use simulation to test out a few penalized regression models in R, starting with making a new DGP:

```
> set.seed(12345)
> library(caret) #! for train() and Penalized Regression Functions

## Warning:  package 'caret' was built under R version 4.1.3
## Loading required package:  lattice
```

```
##
     ## Attaching package:  'lattice'
## The following object is masked from 'package:faraway':
     ##
     ##     melanoma

> library(MASS) # mvrnorm()
> library(Rlab) # for rbern()
> library(glmnet)

## Warning:  package 'glmnet' was built under R version 4.1.3
## Loaded glmnet 4.1-4

> N <- 1000
> P <- 100
> # Wishart distribution for positive definite matricies
> Sigma <- rWishart(n=1, df=P, Sigma=diag(P))[,,1]
> X <- mvrnorm(N, runif(P, -10, 10), Sigma)
> p <- rbern(P, 0.1)
> b <- p * rnorm(P, 5, 5) + (1-p) * rnorm(P, 0, 0.1)
> e <- rnorm(N, 0, sd=sqrt(10))
> y = X%*%b + e
> m1 <- lm(y~X)
> # Set data frame
> my_data = data.frame(X,y)
```

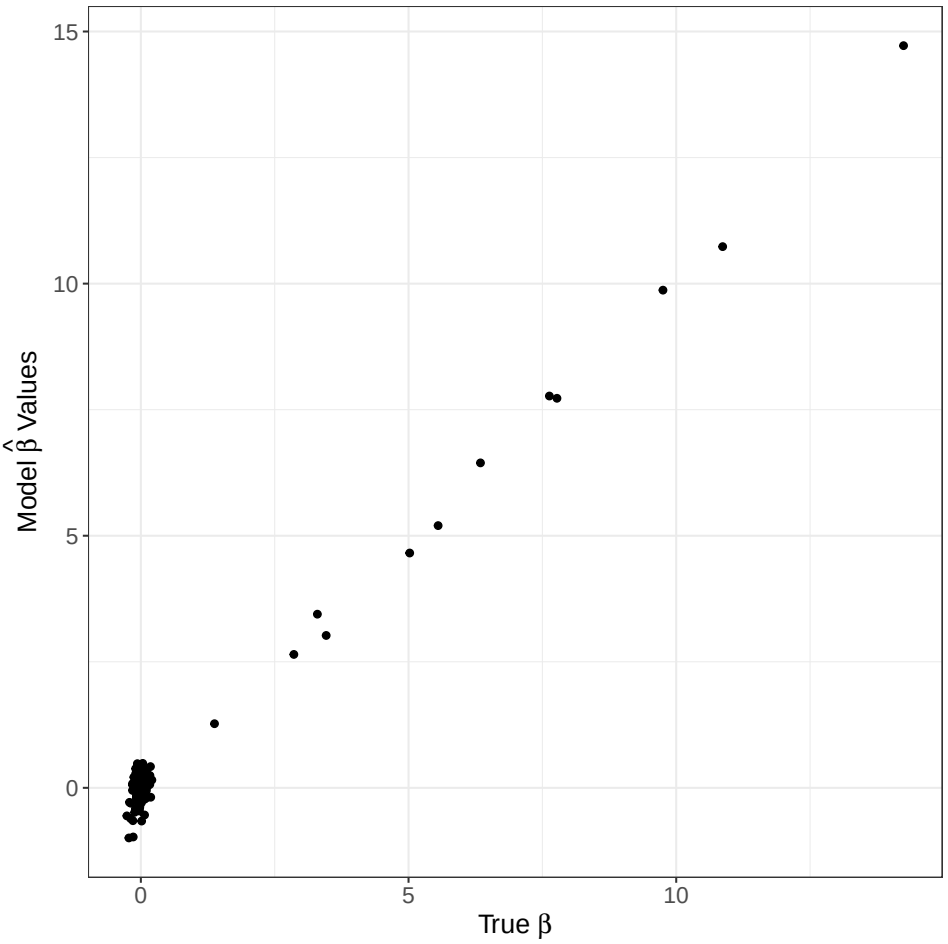
Then we can use a few other functions to make a test set training set:

```
> ## Test Set (20%); Training Set (80%)
> # Get 800 random index from y
> trainidx <- createDataPartition(y, times = 1, p=.80, list=FALSE)
> trainy <- y[trainidx]  # Get traniing set index numbers; sample y's values
> testy <- y[-trainidx]  # Other 20% for test set (y)
> trainx <- X[trainidx,] # same for setting up X variable:
> testx <- X[-trainidx,]
>
> # Create Data Frames for these:
> train_data = data.frame(trainx, trainy)
> test_data = data.frame(testx, testy)
```

The idea here is to do data manipulation and all testing for significance just on a training set, using the test set to come in later to "prove" that everything is good-to-go independent of your training manipulations. It's a great practice, with many ways to do it in R; which will be showcased throughout this chapter. I prefer the `createDataPartition()` function myself.

Let us first plot the correlation between the true β and our model's predicted β:

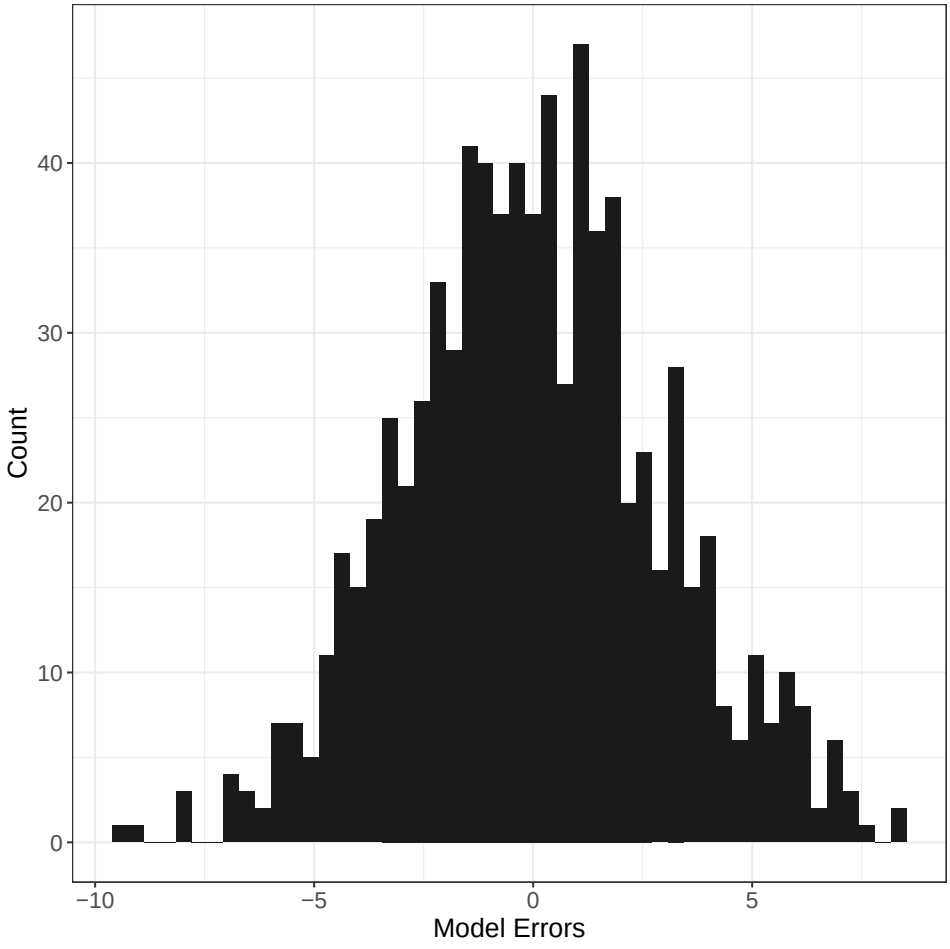
```
> library(ggplot2) # Graphing
>
> qplot(b, coef(m1)[-1])+
+       xlab( expression(paste("True " , beta)))+
+       ylab( expression(paste("Model " , hat(beta), " Values")))+
+       theme_bw()+
+       theme(axis.text=element_text(size=12),
+             axis.title=element_text(size=14,face="bold"))
```



These results make sense, most of our beta's cluster with a mean around zero and some of our beta's are outward with a mean around five and a larger standard deviation. Now, using our training data and leaving the test data aside, let's run the same model within the training data, and examine the errors in a few different ways:

```
> ## Training Model:
> trainmod <- lm(train_data$trainy ~ trainx, data = train_data)
> # summary(trainmod) # Check that this makes sense
>
```

```
> error <- train_data$trainy - predict(trainmod)
> # head(error) # Check this makes sense
>
> # Ideally we want normality here, mean zero
> qplot(error, bins=50, fill=I("grey10"))+
+      xlab( "Model Errors")+
+      ylab( "Count")+
+      theme_bw()+
+      theme(axis.text=element_text(size=12),
+            axis.title=element_text(size=14))
```




```
> ## Root Mean Squred Error
> sqrt(mean(error^2))

## [1] 2.957983

> ## [1] 3.039912
>
```

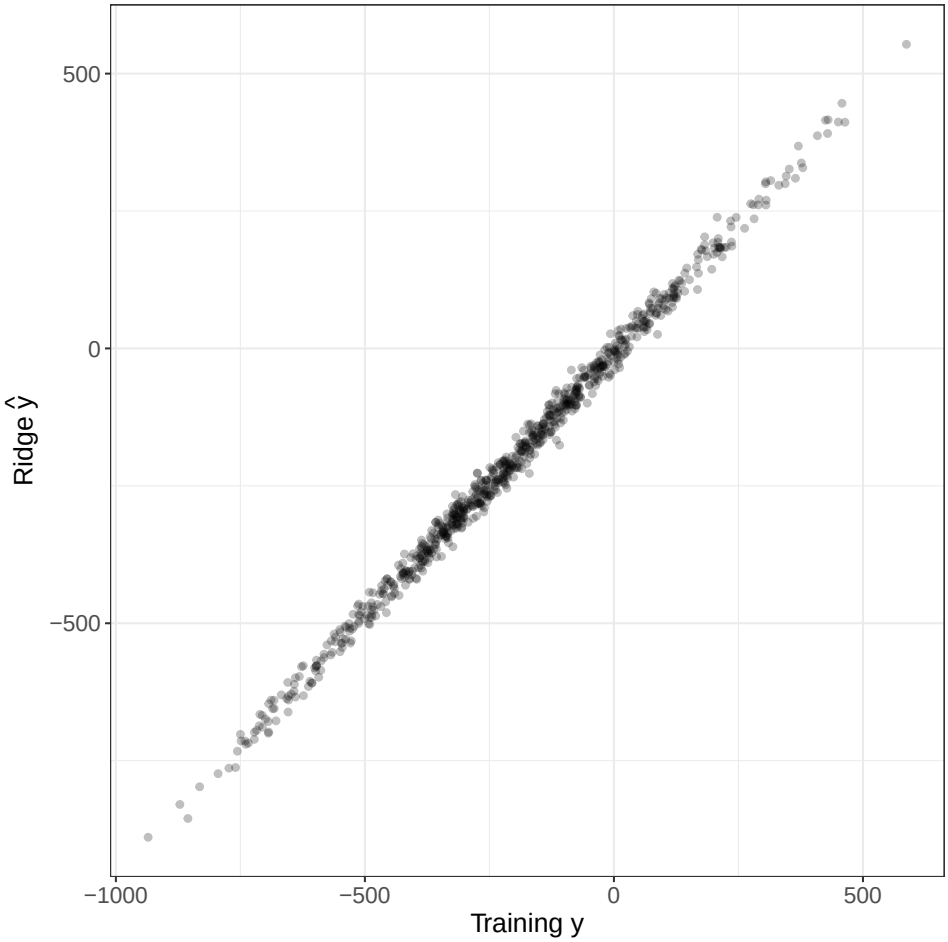
```
> ## Mean Aboslute Error
> mean(abs(error))

## [1] 2.353741

> ## [1] 2.415841
```

Using just our training data with the linear model we see our root mean squared error and mean absolute error are about around 3, pretty low. More intuitively, our errors when plotted are normally distributed with a mean of zero. We can revisit this with a LASSO and Ridge regression to penalize us for having so many parameters for so few observations. Using penalized regression is especially helpful with models that have a large number of predictors relative to our sample size. The "elastic net" is particularly useful because it "switches" between two other penalties (using α), the "LASSO" (Least Absolute Shrinkage and Selection Operator uses a squared transformation) and "Ridge" (using the absolute value of our coefficients), applying both of these in an ever-checking gradient descent down our distribution to the point of least error. Ideally we would like a smooth distribution in this process instead of quick-changing erratic distribution and the elastic net provides the previously mentioned functions to accomplish this via the transformation of our coefficients as they travel farther away from zero.

Let's begin with the LASSO:

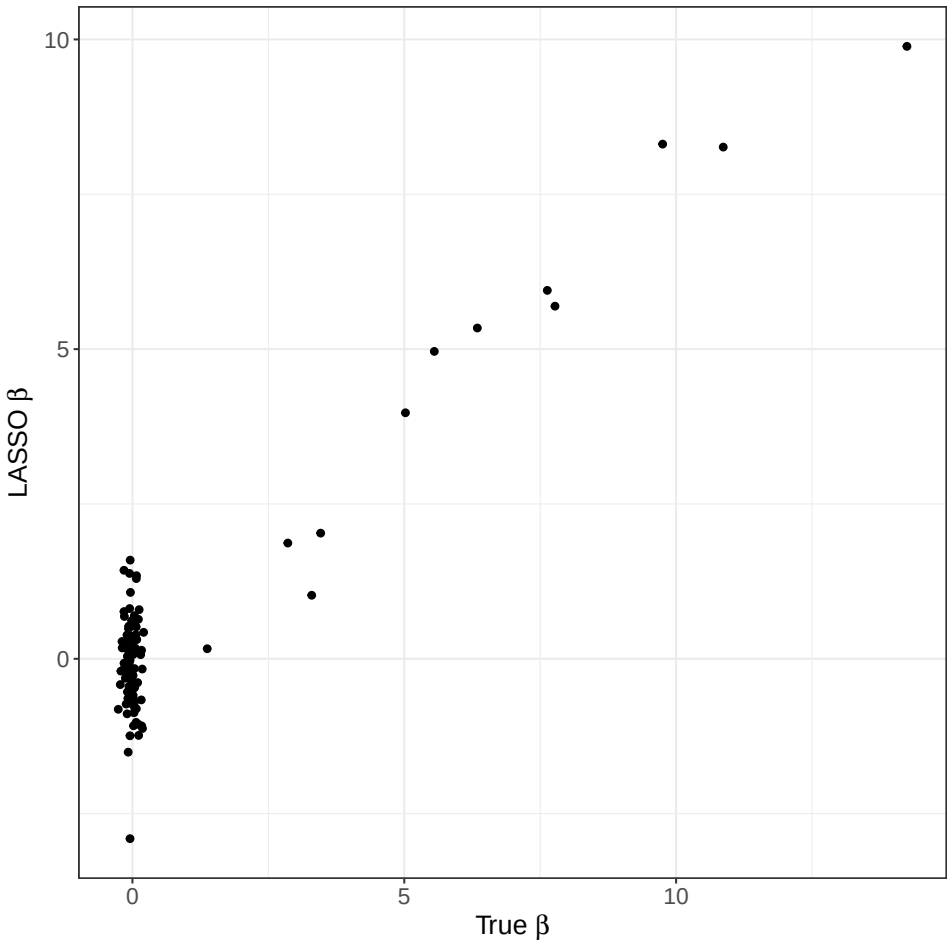
```
> # Remember our training data frame:
> train_data = data.frame(trainx, trainy)
>
> mod_lasso = train(trainy~., method="glmnet",# formula and method
+       tuneGrid=expand.grid(alpha = 0, # alpha=0 is the LASSO
+                 lambda = seq(0,100,1)), #gradient descent to zero
+       data = train_data,
+       preProcess = c("center"),# centering standardization of points
+       trControl = trainControl(method="cv",number=2, search="grid"))
> # "trainControl" sets a resampling method per a "search" parameter grid
> # mod_lasso ## final values were around lambda = 14 using RMSE
>
> ## Narrowing lambda sequence closer to what the lambda reported earlier:
> mod_lasso = train(trainy~., method = "glmnet",
+       tuneGrid = expand.grid(alpha = 0,
+       # *setting bounds closer around 14; smaller "steps" (0.01)
+                           lambda = seq(0,40,0.01)),
+       data = train_data,
+       preProcess = c("center"),
+       trControl = trainControl(method="cv",number=2, search="grid"))
>       # mod_lasso ## Narrowing in, we found a new best lambda of around 14.5!
>
```

```
> ## Predictions:
> yhat = predict(mod_lasso)
> # alpha allows transparent points to see density
> qplot(trainy, yhat, alpha=I(0.25))+
+      xlab( expression(paste("Training ", y)))+
+      ylab( expression(paste("Ridge ", hat(y))))+
+      theme_bw()+
+      theme(axis.text = element_text(size=12),
+            axis.title = element_text(size=14))
```




```
> # best values using best lambda:
> lasso_beta = coef(mod_lasso$finalModel, mod_lasso$bestTune$lambda)
>
> qplot(b, lasso_beta[-1])+ # removing intercept
+      xlab( expression(paste("True ", beta)))+
+      ylab( expression(paste("LASSO ", beta)))+
+      theme_bw()+
```

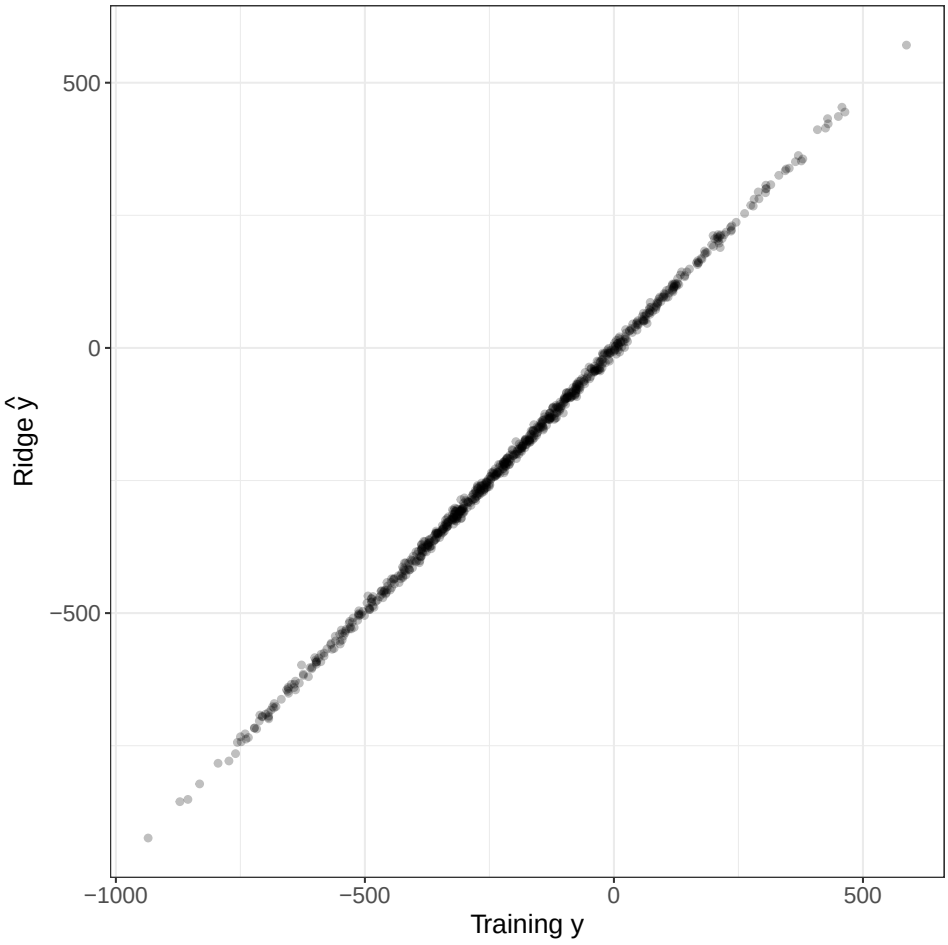
```
+      theme(axis.text = element_text(size=12),
+            axis.title = element_text(size=14))
```



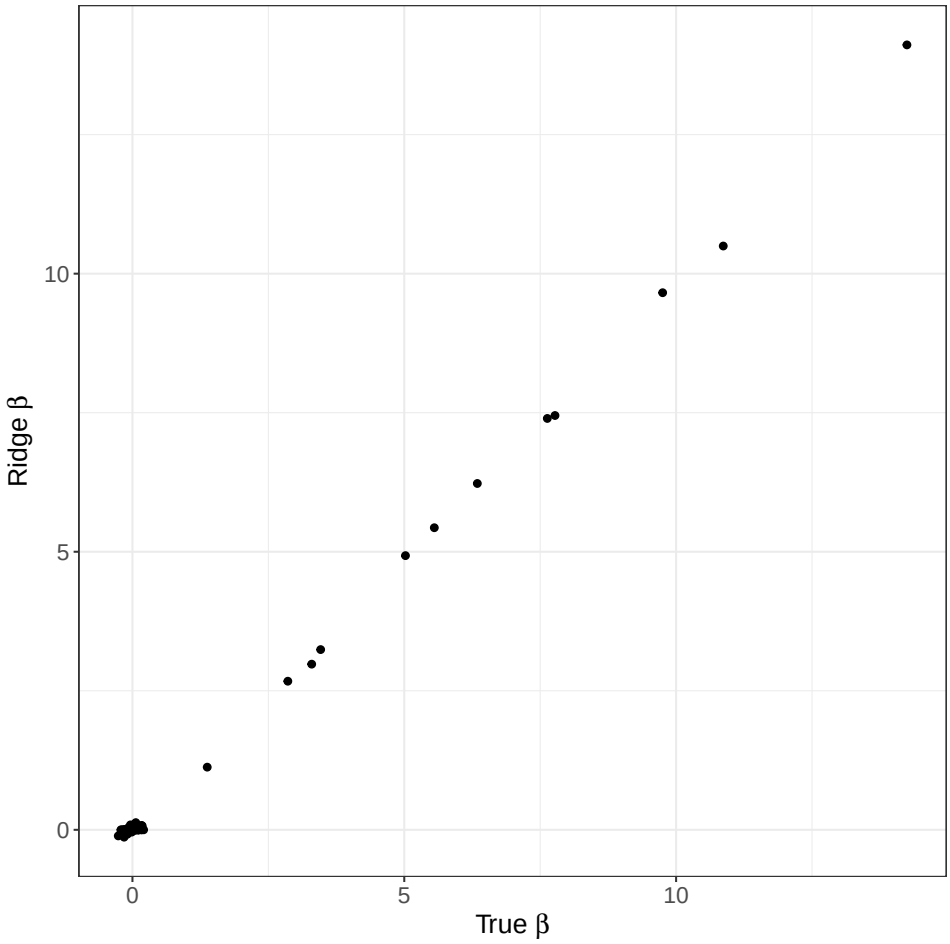
What this tells us is that our predicted values from the LASSO model does well with our training y data. Specifcally, in the first figure we see that our training dependent variable and our penalized LASSO predictions match with each other positively. In the second figure, we see our actual beta's more accurately plotted in conjunction with the LASSO's best fit betas. Similarly, let's see how the Ridge and elastic net's hold up:

```
> ## Running Ridge:
> mod_ridge = train(trainy~., method="glmnet",
+           tuneGrid=expand.grid(alpha=1,
+                lambda=seq(0,10,0.01)),
+           data=train_data,
+           preProcess=c("center"),
+           trControl=trainControl(method="cv",number=2, search="grid"))
> # Best lambda (RMSE) is 1.01
>
```

```
> mod_ridge = train(trainy~., method="glmnet",
+           tuneGrid=expand.grid(alpha=1,
+                lambda=seq(0,3,0.01)),
+           data=train_data,
+           preProcess=c("center"),
+           trControl=trainControl(method="cv",number=2, search="grid"))
>
> # New best lambda of .94
> yhat = predict(mod_ridge)
>
> qplot(trainy, yhat, alpha=I(0.25))+
+      xlab( expression(paste("Training ", y)))+
+      ylab( expression(paste("Ridge ", hat(y))))+
+      theme_bw()+
+      theme(axis.text = element_text(size=12),
+           axis.title = element_text(size=14))
```



```
> ridge_beta = coef(mod_ridge$finalModel, mod_ridge$bestTune$lambda)
>
> qplot(b, ridge_beta[-1])+
+      xlab( expression(paste("True ", beta)))+
+      ylab( expression(paste("Ridge ", beta)))+
+      theme_bw()+
+      theme(axis.text = element_text(size=12),
+            axis.title = element_text(size=14))
```



Our ridge regression shows less variability, our training set y values and ridge predicted values are more tightly linear, and there is less variability in our comparative β values in the plot above. This is interesting and is perhaps explained in the difference between the mathematical formula between Ridge and LASSO themselves (letting λ control the amount of regulation in gradient descent):

Ridge: $(\alpha = 0\ )$

$$\lambda \sum_{j=1}^{p} \left[\frac{1}{2}(1-\alpha)\beta^2 \right] \tag{40}$$

LASSO: ($\alpha = 1$)

$$\lambda \sum_{j=1}^{p} (\alpha) |\beta_j| \tag{41}$$

We seek a set of sparse solutions because we believe that many of our 100 β_j parameters should be zero; a large enough λ will set these exactly or near zero for us! We saw our LASSO λ values become higher than our Ridge λ values; this is because, unlike Ridge, our $\beta_\lambda^{\text{lass}}$ lasso values have no closed form – allowing more variance. A third potential option is to allow our alpha parameter vary between zero and one effectively using both of these penalties and giving us the elastic net (literally just adding LASSO and Ridge).

Elastic Net: $(0 < \alpha < 1)$

$$\lambda \sum_{j=1}^{p} \left[\frac{1}{2}(1-\alpha)\beta^2 \right] + \lambda \sum_{j=1}^{p} (\alpha) |\beta_j| \tag{42}$$

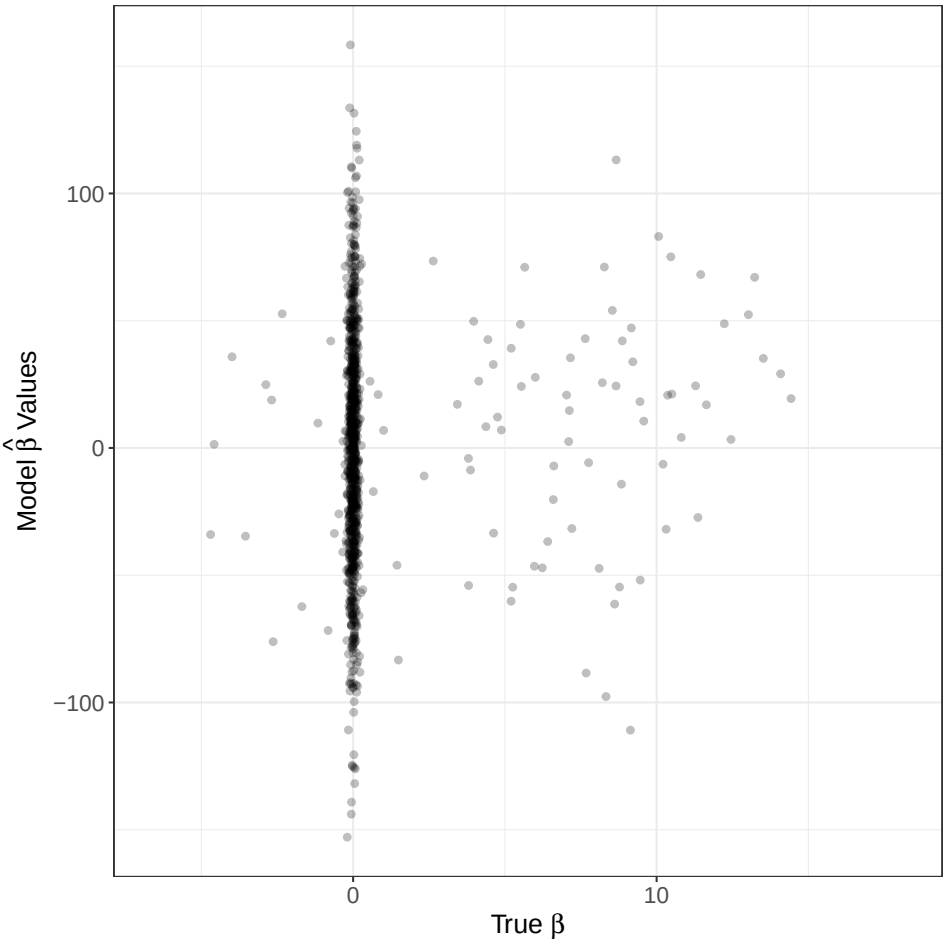
```
> ## Elastic Net:
> mod_enet = train(trainy~., method = "glmnet",
+              tuneGrid = expand.grid(alpha = seq(0, 1, 0.1),
+                                     lambda = seq(0, 100, 1)),
+              data = train_data,
+              preProcess = c("center"),
+              trControl = trainControl(method="cv", number=2, search="grid"))
> # alpha used was .9 and lambda ended up being 1.
```

This elastic net will give us similarlly better results with a varrying α along with it. However, these models work best for when we have data with many parameters, here we have only started with 100 ("only"). Let's ramp up the number of perameters (P) and really put these models to the test. I'll report the DGP in full.

```
> N <- 1000
> P <- 1500 # Note this change
> Sigma <- rWishart(n=1, df=P, Sigma=diag(P))[,,1]
> X <- mvrnorm(N, runif(P, -10, 10), Sigma)
> p <- rbern(P, 0.1)
> b <- p * rnorm(P, 5, 5) + (1-p) * rnorm(P, 0, 0.1)
> e <- rnorm(N, 0, sd=sqrt(10))
> y = X%*%b + e
> m1 <- lm(y~X)
> my_data = data.frame(X,y)
```

With this DGP, let's see how our regular linear model holds up graphically:

```
> ## Problematic:
> qplot(b, coef(m1)[-1], alpha=I(0.25))+
+      xlab( expression(paste("True " , beta)))+
+      ylab( expression(paste("Model " , hat(beta), " Values")))+
+      theme_bw()+
+      theme(axis.text = element_text(size=12),
+           axis.title = element_text(size=14))

## Warning:  Removed 501 rows containing missing values (geom_point).
```



We see quite a bit of noise, also our results will be outrageously significant because we are not penalizing ourselves for multiple parameters unless we are using the AIC and BIC. Now let's see if the elastic net does any better given the larger number of parameters. We'll start by chopping up the data into a testing set and a training set.

```
> # Test Set; Training Set (20% and 80% again)
> trainidx <- createDataPartition(y, times = 1, p=.80, list=FALSE)
>
> trainy <- y[trainidx]
```

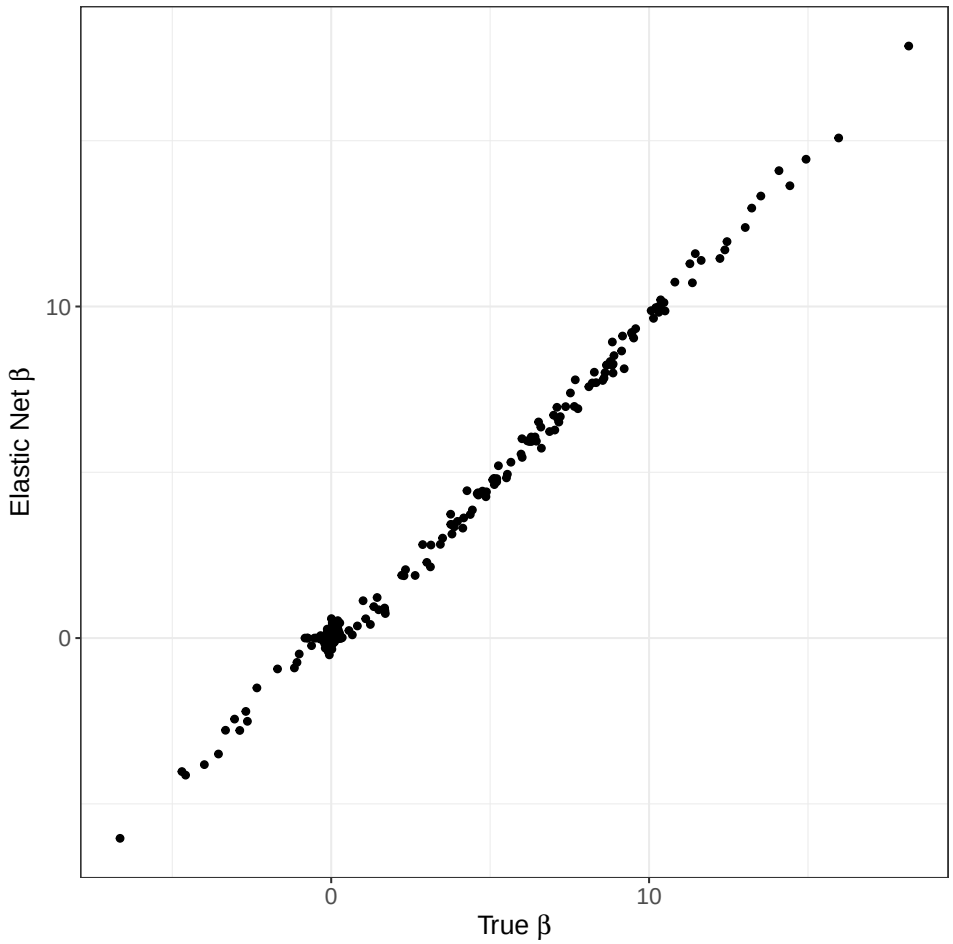
```
> testy  <- y[-trainidx]
> trainx <- X[trainidx,]
> testx  <- X[-trainidx,]
>
> train_data = data.frame(trainx, trainy)
> test_data  = data.frame(testx, testy)
>
> # Training linear Model:
> trainmod <- lm(train_data$trainy ~ trainx, data = train_data)
```

Running the elastic net now looking for the best lambda.

```
> # Using Penalized Regression Helps Conceptualize This:
> # Elastic Net:
> mod_enet = train(trainy~., method = "glmnet",
+           tuneGrid = expand.grid(alpha = seq(0,1,0.1),
+                                  lambda = seq(0,100,1)),
+           data = train_data,
+           preProcess = c("center"),
+           trControl = trainControl(method="cv",number=2, search="grid"))
> # alpha used was 1 and lambda ended up being 8.
>
> mod_enet = train(trainy~., method = "glmnet",
+           tuneGrid=expand.grid(alpha = seq(0.6,1,0.5), #narrowing
+                                lambda = seq(0,10,0.5)),
+           data = train_data,
+           preProcess = c("center"),
+           trControl = trainControl(method="cv",number=2, search="grid"))
> # alpha ended up at .6 and lambda at 10.
```

Great!

```
> enet_beta = coef(mod_enet$finalModel, mod_enet$bestTune$lambda)
> qplot(b, enet_beta[-1])+
+      xlab( expression(paste("True ", beta)))+
+      ylab( expression(paste("Elastic Net ", beta)))+
+      theme_bw()+
+      theme(axis.text = element_text(size=12),
+           axis.title = element_text(size=14))
```



Setting up a standard linear model with more estimators than number of observations is a fundamental assumption violation; it over-inflates our coefficients and gives us essentially no error. This can be really misleading to readers, and similar to how one might penalize their general linear models utilizing AIC and BIC we can use the elastic net to show more variance in our estimates and report our λ values which are much larger than before ($\lambda = 10$). Penalized regression is the intellectually honest thing to do when you have a model with a large number of parameters, the elastic net in this case has showed some uncertainty in our estimates whereas the normal model inflated our certainty (reported an R^2 of 1).

4.3 Support Vector Models, Regression Trees

Support Vector Models (SVMs) can be used to analyze data with the least error given unsupervised machine learning. Many of these models require super computing, or a careful consideration of the data and the underlying calculus going on. Consider the following DGP: balanced data, split into a test set and training set.

```
> # Need a lot of packages for this:
> packages <- c("caret", "MASS", "Rlab", "boot", "ggplot2", "ggridges", "dplyr")
> invisible( lapply(packages, library, character.only = TRUE))
```

```
>
> set.seed(12345)
>
> N <- 2000
> P <- 50 # P is cut down for computation time, (but could be much larger!)
>
> ## Ballanced DGP creation
> mu <- runif(P, -1,1)
> Sigma <- rWishart(n=1, df=P, Sigma=diag(P))[,,1]
> Sigma <- ifelse(row(Sigma) != col(Sigma), 0, Sigma)
> X <- mvrnorm(N, mu=mu, Sigma = Sigma)
> p <- rbern(P, 1)
> beta <- p*rnorm(P,1,0.9) + (1-p)*rnorm(P,0,0.3)
> eta <- X%*%beta
> pi <- inv.logit(eta)
> Y <- rbern(N, pi)
> Y <- as.factor(Y)
> data.1.Bal <- data.frame(X, Y)
>
> ## Test/Training Sets
> test.data.1.Bal <- data.1.Bal[1:500,]
> train.data.1.Bal <- data.1.Bal[501:2000,]
```

With the help of a few handy packages that help users run SVM's; consider the following:

```
> library(kernlab)
> library(microbenchmark)
> # Linear SVM
> linear.svm.bal <- train(Y~.,
+                         data=train.data.1.Bal,
+                         method = "svmLinear")
>
> linear.svm.bal$results$Accuracy

## [1] 0.9677524

> # A polynomial kernal: (long computation!)
> # poly.svm.bal <- train(Y~.,
> #                       data=train.data.1.Bal,
> #                       method="svmPoly")
>
> # poly.svm.bal$results[as.numeric(rownames(poly.svm.bal$bestTune)),]
> # .951 Accuracy; .902 Kappa; AccuracySD of .011
>
```

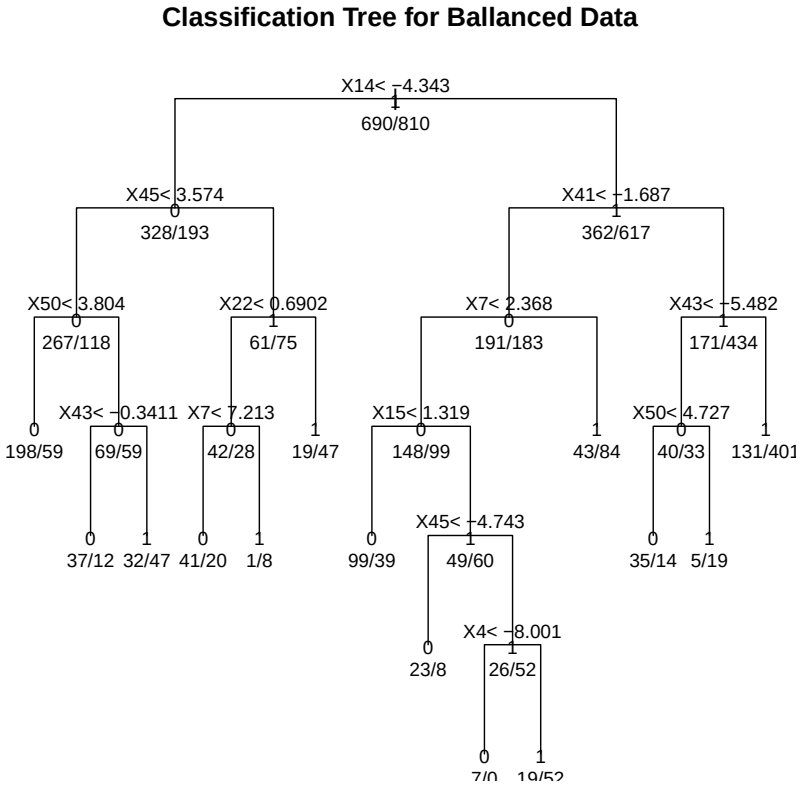
```
>
> # Radial kernal:
> # radial.svm.bal = train(Y~.,
> #                        data=train.data.1.Bal,
> #                        method="svmRadialCost")
>
> # radial.svm.bal$results[as.numeric(rownames(radial.svm.bal$bestTune)),]
> # Accuracy of .54; AccuracySD .017
```

Dangerously simple, this is how an SVM is ran, and could be ran given the need for a polynomial or radial kernel. The microbenchmark package is included above for any timing related concerns conducting research with more advanced kernels.

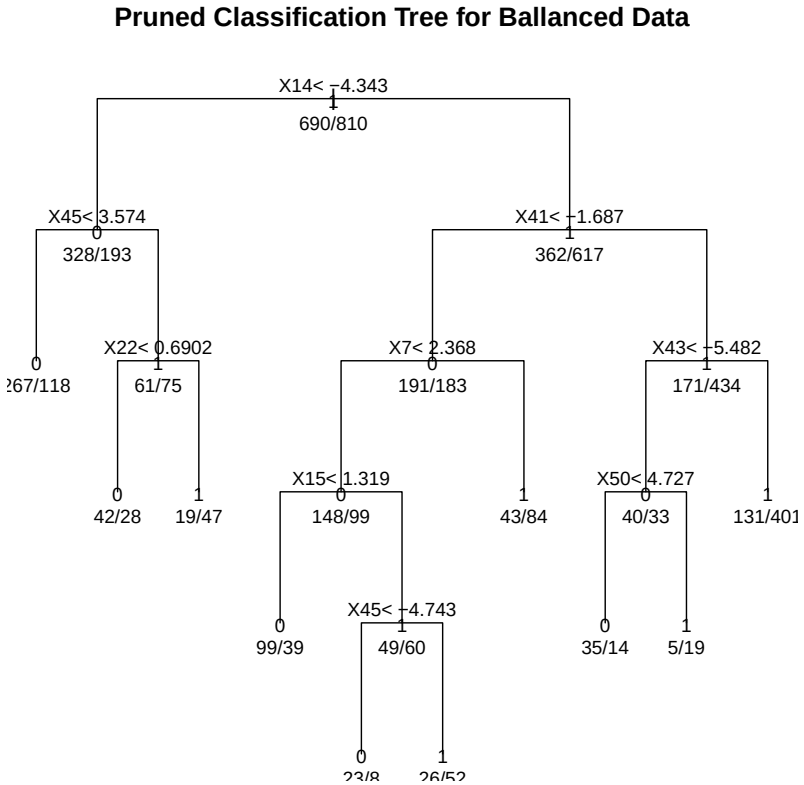
So what's going on exactly? Well, an SVM classifies data "better" by calculating the distances between points and a line (or plane) drawn through the data. The minimal errors between these points and the plane generates more reliable classification models. This is different from a standard regression least-squared-errors approach because this uses Euclidean distance, which is different from a vertically measured residual (commonly seen in linear models). Beyond this, we can specify various kernels to attempt to classify our data throughout various dimensions – something the standard linear model would struggle to accomplish. Overall our data seem to classify about the same as the elastic net for balanced data.

Another option, however, is to use classification trees for visualization and confirmation of results paired with other machine learning techniques. Below is code to accomplish this if you so choose:

```
> # Using "rpart", may need to reload R to get things to work.
> library(rpart)
> # grow tree with same DGP as above.
> tree.bal <- rpart(Y~.,
+                   method = "class",
+                   data = train.data.1.Bal)
> ## other options include:
> # printcp(tree.bal) # display the results
> # plotcp(tree.bal)  # visualize cross-validation results
> # summary(tree.bal) # detailed summary of splits
>
> # plot tree
> plot(tree.bal, uniform = TRUE,
+      main = "Classification Tree for Ballanced Data")
> text(tree.bal, use.n = TRUE, all = TRUE, cex = .8)
```




```
> # then we have a lot of options off of this, including pruning the tree back,
> # or making -multiple- trees into a "forest" and
> # selecting the best one! (intutively a form of advanced bootstrapping!)
> # prune the tree, by the error that was previously generated above
>
> # prune function:
> ptree.bal <- prune(tree.bal,
+                    cp=tree.bal$cptable[which.min(tree.bal$cptable[,"xerror"]),
+                                        "CP"])
> # plot the pruned tree
> plot(ptree.bal, uniform = TRUE,
+      main ="Pruned Classification Tree for Ballanced Data")
> text(ptree.bal, use.n = TRUE, all = TRUE, cex = .8)
```




```
> # can make a postscript chart (good for publishing) using post():
> # post(ptree.bal,
> #     title = "Pruned Classification Tree for Ballanced Data")
```

Overall, regression trees (here they are classification trees) can be used to see which predictors lead our classifications, and similar to how elastic nets will bring some predictors to zero; we can then "prune" these observations and vizualize this process better. In poorly predictive data, this will become well pruned to only a few key observations. Reporting these over other methods may be more intuitive for readers and reviewers!

Those familiar to machine learning will note the similarity between these models to principle components analysis (PCA) – because it is very close.

4.4 K Nearest Neighbors, Nearest Means

5 Time Series

The careful analysis of data over time is classified as "time series" analysis. Data projections, forecasting, historical analysis, and trend analysis all describe timeseries science. Much of this section on timeseries comes from a course and associated text from the course I took.[12] I would highly recommend this text if a reader is serious about learning timeseries analysis, sections of the book will be referenced in this section of the document.

5.1 Data Concerns

Time series does not rid itself of serious data concerns, and actually new data concerns arise with time series data. First, one should be aware of "data vintaging": or where data is not finalized and there is some recency bias in reporting. For example, government data or live polling data may not be finalized or the process of collecting data is not complete. Older data is less controversial (usually!) but recent data can be more debatable. Robertson and Tallman have a 1998 Economic Review article discussing this in more detail. The lesson to learn here is that careful data collection, and taking the proper amount of time to truly understand the underlying data generation process of that data is important before beginning time series analysis.

Another potential concern to take seriously is systematic sampling and temporal aggregation. This is where regular interval (systematic sampling) occurs but not for the entire series. Naively a researcher may aggregate across the cross-sections but further problems arise if the reason why the data is missing relates to the variables in the data (think back SUTVA in the first chapter). For example, government data missing a few years around a particular scandal or tragedy. Further, we might consider what the natural time-unit of data really is in the first place. What is the naturally occurring time of a vote, of opinions, or of healthcare? Is a vaccine or survey best distributed by the year or by the month? This largely depends on the research question and we may not always have the ability to best measure our natural time scale. This creating noise or missing shocks where they may actually exist.

Finally, less of a data concern but rather a justification of the methods to follow when one might consider just adding a lagged time variable within a regression model. Doing this does not solve exogenous problems (there's a Journal of Politics article related to this) and we will just end up with an extra function that calculates the bias from aggregation. Further, if there are multiple frequencies, we could instead just measure those with a variation of the VAR (upcoming section). Adding time into a regular regression is very restrictive when we could use a VAR to pick up lag lengths or a simultaneous regression model built for such variables.

When preparing your data for analysis considering the above and take steps to make sure the data has a proper sampling methodology, includes a systemic time point in which measures were taken, the data is accurately defined the same throughout the series, and take concern for other common data concerns pre-modeling (see chapter 1).

---

[12] Box-Steffensmeier, Janet M., John R. Freeman, Matthew P. Hitt, and Jon CW Pevehouse. Time series analysis for the social sciences. *Cambridge University Press*, 2014.

5.2 Differencing Equations and Operators

Much like mothers all around the world demanding their children eat vegetables, methodologists too have stressed the importance of learning differencing equations for time series analysis.

Much like functions in R, or mathematical operators, differencing equations too could be called a usages of various operators. For simplicity I will start by calling them differencing operators or a variant of the "backshift operators." These are functions in the same way that $f(x)$ is a function being applied to x. Instead, with the differencing operators (Δ) we mean, simply, $(1 - B)$ where B is just a lag operator (or the backshift operator). The "difference" synonym comes from the subtraction of $1 - B$.

For clarity, these two are equivalent:

$$\Delta(.5^t) = .5^t - .5^{t-1} \tag{43}$$

And the backshift operator acts simply:

$$B(.5^t) = .5^{t-1} \tag{44}$$

And the square of this function works similarly:

$$\Delta^2 y(t) = (1 - B)^2 y(t) = (1 - 2B + B^2)y(t) = y(t) - 2y(t - 1) + y(t - 2). \tag{45}$$

Or with $.5^t$ as $y(t)$ we have:

$$\Delta^2(.5^t) = .5^t - 2(.5^{t-1}) + .5^{t-2} = .5^t \tag{46}$$

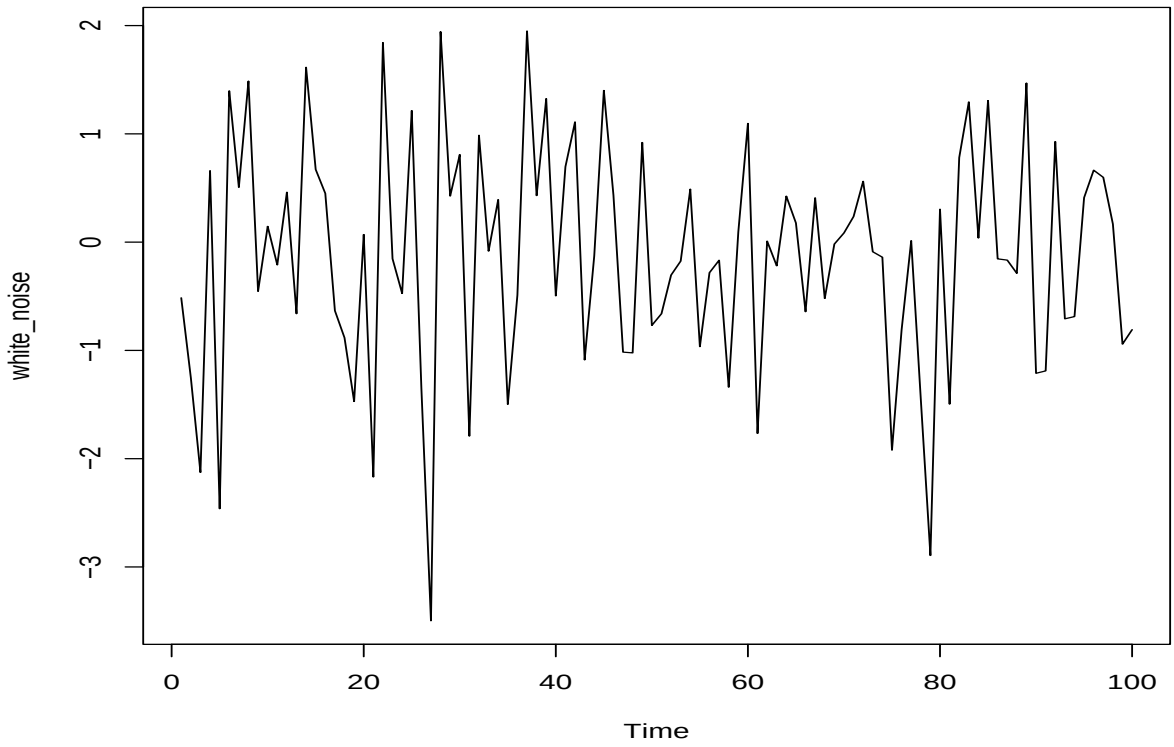
Finally, we have a third operator to help us and that is the "expectations" or "lead" operator (E). Which brings our variable forward a time period.

$$Ey(t) = y(t + 1) \tag{47}$$

Difference equations, then, are the integrate combination of these operators to produce particular equalities. Standard regression models often do not consider these nuances, but the study of difference equations helps us understand the dynamics that are implicit in our time series related equations.

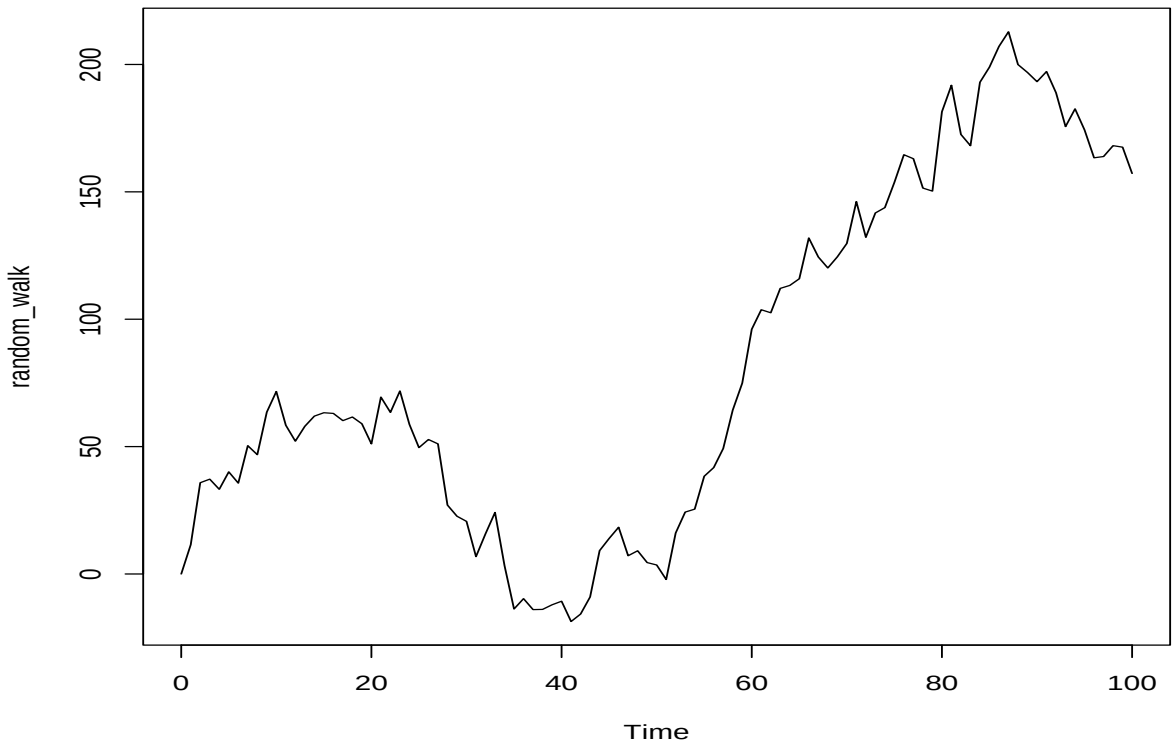
To show this in greater detail (and far less math), let's just start with a white noise model in R. A white noise (WN) model is a basic time series model, and it's sort of the basis of most models time series analysts consider. White noise is defined as a completely random predictive trend, typically centered around a static mean line. In the example below, we'll use a base-R ARIMA (or "autoregressive integrated moving average) simulation and plot for timeseries classed data. ARIMAs and the generation of timeseries data will be discussed further in a following section in this chapter. For now, let's just visualize a WN model which has one hundred observations and is independent and identically distributed (iid).

```
> # Stats package loaded by default base-R
> white_noise <- arima.sim(model = list(order = c(0, 0, 0)), n = 100)
>
> ts.plot(white_noise)
```



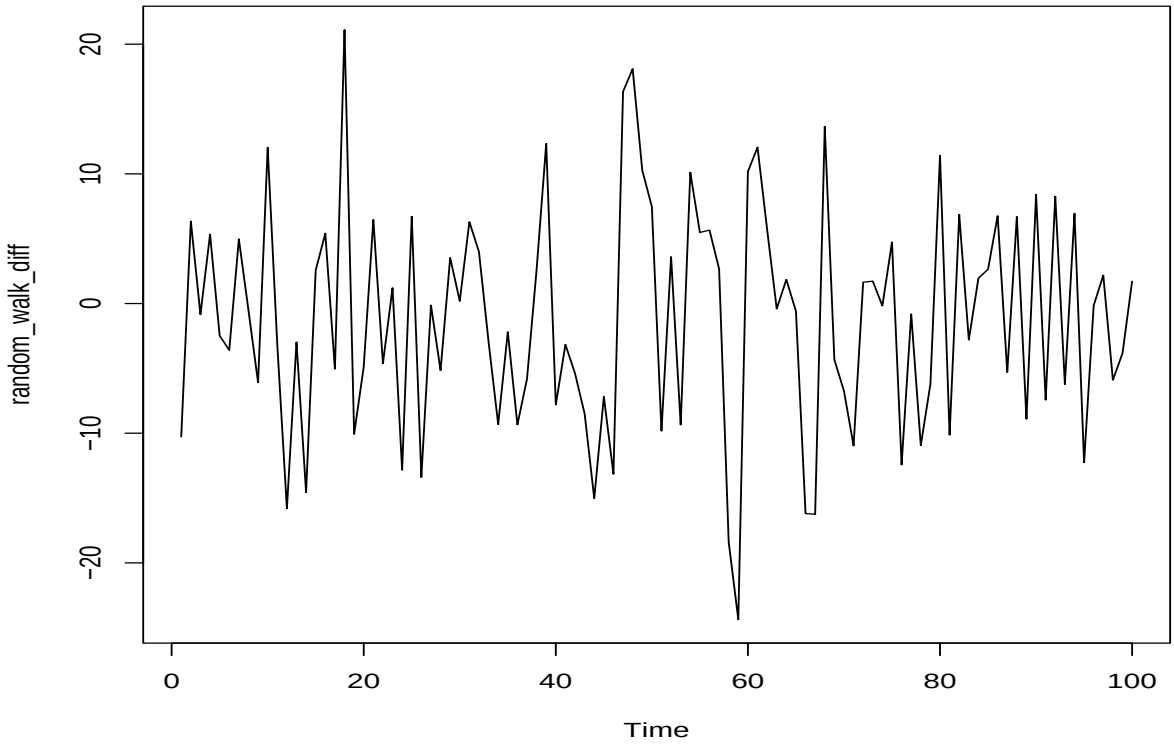
Next, let us talk about the random walk (RW) model - another basic time series model. It is a cumulative sum (or integration) of a mean zero white noise (WN) series. In other words, if we took the first difference of RW series we would end with a WN series. To simulate this model, we need to change the ARIMA model specification to adjust the integration argument from zero to one. This happens to be the middle ordered command in the model argument (shown below). For this series I will also show that we can adjust the standard deviation for the series. Adjusting the mean of the series adds "drift" or a intercept to the RW model, which corresponds to the slope of the RW time trend.

```
> # Stats package loaded by default base-R
> random_walk <- arima.sim(model = list(order = c(0, 1, 0)), n = 100, sd = 10)
>
> ts.plot(random_walk)
```



Now, let's take the same model and difference it in R, showing us its underlying WN model series.

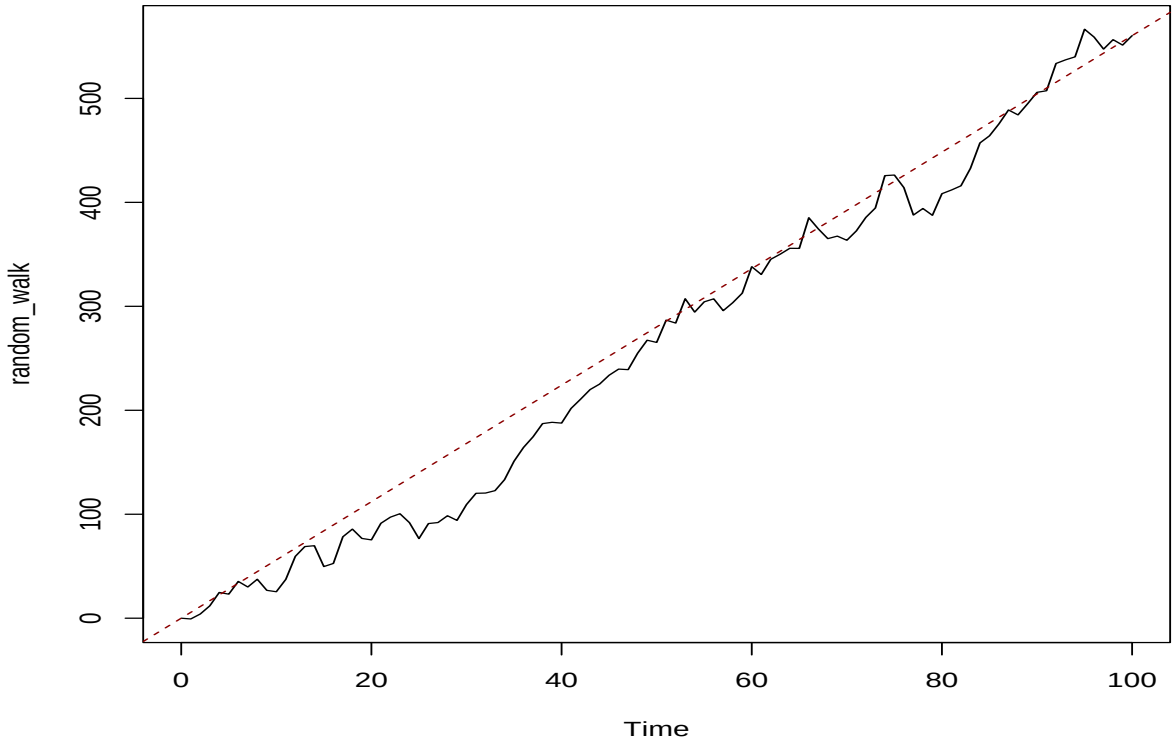
```
> random_walk <- arima.sim(model = list(order = c(0, 1, 0)), n = 100, sd = 10)
>
> random_walk_diff <- diff(random_walk)
>
> ts.plot(random_walk_diff)
```



How we could use both of these models is widely varied, but let's generate a time series y and suppose it is something we are interested in. We can fit a random walk model with a drift (for example) by first differencing the data, fitting the resulting WN model to the differenced data using the ARIMA function in R with the right order arguments (all zeros for WN). This ARIMA command in R will then output model information that can be interpreted for our given series y (in this example a RW with a drift). Under coefficients there is the estimated drift variable (Intercept). Its approximate standared error is provided directly below it and the variance of the WN part of the model is also estimated under the label σ^2!

```
> # RW data with a drift (y) to fit WN model to:
> random_walk <- arima.sim(model = list(order = c(0, 1, 0)),
+                          n = 100, sd = 10, mean = 5)
>
> # Create white noise from our data:
> rw_diff <- diff(random_walk)
>
> # ARIMA model the differenced data with the same white noise process:
> model_wn <- arima(rw_diff, order = c(0, 0, 0))
```

```
>
> # let's grab our intercept and plot:
> int_wn <- model_wn$coef # get intercept from model results
> ts.plot(random_walk)    # plot original data
> abline(0, int_wn,       # display model estimate onto graph
+        col = "darkred", lty = 2)
```



You can imaging modeling out a random walk process where the data is entirely simulated and comparing model output from a white noise process to your real data - displaying that your data does not behave randomly but has some sort of systemic process or "something else going on" that cannot be explained with randomness like the simulated data may be.

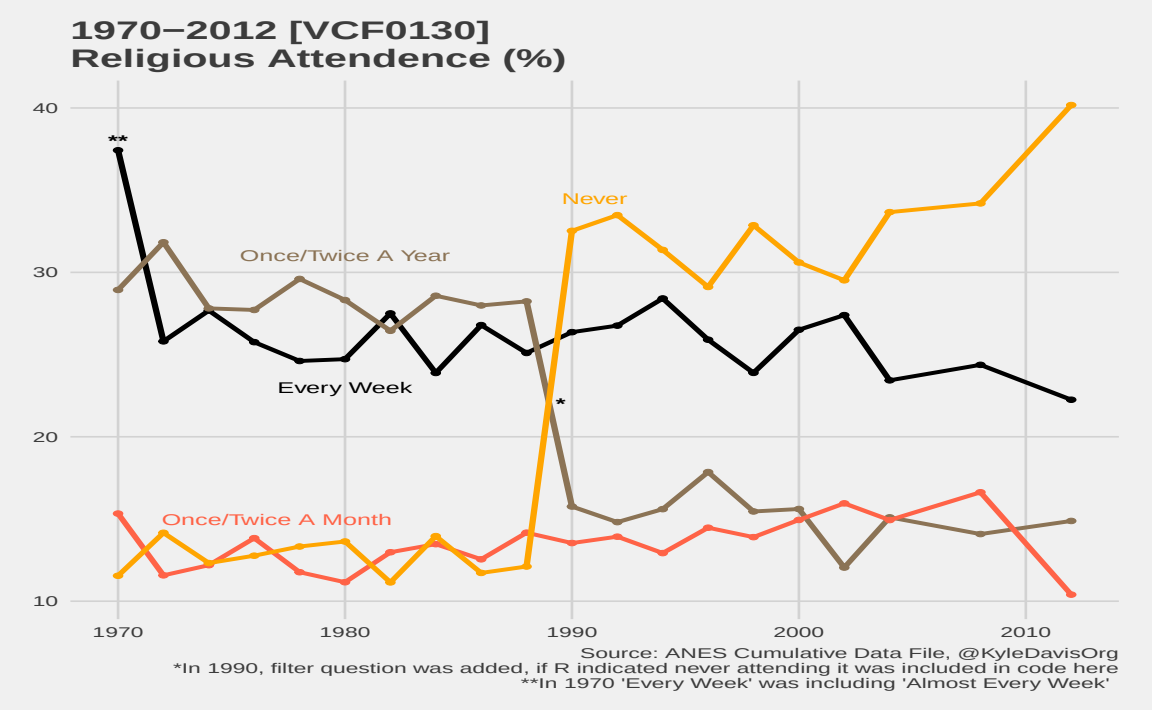
5.3 Identifying AR;MA - ACF, PACF

5.4 Unit Roots

5.5 ARIMA

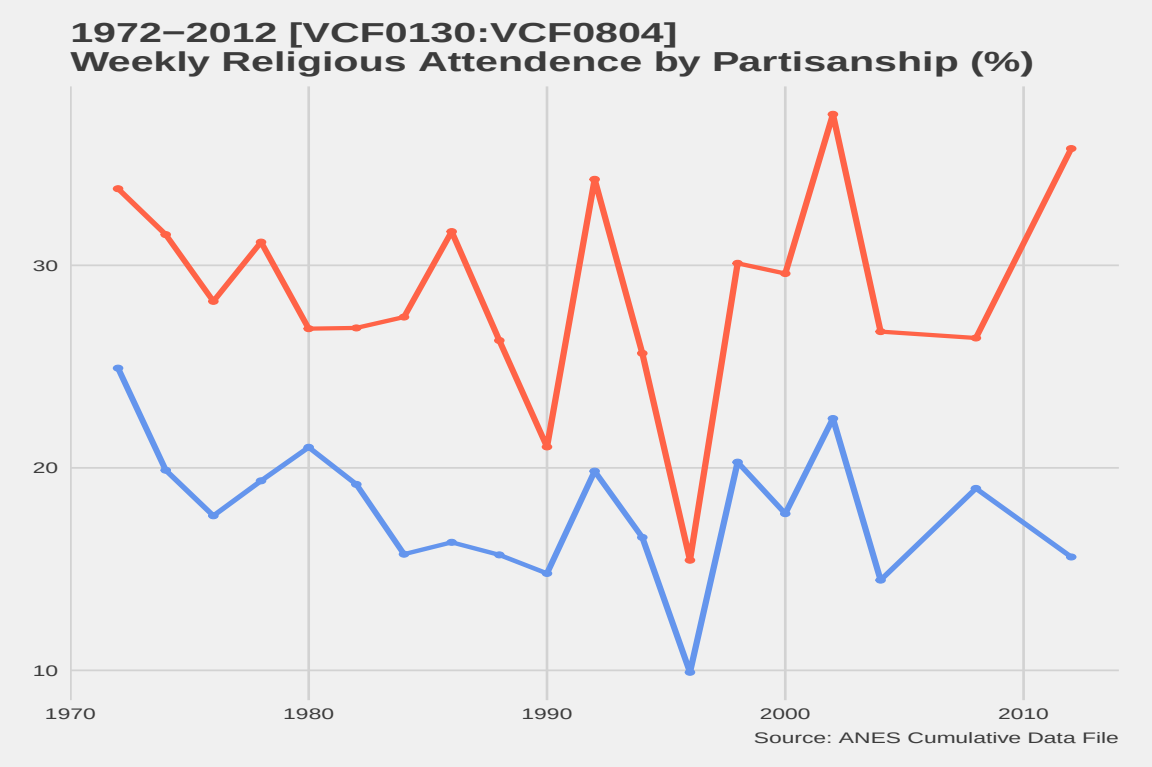
5.6 VAR

5.6.1 Example VAR



Using ANES data[13], we see some survey changes over time on how we measured religious attendance. To narrow down, I will be analyzing only those respondents that reported weekly attendance. Further, I will be looking at those who reported being "strongly liberal" (conservative) including leaning liberal (conservative). Since the latter question was only asked from 1972 on, I will subset the weekly church attendance from 1972 to 2012. This has an unexpected benefit of removing the 1970 survey change on weekly church attendance.

[13]American National Election Survey, due to size this could not be uploaded to GitHub. While this information is readily available online I think it required academic licensing and authorization. Feel free to reach out to me for this (now old) data if interested in replication.



Using this data, I wish to formally test that the two series are not Granger-causal of one another and that the two trends are not deterministic of the other. It is my hypothesis that the two series do not rely on one another, but larger economic and sociopolitical influences are at play. Following testing this hypothesis, I will end this assignment discussing future directions.

Testing for Optimal Lag Order for VAR

Before I begin testing for optimal lag length for the model, I test for the presence of unit roots. The following chunk of R code goes through setting up the data and conducting unit root tests. It reveals that in every unit root test I conduct, there is evidence against the presence of unit roots and nonstationarity.[14]

```
> libweekTS <- ts(data = libweek[,2],
+                 start = head(year, n = 1),
+                 end = 2008, # 2008 end here because ANES aregular surveys
+                 frequency = .5)
> # plot(libweekTS) # Checking
>
> consweekTS <- ts(data = consweek[,2],
+                 start = head(year, n = 1),
```

[14]In setting up the time series in R, I had to set the end date back, to get the right frequency to line up with skipped year surveys. Regardless, all data is still present, the year index label for 2012 is, however, masked. This should not change any results, I checked the plots between the full ANES dat and the time series object reported here for any discrepencies, finding none.

```
+                 end = 2008,
+                 frequency = .5)
> # plot(consweekTS) # Checking
>
> ## Unit root testing:
> library(tseries)
>  adf.test(libweekTS)
>  adf.test(consweekTS)
> # Reject the null of non - stationarity and a unit root
>
>  kpss.test(libweekTS)
>  kpss.test(consweekTS)
> # Reject the null of stationarity around a deterministic trend
>
>  PP.test(libweekTS)
>  PP.test(consweekTS)
> # Reject the null that the series is integrated of order 1 (unit root)
```

Following these tests I feel confident that there are no unit roots. Moving to testing the lag-order of the model is realitively easy to do in R, and give us the lag order based upon different information criteria such as the AIC, HQ, SC, and FPE. Results of these tests all indicate best lag-length of four.

```
> # Ideal lag order:
> varData <- data.frame(libweekTS, consweekTS)
>
> library(vars)
>
> lagselection <- VARselect(varData, lag.max = 10, type = "none")
> lagselection$selection

## AIC(n)  HQ(n)  SC(n) FPE(n)
##      4      4      4      4
```

Granger–Causality Test

A time series is said to be Granger-causal to an outcome if it (through lagged values) can provide statistically significant information about the outcome. This relationship is based upon two requirements: the cause happens prior to its effect (lagged values), and the cause provides unique information about the future values of its effect. These assumptions are in some-way built into nearly every cause inference test.

More specifically, a variable X is Granger-causal to some Y if the prediction of Y_t based upon all of its past lagged values $(Y_{t-1}, Y_{t-2}, ...)$ and on lagged values of X does better than just Y based solely on it's past values alone. Put another way, Granger-causality compares two predictive models ("worlds"), one where X exists or where Y is soley predictive upon itself. If the model which includes X does better than the Y being predictive of itself, X is said to be Granger-causal of Y.

```
> var1 <- VAR(varData, p=4, type="const") # Indicating lag length 4 here.
>  # summary(var1) VAR results, including a 4-lag model results.
>  # plot(var1) # can easily check ACF and PACF here
>
>  granger1 <- causality(var1, cause="consweekTS")
>  granger1$Granger$method

## [1] "Granger causality H0: consweekTS do not Granger-cause libweekTS"

>  granger1$Granger$p.value

##           [,1]
## [1,] 0.3452188

>  granger2 <- causality(var1, cause="libweekTS")
>  granger2$Granger$method

## [1] "Granger causality H0: libweekTS do not Granger-cause consweekTS"

>  granger2$Granger$p.value

##           [,1]
## [1,] 0.9601701
```

This hypothesis test takes as the null that X is not a Granger-cause of Y , and as such, a p-value below some α level would lead one to reject the null hypothesis. These tests lead me to confirm that neither series is Granger-causal of the other. Placed back in theory, neither liberals nor conservatives follow each others trends in weekly church attendance – indicating other sociological, political, or economic forces to be at play.

A sign-and-significance reading of the Granger causality results conclude that, with 12 degrees of freedom, neither variation of the Granger-causality test pass statistical significance to declare Granger-causality. The liberal timeseries onto the conservative series holds a p-value of .96, and the conservative series on the liberal holds a p-value of .34. Neither pass conventional .05 thresholds because the series better explains itself than other explains it with respect to the VAR with lag length four.

Innovation Accounting

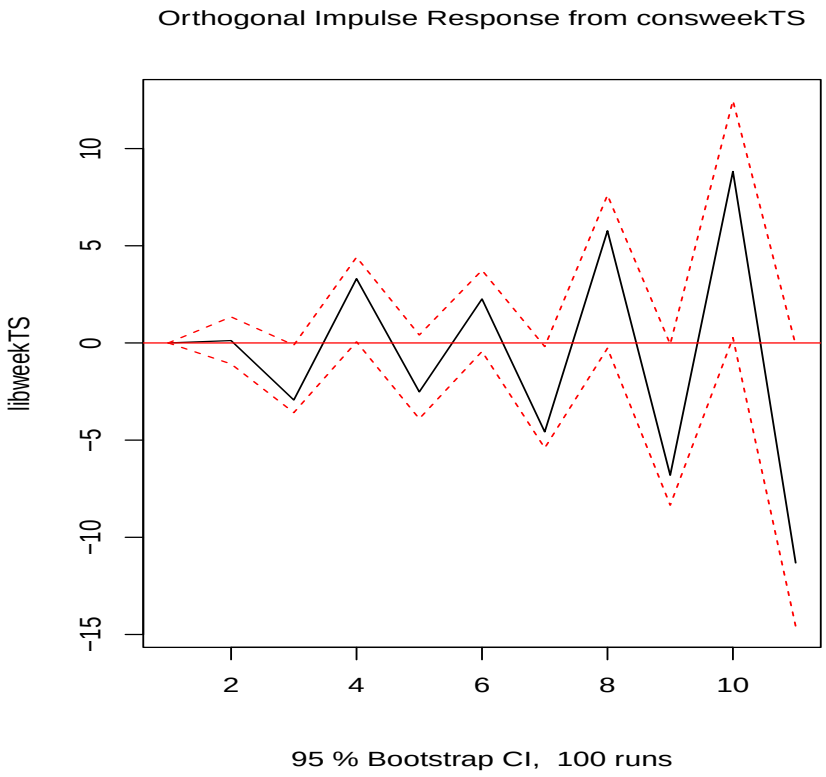
Innovation accounting can tell us how a VAR model responds to individual and orthogonalized shocks to each of the variables. This necessitates an assumption that our variable orderings matter as these shocks transition through our structure. However, orthogonalizing the equation allows for us to see the marginal effects, if we didn't do this we would allow the variable to increase its ripples through the structure without controlling for confounds that could occur.

In this particular case I am not sure if the ordering of variables really matters much, I think respondents from year-to-year don't really consider their parties past responses to church identification all that much. The figure below kind-of shows this, a moving around

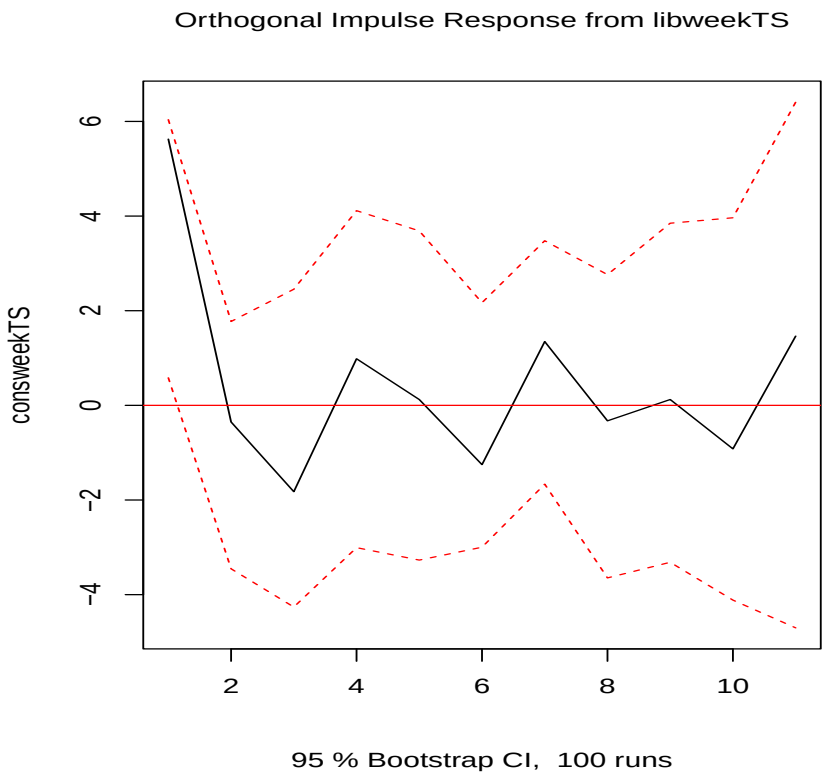
0, insignificant on standard thresholds.

Finally we move to looking at the forecast error variance decomposition, which allows for examining the effect of a particular variable to the future forecast error variance of another variable. To examine this, see the plots below the innovation accounting. These reveal that not one particular variable is contributing to forecast uncertainty. It is notable, however, that the FEVD did detect some variance explained by liberal church attendance on liberal church attendance in the first two horizons.

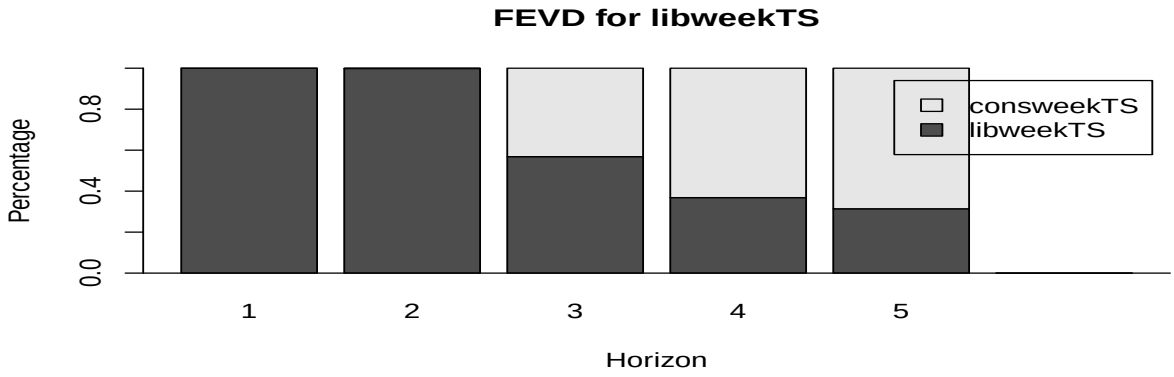
```
> # Innovation accounting:
>  irf1 <- irf(var1, impulse = "consweekTS", response = "libweekTS")
>  plot(irf1)
```

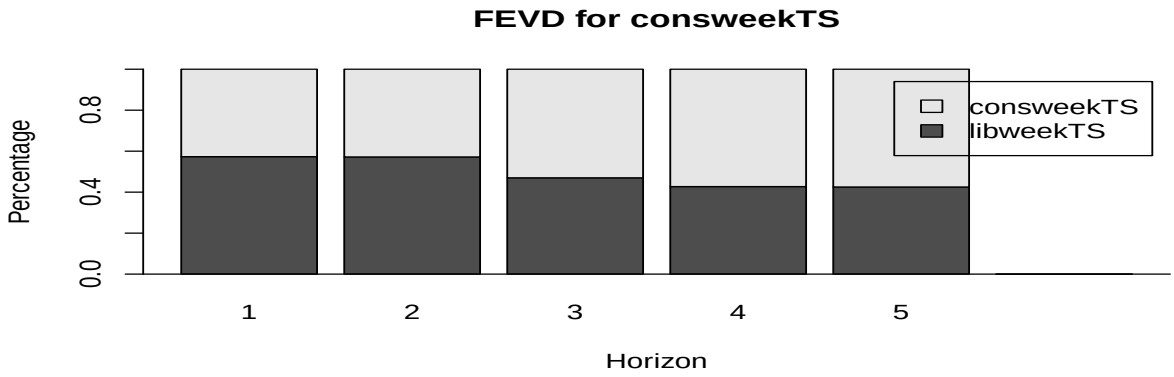



```
>  irf2 <- irf(var1, impulse = "libweekTS", response = "consweekTS")
>  plot(irf2)
```




```
> # Forecast Error Variance Decomposition
>  fevd1 <- fevd(var1, n.ahead = 5)
>  plot(fevd1)
```





Summary, Concluding Thoughts

Overall, the purpose of this assignment was to test the relationship between two series and report how they may relate or not. I chose to test a fundamental underlying assumption of the "god gap" literature, for if these two series did influence one-another then all past examinations would need to take in to account this. However I have found little evidence of this occurring. This makes sense, survey responses of one partisan on church attendance are likely not to consider the other partisan and their historical activites. Rather, I think larger socioeconomic, historical, and geographical variables matter more at moving partisans to church at different levels over time. VAR too can be used to study these variables in relation to the others.

While VAR allows for flexibility in helping determine the data generating process, proper treatment of the error term through lag-length detection, and easy transition to error correction models and OLS it is not without its disadvantages. If a series is nonstationary, statistical testing can become problematic. If the series is cointegrated differencing may not solve these problems. Finally, as I mentioned before, I think a lot of different variables influence the widening and contraction of the god gap, however including all of them for all of their past lags may easily overfit the model if proper precautions are not used.

6 Network Analysis

Network analysis, broadly speaking, is just the analysis of the connection between nodes and the broader interconnections of a group of units. Nodes can be people, events, animals, anything really. Edges can be any measurable connection between nodes, be it frequency of community, survey responses, amount of eye contact - anything really.

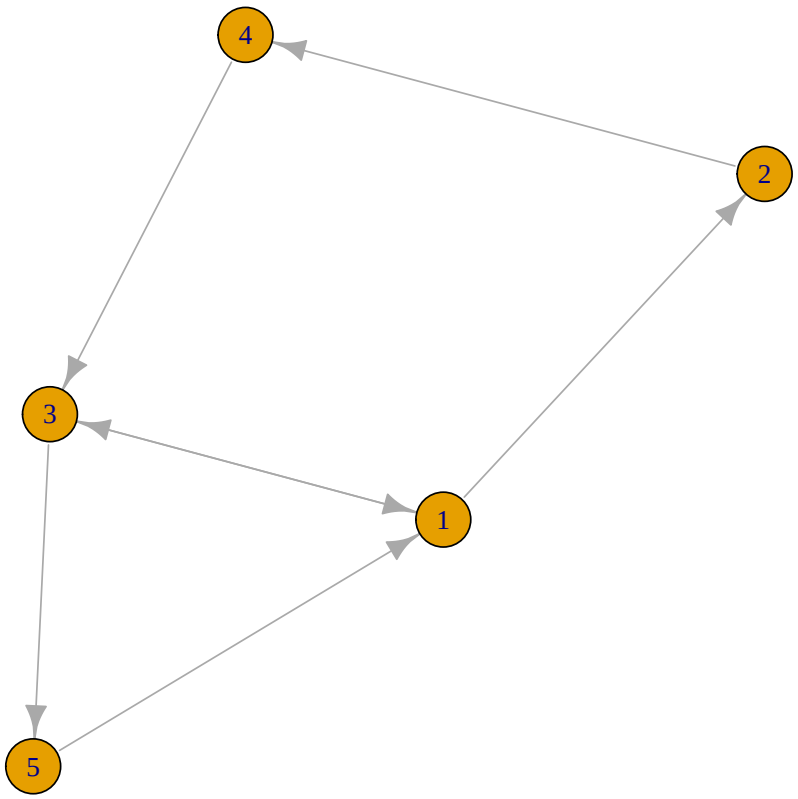
This chapter on network analysis will show the fundamentals of executing network techniques in R, but will dodge a lot of the underlying math making up these methods. A lot of this code and practice has been from previous courses and tutoring I've participated in, so there may be references to this in the comments of the code in this chapter.

6.1 Network Basics: Data

In this section we'll cover just the basics of setting up network data and reading some of the general network statistics.

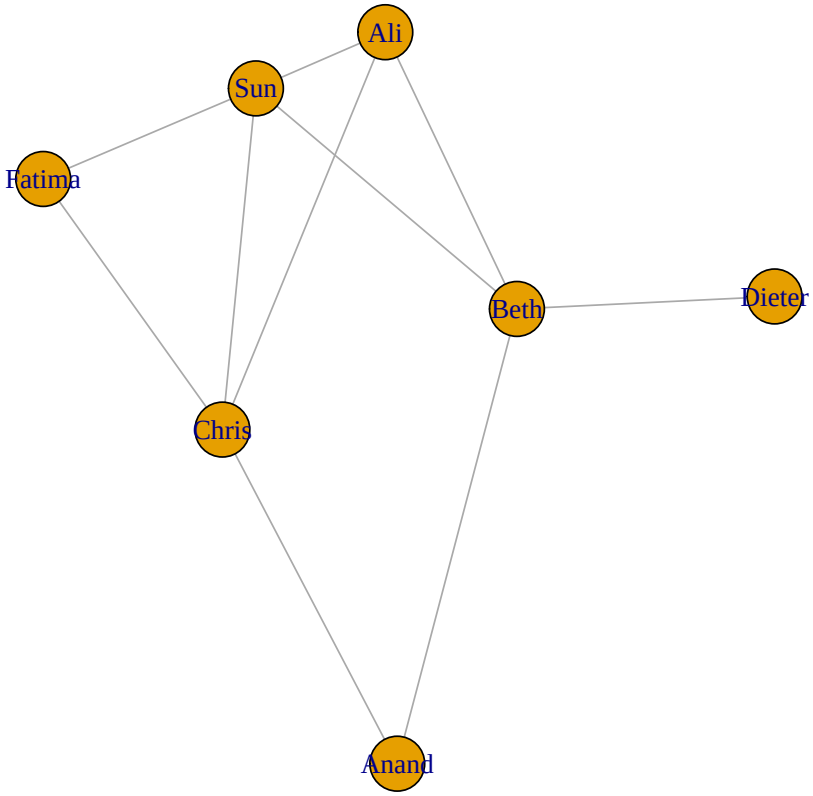
```
> # Packages used:
> library(igraph) # igraph is one of the more popular network analysis packages
> library(igraphdata)
> # use  View() (capital "V" to examine data when you get it.)
>
> # There will be some randomness, we'll set a seed to keep things consistent:
> set.seed(8675305)
>
> # Manually entering Network Data/ viewing networks ----------------------------
>
> # Entering Edge-List data manually (works fine for now).
> # ?graph()
> g1 <- graph(edges=c(1,2,
+                     1,3,
+                     2,4,
+                     3,1,
+                     3,5,
+                     5,1,
+                     4,3))
> # Let's view this:
> g1

## IGRAPH d482d88 D--- 5 7 --
## + edges from d482d88:
## [1] 1->2 1->3 2->4 3->1 3->5 5->1 4->3

> # So, Directional network, 5 nodes, 7 edges.
> # Or more briefly:
> summary(g1)
```

```
## IGRAPH d482d88 D--- 5 7 --

> # Or using the default plot() function:
> plot(g1)
```




```
> # Another Edge-List dataset with names instead of numbers:
> g2 <- graph(edges=c("Anand","Beth",
+                     "Anand","Chris",
+                     "Beth","Dieter",
+                     "Chris","Sun",
+                     "Chris","Ali",
+                     "Beth","Ali",
+                     "Ali","Sun",
+                     "Sun","Beth",
```

```
+                       "Fatima","Chris",
+                       "Fatima","Sun"),
+              directed=FALSE)
> # Now we have no directionality on nodes.
>
> plot(g2)
```

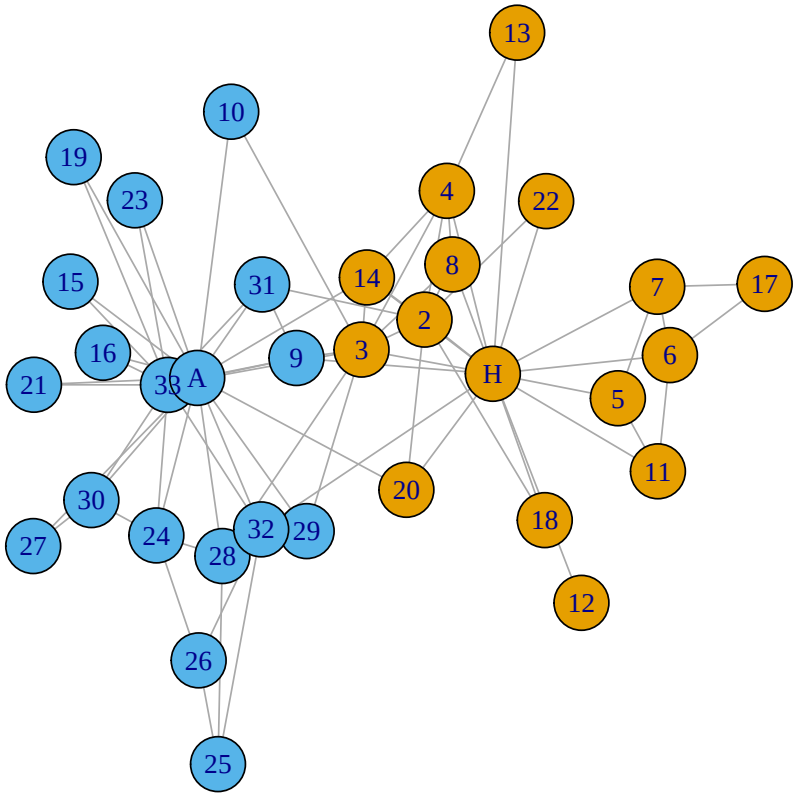



```
> summary(g2) # Undirected, now named, 7 nodes, 10 edges.

## IGRAPH d488b8b UN-- 7 10 --
## + attr: name (v/c)

> # We can use functions to visualize our data another way (beyond edge-list)
> # Adjacency-Matrix:
```

```
>
> g1[]

## 5 x 5 sparse Matrix of class "dgCMatrix"
##
## [1,] . 1 1 . .
## [2,] . . . 1 .
## [3,] 1 . . . 1
## [4,] . . 1 . .
## [5,] 1 . . . .

> ## 5 x 5 sparse Matrix of class "dgCMatrix"
> ##
> ## [1,] . 1 1 . .
> ## [2,] . . . 1 .
> ## [3,] 1 . . . 1
> ## [4,] . . 1 . .
> ## [5,] 1 . . . .
>
> get.adjacency(g2)

## 7 x 7 sparse Matrix of class "dgCMatrix"
##        Anand Beth Chris Dieter Sun Ali Fatima
## Anand      .    1     1      .   .   .      .
## Beth       1    .     .      1   1   1      .
## Chris      1    .     .      .   1   1      1
## Dieter     .    1     .      .   .   .      .
## Sun        .    1     1      .   .   1      1
## Ali        .    1     1      .   1   .      .
## Fatima     .    .     1      .   1   .      .

> ## 7 x 7 sparse Matrix of class "dgCMatrix"
> ##        Anand Beth Chris Dieter Sun Ali Fatima
> ## Anand      .    1     1      .   .   .      .
> ## Beth       1    .     .      1   1   1      .
> ## Chris      1    .     .      .   1   1      1
> ## Dieter     .    1     .      .   .   .      .
> ## Sun        .    1     1      .   .   1      1
> ## Ali        .    1     1      .   1   .      .
> ## Fatima     .    .     1      .   1   .      .
>
> # Similarly, we can vizualise the Edges:
> E(g1)

## + 7/7 edges from d482d88:
## [1] 1->2 1->3 2->4 3->1 3->5 5->1 4->3
```

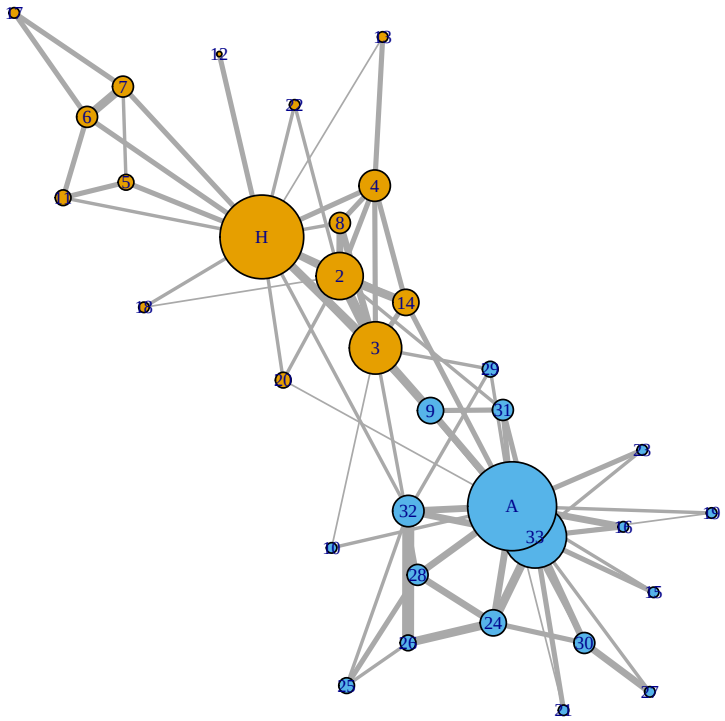
```
> ## + 7/7 edges from fd21364:
> ## [1] 1->2 1->3 2->4 3->1 3->5 5->1 4->3
> V(g2)

## + 7/7 vertices, named, from d488b8b:
## [1] Anand  Beth   Chris  Dieter Sun    Ali    Fatima

> ## + 7/7 vertices, named, from dd35840:
> ## [1] Anand  Beth   Chris  Dieter Sun    Ali    Fatima
>
> # We can also pull up variables from this object: like $name
> V(g2)$name

## [1] "Anand"  "Beth"   "Chris"  "Dieter" "Sun"    "Ali"    "Fatima"

> ## [1] "Anand"  "Beth"   "Chris"  "Dieter" "Sun"    "Ali"    "Fatima"
>
> # So let's plug in more data on top of that:
> V(g2)$gender <- c("M","F","M","M","F","M","F")
> # Here, we're assigning the gender for each vertex.
>
> # Or we'll assign some arbitrary weights to the edges (notice the E(g2) now.)
> E(g2)$count <- rep(1:2,5)
> g2 <- set.vertex.attribute(g2, "age", value = c(20,26,19,34,22,30,21))
> # Because we want R to save this addition to the network, when using the
> # set.vertex.attribute() function, we have to assign this again
>
> # With this new data added, let's summarize the graph:
> summary(g2) # We see the new variables added, and how they are connected.

## IGRAPH d488b8b UN-- 7 10 --
## + attr: name (v/c), gender (v/c), age (v/n), count (e/n)

> ## Larger data ------------------------------------------------------------
>
> # A classical long-standing training dataset for Network Analysis has been
> #  Zachary's Karate Club, some researchers even today test methods on this data
>
> data("karate")
> summary(karate)

## IGRAPH 4b458a1 UNW- 34 78 -- Zachary's karate club network
## + attr: name (g/c), Citation (g/c), Author (g/c), Faction (v/n), name
## | (v/c), label (v/c), color (v/n), weight (e/n)
```

```
> # What does this summary output mean?
> #     - U means undirected (versus D for directed)
> # - N means that the graph is named
> # - W means weighted graph
> # - 34 is the number of nodes
> # - 78 is the number of edges
> # - Other information is various variable data and data types.
>
> plot(karate)
```



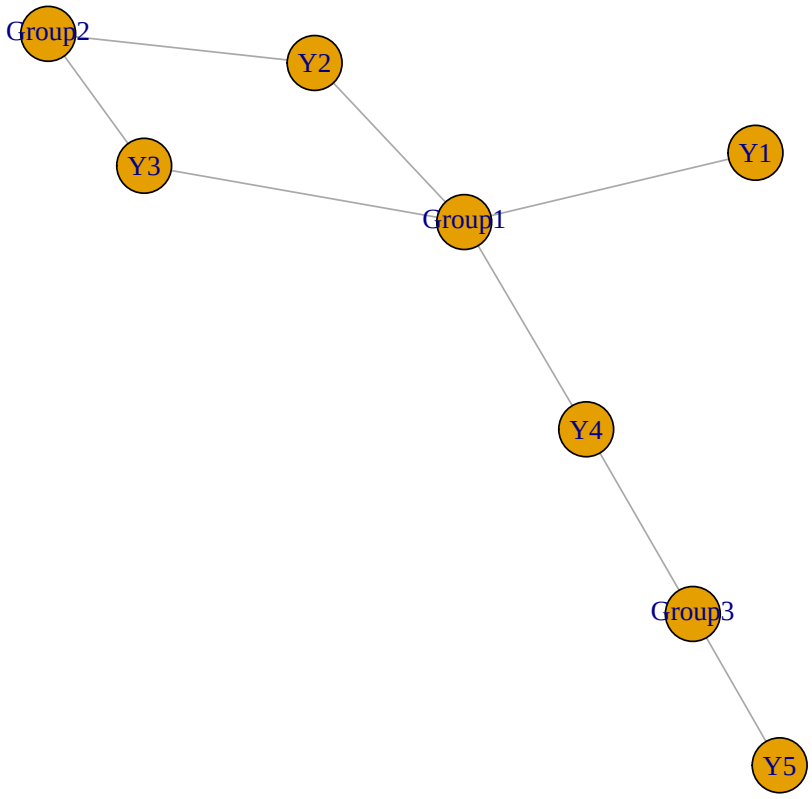

```
> vcount(karate) # Vertical (node) count

## [1] 34
```

```
> # [1] 34
> ecount(karate) # Edge count

## [1] 78

> # [1] 78
>
> # ?degree() # a summation of all edges linked to nodes.
> # or, (undirected) - 2E/N;  (directed) E/N
> mean(degree(karate)) # Average network degree

## [1] 4.588235

> # [1] 4.588235      # "on average" we have 4.5 connections in the karate club
>
> # This is (usually) just: (Actual Connections/Potential Connections)
> graph.density(karate) # Density

## [1] 0.1390374

> # [1] 0.1390374    # This network is about 14% dense/100%
>
> # Diameter gives you the longest distance and path:
> # ?diameter()
> diameter(karate) # It's diameter

## [1] 13

> # [1] 13    # The farthest nodes are 13 people apart.
> get_diameter(karate) # Here we see the long-pathway

## + 6/34 vertices, named, from 4b458a1:
## [1] Actor 16 John A   Actor 20 Mr Hi    Actor 6  Actor 17

> farthest_vertices(karate) # Here we see the people that are the farthest apart.

## $vertices
## + 2/34 vertices, named, from 4b458a1:
## [1] Actor 16 Actor 17
##
## $distance
## [1] 13

> # Some graphical options:
> plot(karate,
+      edge.width = E(karate)$weight,
+      vertex.size = degree(karate)*1.5,
+      edge.arrow.size = 0.4,
+      vertex.label.cex = 0.7)
```



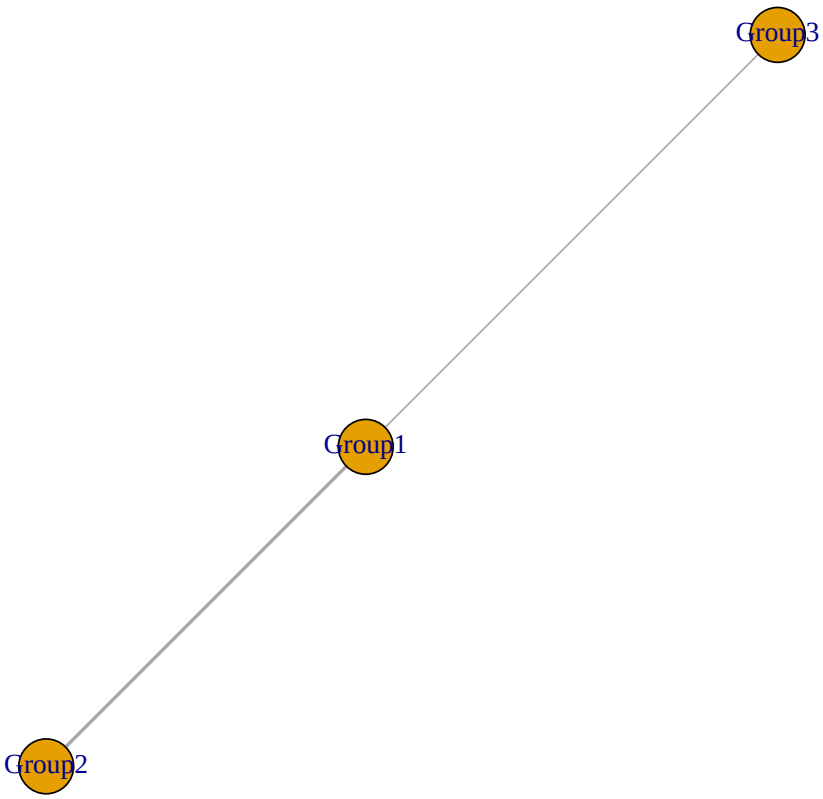

```
> # In this graph we have weighted edges based on our data, the nodes are bigger
> #   depending on their degree "importance"
>
>
> ## Bipartide Networks: ------------------------------------------------------
> # This is where you have the relationship between nodes be that they are
> #   connected to another party.
> #   (I published with X, and so did you, so we're connected)
>
> # Seminar notes on this:
> mem <- matrix(data = c(1,1,1,1,0,
+                        0,1,1,0,0,
+                        0,0,0,1,1),
+               nrow = 3,
```

```
+                byrow = TRUE)
> # This is assigning the names first to the 3 rows, and then to the 5 columns
> dimnames(mem) <- list(c("Group1","Group2","Group3"),
+                       c("Y1","Y2","Y3","Y4","Y5"))
> mem

##        Y1 Y2 Y3 Y4 Y5
## Group1  1  1  1  1  0
## Group2  0  1  1  0  0
## Group3  0  0  0  1  1

> ##        Y1 Y2 Y3 Y4 Y5
> ## Group1  1  1  1  1  0
> ## Group2  0  1  1  0  0
> ## Group3  0  0  0  1  1
>
> bg <- graph.incidence(mem)
> bg  # We see the "B" there now, for Bipartide

## IGRAPH d490cf6 UN-B 8 8 --
## + attr: type (v/l), name (v/c)
## + edges from d490cf6 (vertex names):
## [1] Group1--Y1 Group1--Y2 Group1--Y3 Group1--Y4 Group2--Y2 Group2--Y3
## [7] Group3--Y4 Group3--Y5

> ## IGRAPH 6d6e710 UN-B 8 8 --
> ## + attr: type (v/l), name (v/c)
> ## + edges from 6d6e710 (vertex names):
> ## [1] Group1--Y1 Group1--Y2 Group1--Y3 Group1--Y4 Group2--Y2 Group2--Y3
> ## [7] Group3--Y4 Group3--Y5
> plot(bg)
```

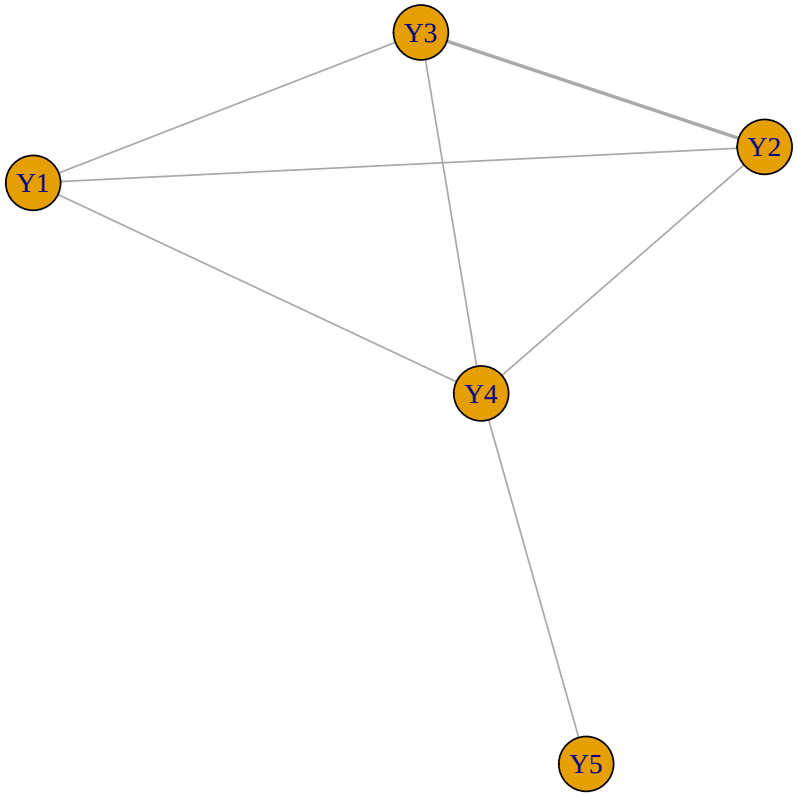



```
> pr <- bipartite.projection(bg)
> pr

## $proj1
## IGRAPH d491f46 UNW- 3 2 --
## + attr: name (v/c), weight (e/n)
## + edges from d491f46 (vertex names):
## [1] Group1--Group2 Group1--Group3
##
## $proj2
## IGRAPH d491f46 UNW- 5 7 --
## + attr: name (v/c), weight (e/n)
## + edges from d491f46 (vertex names):
## [1] Y1--Y2 Y1--Y3 Y1--Y4 Y2--Y3 Y2--Y4 Y3--Y4 Y4--Y5
```

```
> ## $proj1
> ## IGRAPH 749b7be UNW- 3 2 --
> ## + attr: name (v/c), weight (e/n)
> ## + edges from 749b7be (vertex names):
> ## [1] Group1--Group2 Group1--Group3
> ##
> ## $proj2
> ## IGRAPH fbde6d7 UNW- 5 7 --
> ## + attr: name (v/c), weight (e/n)
> ## + edges from fbde6d7 (vertex names):
> ## [1] Y1--Y2 Y1--Y3 Y1--Y4 Y2--Y3 Y2--Y4 Y3--Y4 Y4--Y5
> plot(pr$proj1, edge.width = E(pr$proj1)$weight)
```



```
> plot(pr$proj2, edge.width = E(pr$proj2)$weight)
```




```
> get.adjacency(pr$proj1, sparse = FALSE, attr = "weight")

##        Group1 Group2 Group3
## Group1      0      2      1
## Group2      2      0      0
## Group3      1      0      0

> ##        Group1 Group2 Group3
> ## Group1      0      2      1
> ## Group2      2      0      0
> ## Group3      1      0      0
> get.adjacency(pr$proj2, sparse = FALSE, attr = "weight")
```

```
##    Y1 Y2 Y3 Y4 Y5
## Y1  0  1  1  1  0
## Y2  1  0  2  1  0
## Y3  1  2  0  1  0
## Y4  1  1  1  0  1
## Y5  0  0  0  1  0

> ##    Y1 Y2 Y3 Y4 Y5
> ## Y1  0  1  1  1  0
> ## Y2  1  0  2  1  0
> ## Y3  1  2  0  1  0
> ## Y4  1  1  1  0  1
> ## Y5  0  0  0  1  0
>
>
> aff1 <- mem %*% t(mem)
> diag(aff1) <- 0
> # the diagonal doesn't really have important information for these,
> #   as it's about self-affiliation. With this command,
> #   we change those values to 0.
> aff1

##        Group1 Group2 Group3
## Group1      0      2      1
## Group2      2      0      0
## Group3      1      0      0

> ##        Group1 Group2 Group3
> ## Group1      0      2      1
> ## Group2      2      0      0
> ## Group3      1      0      0
> aff2 <- t(mem) %*% mem
> diag(aff2) <- 0
> aff2

##    Y1 Y2 Y3 Y4 Y5
## Y1  0  1  1  1  0
## Y2  1  0  2  1  0
## Y3  1  2  0  1  0
## Y4  1  1  1  0  1
## Y5  0  0  0  1  0

> ##    Y1 Y2 Y3 Y4 Y5
> ## Y1  0  1  1  1  0
> ## Y2  1  0  2  1  0
```

```
> ## Y3  1  2  0  1  0
> ## Y4  1  1  1  0  1
> ## Y5  0  0  0  1  0
```

6.2 Network Properties, Structural Equivalence, Roles, and Positions.

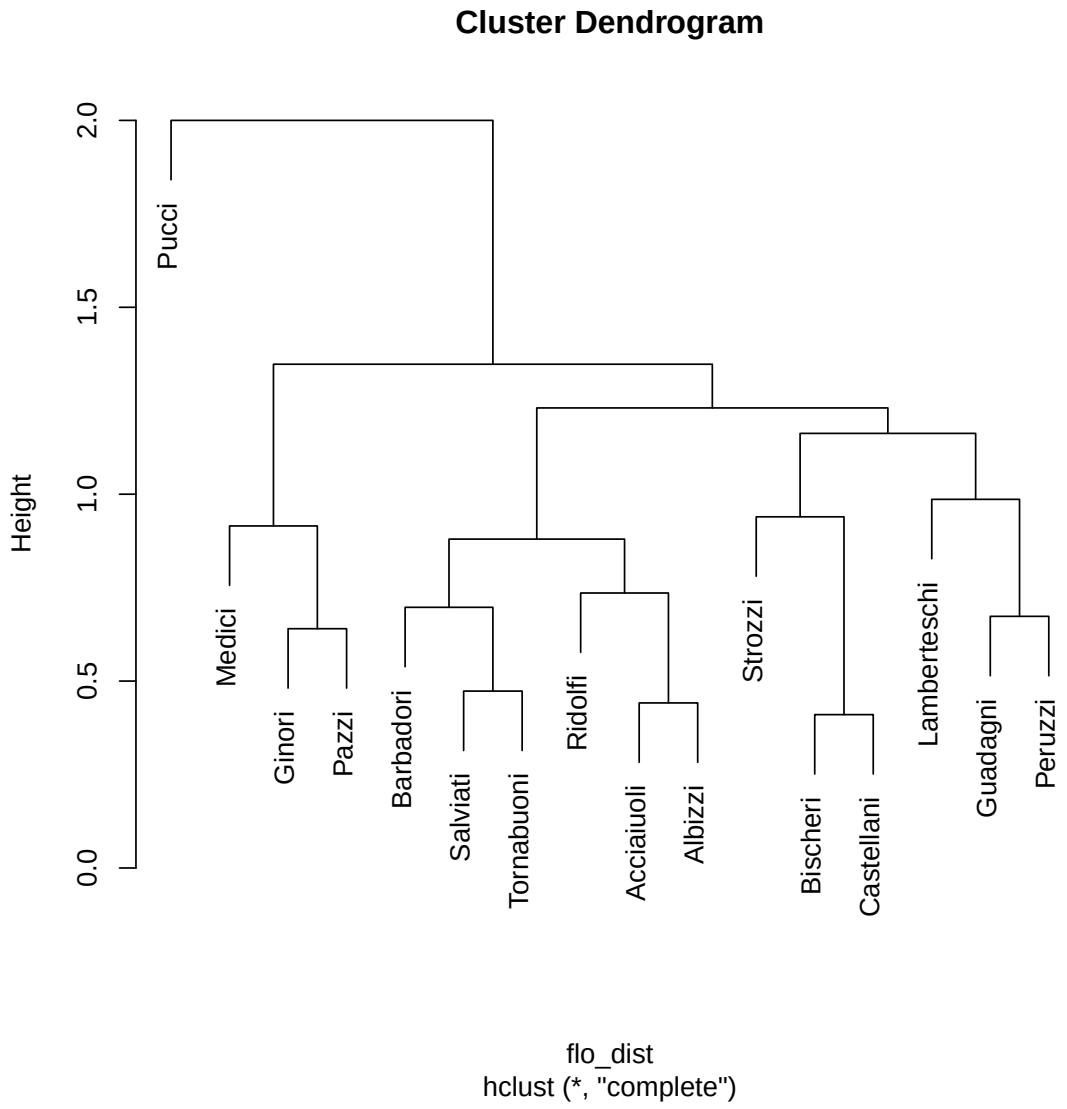
```
> set.seed(123)
>
> # Let's try something fancy:
> # List all packages:
> packages <- c("NetCluster", "blockmodeling", "gplots", "igraph")
>
> # Install all packages in one function:
> #   lapply(packages, install.packages, character.only = TRUE)
>
> # Load all packages in one function:
> invisible( # supressing out output which shows all packages.
+ lapply(packages, library, character.only = TRUE)
+ )
   ## Error in FUN(X[[i]], ...):  there is no package called 'NetCluster'
> #  lapply(object, function, ...)  # For "list" apply... nifty tool!
>
> # swi <- read.csv("swn.csv", header = TRUE, row.names = 1)
> # header = TRUE uses the first row as the column names;
> #  row.names = 1 uses the first column as the row names
> #    (if we put row.names = 2, it would use the second column, etc.)
>
> ## I couldn't find the .csv... so I just took the data from the source
> # https://networkdata.ics.uci.edu/netdata/html/davis.html
> #  and hit "download" - open that r.data with your favorite R platform
>
> library(readr)
> data <- read_csv("https://raw.githubusercontent.com/KyleDavisGithub/Graduate_Methods_H
 ## Error in open.connection(structure(4L, class = c("curl", "connection"),
       conn_id = <pointer: 0x0000000000000679>), :  HTTP error 404.
> davis <- as.data.frame(data)
>
> swi <- as.matrix(davis) # Make incidence matrix - and put as igraph network
 ## Error in as.vector(x, mode):  cannot coerce type 'closure' to vector of
                                 type 'any'
```

```
> swn <- graph.incidence(swi) # This specifically makes a bipartite igraph

         ## Error in graph.incidence(swi):  object 'swi' not found

> swn # UN-B ; 32 89

       ## Error in eval(expr, envir, enclos):  object 'swn' not found

> plot(swn)

## Error in h(simpleError(msg, call)):  error in evaluating the argument 'x'
     in selecting a method for function 'plot':  object 'swn' not found

> swpr <- bipartite.projection(swn) # This makes the graph two one-mode networks

       ## Error in "igraph" %in% class(graph):  object 'swn' not found

> # What is this? - instead of just 1:1 node relationships there are two
> # overlapping networks and the relationships we care about are between people
> # of the other network. so now our edges are weighted, and igraph strangely
> # calls this a nonbipartite graph? Why? Well now we don't really have one
> # anymore! Instead we transferred the bipartide relationship To a weighted
> # relationship using that projection function. - Weights are number of shared
> # Event interactions.
>
> # Note two proj's we get from that projection, via X or Y projection weights
> plot(swpr$proj2,
+      edge.width = E(swpr$proj2)$weight)

## Error in h(simpleError(msg, call)):  error in evaluating the argument 'x'
     in selecting a method for function 'plot':  object 'swpr' not found

> cor(swi)

            ## Error in is.data.frame(x):  object 'swi' not found

> # Correlation of our matrix object to see structural position relationship.
> # Note that Flora and Oliva are structurally similar here and elsewhere
>
>
> #  Going beyond Pearson correlation to see structural similarity:
> # dissimilarity measure (-1 to cor).
> as.dist(1-cor(swi), upper=TRUE)

            ## Error in is.data.frame(x):  object 'swi' not found
```

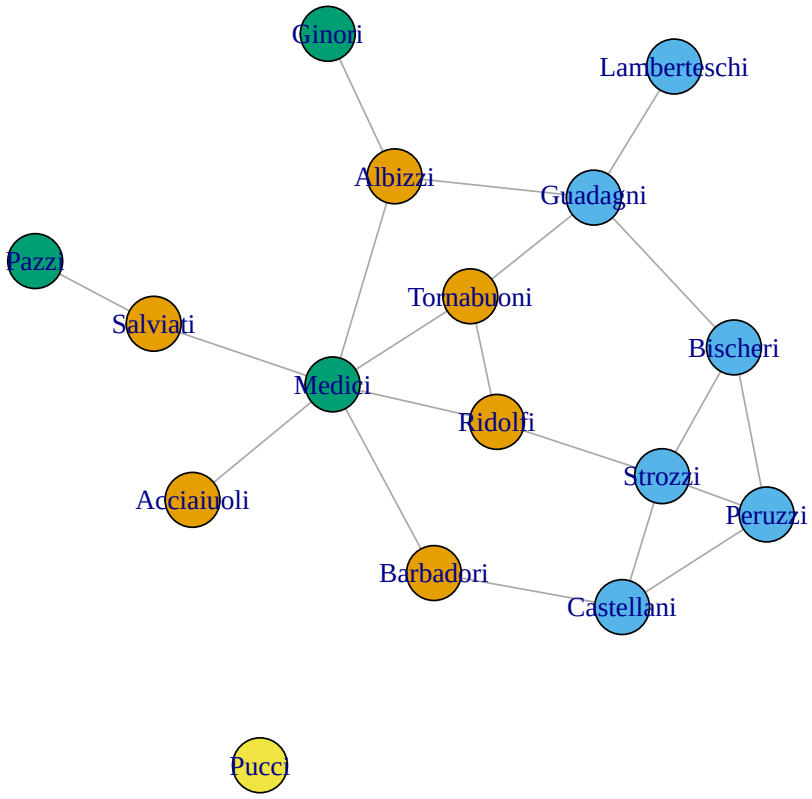
```
> # Upper = T just prints both sides of diag. Doesn't effect analysis
> # We want to move away from just comparing two individuals, but instead
> # find entire clusters of attendants that exibit similar behavior. And then be
> # able to analyze that more interesting result. But to do this, we have to
> # find how nodes are dissimilar, so this data is more valuable to find these
> # clusters! Note we have a wider range now because with the -1 we could have
> # a 1 - a negative -1, or 1+1 (1- -1).
>
> swdend <- hclust(as.dist(1-cor(swi))) # Standard clustering tool

           ## Error in is.data.frame(x):  object 'swi' not found

> plot(swdend) # So how do we read this Dendrogram?

## Error in h(simpleError(msg, call)):  error in evaluating the argument 'x'
    in selecting a method for function 'plot':  object 'swdend' not found

> # Height is our value, and as we parse that value down,it's grouped into values
> # Similar values become grouped under other values.
> #  So, read from the top down, initial separation was around the 1.0 mark,
> #  developing two clusters, which have unique values apart from one another.
> #  as more and more nodes find similarity.
>
>
> # Here is a heatmap with the same viz on each axis
> heatmap.2(as.matrix(as.dist(1-cor(swi))),
+           trace="none",
+           revC=TRUE)

## Error in heatmap.2(as.matrix(as.dist(1 - cor(swi))), trace = "none", revC
                = TRUE): could not find function "heatmap.2"

> cutree(swdend, k=2) # here we are telling R where we should cut our clusters at.

          ## Error in nrow(tree$merge):  object 'swdend' not found

> #  THEORY and your interpretation of the data drives this decision,
> #  a dendrogram will not tell you where to cut your clusters.
>
> # We identified two groups. Let's graph each of these to show our new network:
> plot(swpr$proj2,
+      edge.width=E(swpr$proj2)$weight,
+      vertex.color=cutree(swdend, k=2))

## Error in h(simpleError(msg, call)):  error in evaluating the argument 'x'
     in selecting a method for function 'plot':  object 'swpr' not found
```

```
> ## Let's do the same analysis for a one-mode network.
> # Let's just ignore this package and data coding here for now.. Copy and Pasted
> library(ergm)
> data("florentine")
> require(intergraph)
> require(igraph)
> # igraph  here to override other packages-might have similarly named functions
>
> # Igraph objects, thanks to intergraph package (as seen before)
> flo_m <- asIgraph(flomarriage)
> flo_b <- asIgraph(flobusiness)
>
> # This sets a default name, and fixes an issue with unnamed nodes:
> #   (Professor discovered this error by first not doing this code:)
> V(flo_m)$name <- V(flo_m)$vertex.names
> V(flo_b)$name <- V(flo_b)$vertex.names
>
> # Stacking our adjacency matricies on top of one another by row bind (rbind())
> #  Why? To consider multiple relationship types at the same time.
> flo_both <- rbind(as.matrix(get.adjacency(flo_m,names=TRUE)),
+                   as.matrix(get.adjacency(flo_b,names=TRUE)))
>
> flo_dist <- as.dist(1-cor(flo_both))
> ## Warning in cor(flo_both): the standard deviation is zero
> flo_dist

##              Acciaiuoli   Albizzi Barbadori  Bischeri Castellani    Ginori
## Albizzi       0.4415844
## Barbadori     0.6261217 0.8798272
## Bischeri      1.0862796 0.8798272 1.0256410
## Castellani    1.0862796 1.1545079 1.0256410 0.4102564
## Ginori        1.0577671 1.1034483 0.8798272 1.1545079  0.8798272
## Guadagni      1.0862796 1.1545079 1.2307692 1.0256410  1.0256410 0.8798272
## Lamberteschi  1.0772898 0.8431363 0.7657122 0.5452060  0.9862184 1.1384091
## Medici        1.1299888 1.2327814 1.1791062 1.3476767  1.0105357 0.7813265
## Pazzi         1.0463739 1.0830455 0.7932754 1.1240347  1.1240347 0.6401363
## Peruzzi       1.0950382 1.1701926 0.8668499 0.8668499  0.6731771 0.9108515
## Pucci               NA        NA        NA        NA         NA        NA
## Ridolfi       0.4415844 0.7356322 0.8798272 0.8798272  0.8798272 1.1034483
## Salviati      0.4415844 0.7356322 0.6051466 1.1545079  1.1545079 0.7356322
## Strozzi       1.0678844 1.1215661 0.9394772 0.9394772  0.9394772 1.1215661
## Tornabuoni    0.5248090 0.4732134 0.6973862 0.9394772  1.1815683 0.7973898
##               Guadagni Lamberteschi    Medici     Pazzi   Peruzzi     Pucci
## Albizzi
## Barbadori
```

```
## Bischeri
## Castellani
## Ginori
## Guadagni
## Lamberteschi 0.9862184
## Medici       1.0105357    1.3114511
## Pazzi        1.1240347    1.1111111 0.9150588
## Peruzzi      0.6731771    0.8113260 1.2238141 1.1366260
## Pucci               NA           NA        NA        NA        NA
## Ridolfi      0.8798272    1.1384091 1.0070540 1.0830455 0.9108515        NA
## Salviati     1.1545079    1.1384091 1.2327814 0.6401363 1.1701926        NA
## Strozzi      0.9394772    1.1626500 1.0746047 1.0975900 0.7428571        NA
## Tornabuoni   1.1815683    0.9024100 1.0746047 0.7072300 1.2000000        NA
##                Ridolfi  Salviati   Strozzi
## Albizzi
## Barbadori
## Bischeri
## Castellani
## Ginori
## Guadagni
## Lamberteschi
## Medici
## Pazzi
## Peruzzi
## Pucci
## Ridolfi
## Salviati     0.7356322
## Strozzi      1.1215661 1.1215661
## Tornabuoni   0.7973898 0.4732134 0.8571429

> flo_dist[is.na(flo_dist)==TRUE]<-2 # Fixes NA error by reassigning to number 2
>
> ## With NetCluster
> flo_dend <- hclust(flo_dist)
> plot(flo_dend)
```




```
> ## With gplots
> heatmap.2(as.matrix(flo_dist),trace="none",revC=TRUE)

  ## Error in heatmap.2(as.matrix(flo_dist), trace = "none", revC = TRUE):
                    could not find function "heatmap.2"

> plot(flo_m,vertex.color=cutree(flo_dend, k=4))
```

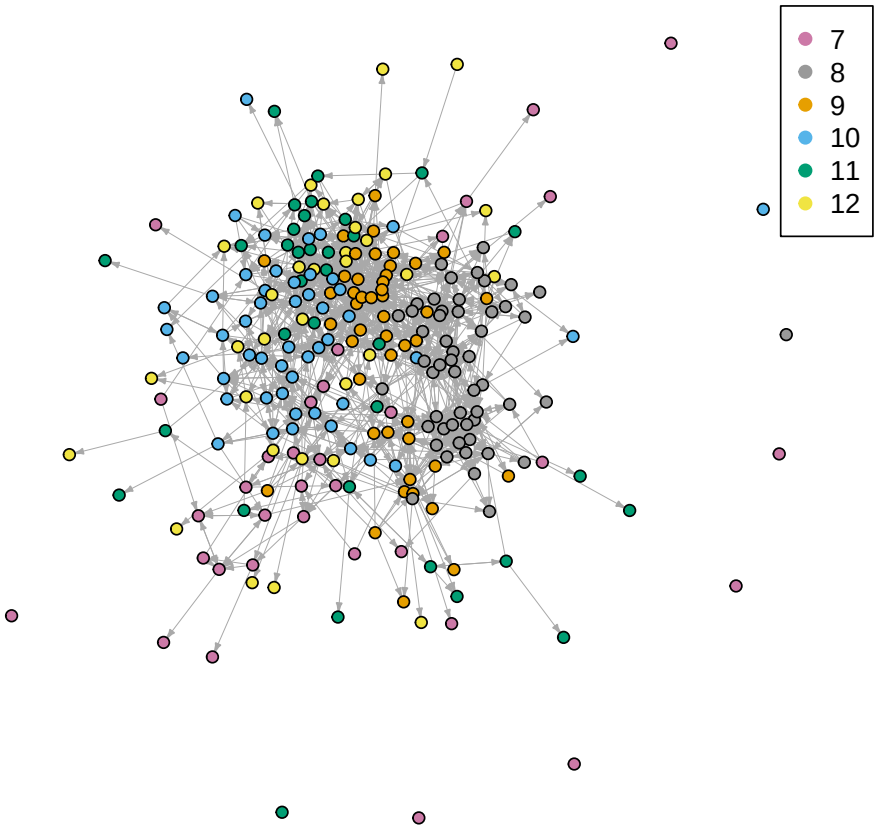



```
> # lots of explaination and data reading in the seminar notes from the course!
> #   Revisit it there if there are any questions it's very well documented.
```

6.3 Diads and Triads

```
> library(ergm)
> data("faux.dixon.high")
> detach(package:ergm)
> detach(package:network)
> require(intergraph)
> require(igraph)
> dixon <- asIgraph(faux.dixon.high)
```

```
>
> # Network Basics:
> summary(dixon)

## IGRAPH d52fd23 D--- 248 1197 --
## + attr: title (g/c), grade (v/n), na (v/l), race (v/c), sex (v/n),
## | vertex.names (v/c), na (e/l)

> # Plotting with color by grade. [How might we clean this code?]
> plot(dixon,
+      vertex.label=NA,
+      edge.arrow.size=0.25,
+      vertex.size=3,
+      edge.width=0.5,
+      vertex.color=V(dixon)$grade)
> legend("topright",
+        legend=7:12,
+        pch=19,
+        col=categorical_pal(8)[c(7,8,1,2,3,4)])
```




```
> ## Dyads:
> #    Reciprocity:  both nodes point to one another.
> #    Here we can get mutual, asymmetrical, and null counts:
>
> dyad.census(dixon) # Note the $ indicator for each object.

## $mut
## [1] 219
##
## $asym
## [1] 759
##
## $null
## [1] 29650
```

```
> # We can calculate this via dyad census (above) or through adjacency matrix.
> reciprocity(dixon)

## [1] 0.3659148

> # Dyad Census:
> 2*dyad_census(dixon)$mut/ecount(dixon)

## [1] 0.3659148

> # two times the mutual count to control for the directionality of edges
> # (which we use as denominator)
>
> # Using the adjacency matrix
> dam <- as.matrix(get.adjacency(dixon))
>
> sum(dam*t(dam))

## [1] 438

> ## Element-wise multiplation of the adjacency matrix, giving us the full number
> #  of mutual edges (equivalent to 2*dyad_census(dixon)$mut.)
>
> sum(dam) ## The count of the total number of edges in the network

## [1] 1197

> #           (equivlaent to ecount(dixon).)
>
> sum(dam*t(dam))/sum(dam)

## [1] 0.3659148

> # Note all three are the same:
> reciprocity(dixon) # Simple function

## [1] 0.3659148

> 2*dyad_census(dixon)$mut/ecount(dixon) # Dyad Census utilization

## [1] 0.3659148

> sum(dam*t(dam))/sum(dam) # Using adjacency matrix

## [1] 0.3659148
```

```
> # 37% chance that if Student A names Student B as their friend; Student B will
> # have also named A. More generally, we can say that 37% of edges are
> # reciprocated.
>
>
> # Or we can use an alternative reciprocity calculation:
> reciprocity(dixon, mode="ratio")

## [1] 0.2239264

> dyad.census(dixon)$mut/(dyad.census(dixon)$mut + dyad.census(dixon)$asym)

## [1] 0.2239264

> # mutual connections / mutal + asym
>
> # Homophily  - relationships among people given a variable
> assortativity.nominal(dixon,V(dixon)$grade)

## [1] 0.5592401

> assortativity.nominal(dixon,V(dixon)$sex)

## [1] 0.1358054

> assortativity.nominal(dixon,factor(V(dixon)$race)) # Had to factorize race here

## [1] 0.5761453

> # .nominal is giving us group-wise assortativity, without that it would treat
> # as continuous. postive number means more assortment (sorting) into like
> # groups with the variable than non-variable.
>
>
> ## Triads
> transitivity(dixon)

## [1] 0.1812741

> # 18% chance of two neighbors of a vertex are themselves connected.
> #    put another way, 18% of all triads are closed triangles (triads)
>
> # Social networks often have a high level of transitivity, compare this to a
> # completely random graph: of same size:
> rand <- sample_gnm(vcount(dixon),ecount(dixon),directed=TRUE)
> summary(rand)
```

```
## IGRAPH d541110 D--- 248 1197 -- Erdos-Renyi (gnm) graph
## + attr: name (g/c), type (g/c), loops (g/l), m (g/n)

> transitivity(rand)

## [1] 0.03954752

> # Triad Census:
> # ?triad.census() # This help page defines most of the below information:
>
> census_labels <- c('003','012','102','021D','021U','021C','111D','111U',
+                    '030T','030C','201','120D','120U','120C','210','300')
> ## this is the order in which the triads appear in the triad census output.
> triad.census(dixon)

##  [1] 2280642  172313   49433    1400    1113    2576    1396    1611     163
## [10]      22     386      73      92      98     135      43

> df <- data.frame(census_labels, triad.census(dixon))
> ## binding the two labels with the output will let us actually interpret this!
> # Most common are unconnected triads, then triads with only one asymmetric edge
>
> triad.census(rand)

##  [1] 2231311  266301    2953    2585    2648    5314     111     124     108
## [10]      32       1       2       3       3       0       0
```

6.4 Network Centrality

Measuring the centrality of a network is indicating which node(s) are most central, or critical, to the network remaining the shape it is in. A central node, for example, can be the person between groups, or just a well connected node in general. There are many ways to measure and detect network centrality, some of which I've studied are listed below. Later in this section we will compare each of these finding that they typically correlate with one another (good!). Cannon advice is to report all of the relevant measures of centrality rather than spending too much time choosing any particular one.

Borgatti (2005):

Name	Definition	Properties	Notes
Betweenness (Freeman, 1979)	$\Sigma_i \Sigma_j \frac{g_{ikj}}{g_{ij}}, \quad i \neq j \neq k$	Assumes traffic is indivisible, traffic takes known-shortest paths,	Likely not a good index for information flow or "gossip"

Eigenvector centrality (Bonacich, 1972)	$\lambda v = Av$	Traffic moves along unrestricted "walks." Traffic can effect subsequent nodes simultaneously. Measures long term and indirect risk.	Related to other measures on sequentially influencing nodes ("influence-type process").
Degree centrality (Freeman, 1979)	Limiting probabilities of nodes by their degree. (Markov Process).	Measures immediate risk. Assumes an infinitely long random walk (a dollar traveling hands).	

But how do we actually do this? Let's start with reading in the data, and setting everything up. I'll save space by not commenting this inital code, just know that it is creating an edgelist matrix for us to generate network objects. This data is from Google's BigQuery - Reddit data in October and November (2017) for the r/asksocialscientists page. This contains those who post and those who comment.

```
> # Initial Packages:
> library(statnet)
> library(dplyr)
> library(igraph)
>
> # Let's read in, add on top of one/ another the Reddit EdgeList data:
> ##   October:
> # read in the posts data
> library(readr)
>
> posts <- read_csv('https://raw.githubusercontent.com/KyleDavisGithub/Graduate_Methods_

## Error in open.connection(structure(5L, class = c("curl", "connection"),
conn_id = <pointer: 0x0000000000000878>), : HTTP error 404.

> # read in the comments data
> comments <- read_csv('https://raw.githubusercontent.com/KyleDavisGithub/Graduate_Metho

## Error in open.connection(structure(6L, class = c("curl", "connection"),
conn_id = <pointer: 0x000000000000087a>), : HTTP error 404.

> comments$id2 <- paste("t1_", comments$id, sep="")

## Error in paste("t1_", comments$id, sep = ""): object 'comments' not
found

> posts$id2 <- paste("t3_", posts$id, sep="")

## Error in paste("t3_", posts$id, sep = ""): object 'posts' not found
```

```
> for_el <- data.frame(
+       author = c(as.character(posts$author), as.character(comments$author)),
+       id = c(as.character(posts$id2), as.character(comments$id2)),
+       parent_id = c(rep(NA, dim(posts)[1]), as.character(comments$parent_id))
+ )

## Error in data.frame(author = c(as.character(posts$author),
as.character(comments$author)), :  object 'posts' not found

> for_el <- for_el[for_el$author!="[deleted]",]

## Error in eval(expr, envir, enclos):  object 'for_el' not found

> el1 <- inner_join(for_el, for_el, c("parent_id" = "id"))

## Error in inner_join(for_el, for_el, c(parent_id = "id")):  object 'for_el'
not found

> el <- data.frame(auth1 = el1$author.x, auth2 = el1$author.y)

## Error in data.frame(auth1 = el1$author.x, auth2 = el1$author.y):  object
'el1' not found

> el <- as.matrix(el)

## Error in as.vector(x, mode):  cannot coerce type 'closure' to vector of
type 'any'

> #-------------------------------------------------------------------------
> ##   Nov:
> posts2 <- read_csv('https://raw.githubusercontent.com/KyleDavisGithub/Graduate_Methods

## Error in open.connection(structure(7L, class = c("curl", "connection"),
conn_id = <pointer:  0x0000000000000883>), :  HTTP error 404.

> comments2 <- read_csv('https://raw.githubusercontent.com/KyleDavisGithub/Graduate_Meth

## Error in open.connection(structure(8L, class = c("curl", "connection"),
conn_id = <pointer:  0x0000000000000885>), :  HTTP error 404.

> comments2$id2 <- paste("t1_", comments2$id, sep="")

## Error in paste("t1_", comments2$id, sep = ""):  object 'comments2' not
found

> posts2$id2 <- paste("t3_", posts2$id, sep="")

## Error in paste("t3_", posts2$id, sep = ""):  object 'posts2' not found
```

```
> for_el2 <- data.frame(
+       author = c(as.character(posts2$author), as.character(comments2$author)),
+       id = c(as.character(posts2$id2), as.character(comments2$id2)),
+       parent_id = c(rep(NA, dim(posts2)[1]), as.character(comments2$parent_id))
+ )

## Error in data.frame(author = c(as.character(posts2$author),
as.character(comments2$author)), :  object 'posts2' not found

> for_el2 <- for_el2[for_el2$author!="[deleted]",]

## Error in eval(expr, envir, enclos):  object 'for_el2' not found

> el12 <- inner_join(for_el2, for_el2, c("parent_id" = "id"))

## Error in inner_join(for_el2, for_el2, c(parent_id = "id")):  object
'for_el2' not found

> el2 <- data.frame(auth1 = el12$author.x, auth2 = el12$author.y)

## Error in data.frame(auth1 = el12$author.x, auth2 = el12$author.y):
object 'el12' not found

> el2 <- as.matrix(el2)

## Error in h(simpleError(msg, call)):  error in evaluating the argument 'x'
in selecting a method for function 'as.matrix':  object 'el2' not found
```

Now that we have the data read in and cleaned (mostly tracking the id numbers), I'll combine the two months and plot the network object that we'll be using for the rest of this section.

```
> # Stack one edge list on top of the other:
> edge.list <- rbind(el, el2)
>
> graph.object <- graph_from_edgelist(edge.list)
> summary(graph.object) # Basic statistics seem okay
>
> # Grabbing the largest component:
> giant.component <- function(graph) {
+       cl <- clusters(graph)
+       induced.subgraph(graph, which(cl$membership == which.max(cl$csize)))
+ }
>
> l.graph.object <- giant.component(graph.object)
>
```

```
> # Let's graph it
> par(mfrow=c(1,2))
> plot.igraph(graph.object,
+             vertex.color="gray15",
+             vertex.size=2.7,
+             vertex.label=NA,
+             vertex.shape="circle",
+             edge.arrow.size=0.4,
+             edge.arrow.width=.6,
+             edge.width=1.5)
> title("\n October:November; 2017",
+       cex.main=1, col.main="gray10")
>
> plot.igraph(l.graph.object,
+             vertex.color="gray15",
+             vertex.size=2.7,
+             vertex.label=NA,
+             vertex.shape="circle",
+             edge.arrow.size=0.4,
+             edge.arrow.width=.6,
+             edge.width=1.5)
> title("\n Largest Component",
+       cex.main=1, col.main="gray10")
> par(mfrow=c(1,1))
```

We can examine some measures of centrality from this network (using the full network instead of largest component). I'll calculate the degree, the inward degree, and outward degree. Then, I find the summary statistics and plot all of these in one concise plot.

```
> # For this assignment, we are only interested in large component:
> l_graph.object <- as_edgelist(l.graph.object)
> # --
> degree <- degree(l.graph.object)
> in.degree <- degree(l.graph.object, mode = 'in')
> out.degree <- degree(l.graph.object, mode = 'out')
>
> # Summary Statistics:
> summary(out.degree)
> summary(in.degree)
> summary(degree)
>
> sd(out.degree)
> sd(in.degree)
```

```
> sd(degree)
>
> # Grammar of Graphics:
> library(ggplot2)
> library(gridExtra)
> library(grid)
>
> p1 <- qplot(out.degree) +
+      theme_bw() +
+      xlab("Outward Degree") +
+      ylab("N") +
+      geom_vline(xintercept=1.859, linetype="dashed") +
+      annotate("text", x = 50, y = 400, label = "sd = 3.30")
>
> p2 <- qplot(in.degree) +
+      theme_bw() +
+      xlab("Inward Degree") +
+      ylab("N") +
+      geom_vline(xintercept=1.859, linetype="dashed") +
+      annotate("text", x = 16, y = 213, label = "sd = 2.23")
>
> p3 <- qplot(degree) +
+      theme_bw() +
+      xlab("Degree") +
+      ylab("N") +
+      geom_vline(xintercept=3.718, linetype="dashed") +
+      annotate("text", x = 61, y = 214, label = "sd = 4.76")
>
> grid.arrange(
+      p1, p2, p3,
+      # Graphing Matrix:
+      widths = c(.5, .9 ,2, 2, .9, .5),
+      layout_matrix = rbind(c(NA, 1, 1, 2, 2, NA),
+                            c(NA, NA, 3, 3, NA, NA)),
+      top = "Network Degree Distributions;
         \n Oct : Nov'r/AskSocialScientists' Reddit Data.",
+      # Any citations:
+      bottom = textGrob(
+           "Mean Reported by Dashed Line.
         \n All Medians = 1 except 'Degree' which = 2 ",
+           gp = gpar(fontface = 3, fontsize = 8),
+           hjust = 1,
+           x = 1
+      )
```

```
+ )
```

The next part was difficult, and probably could be done better. I looked at a list of the ordered centrality measures, then created an edge-graph and looked at those lists similarly. We can see from the lists that some names pop up more often then others (MoralMidgetry, apaniyam, Atriox998), but there has to be some code to get the "max" or to highlight those with the largest degree more clearly. How might we code that?

```
> ### Centrality measures:
> lg_graph.object <- line.graph(graph.object)
>
> ## closeness centrality
> c.graph.object <- closeness(graph.object)
> summary(c.graph.object)
> E(graph.object)[order(c.graph.object, decreasing=T)[1:10]]
>
> c.lg.graph.object <- closeness(lg_graph.object)
> E(graph.object)[order(c.lg.graph.object, decreasing=T)[1:10]]
>
>
> ## betweenness centrality
> b.graph.object <- betweenness(graph.object)
> summary(b.graph.object)
> E(graph.object)[order(b.graph.object, decreasing=T)[1:10]]
>
> b.lg.graph.object <- betweenness(lg_graph.object)
> E(graph.object)[order(b.lg.graph.object, decreasing=T)[1:10]]
>
>
> ## eigenvector centrality
> e.graph.object <- evcent(graph.object)
> summary(e.graph.object$vector)
> E(graph.object)[order(e.graph.object$vector, decreasing=T)[1:10]]
>
> evc_lg_graph.object <- evcent(lg_graph.object)
> E(graph.object)[order(evc_lg_graph.object$vector, decreasing=T)[1:10]]
```

At least from above we know where to start looking for central nodes within our data. Following this, I compare each measure of centrality by generating a matrix of correlation values, then plot it.

```
> # Let's make a matrix of just our values, look at the correlation between them.
> vec.degree <- as.vector(degree)
> vec.in.degree <- as.vector(in.degree)
> vec.out.degree <- as.vector(out.degree)
> vec.c.graph.object <- as.vector(c.graph.object)
> vec.b.graph.object <- as.vector(b.graph.object)
> vec.e.graph.object <- as.vector(e.graph.object$vector)
>
> Cor.Object <- matrix(c(vec.degree,
+                        vec.in.degree,
+                        vec.out.degree,
+                        vec.c.graph.object,
+                        vec.b.graph.object,
+                        vec.e.graph.object,
+                        ncol = 6))
>
> colnames(Cor.Object) <- c("Degree", "In Degree", "Out Degree",
+                           "Closeness", "Betweeness", "Eigenvector")
>
> cor(Cor.Object)
>
> # Trying to think of cool ways to plot this:
> library(corrplot)
> x <- cor(Cor.Object)
> corrplot(x, type="upper", order="hclust")
```

I find it easier to think about correlation by looking at the plot (albeit a very rough plot). The two "darkest" or most correlated measures was the in-degree with degree, and the degree with the out-degree. This makes sense because the degree is encapsulating both of those measures. All measures are positively correlated, except the eigenvector centrality measure compared to closeness and betweeness. This is likely because of the different assumptions and properties each of these measures have. For example, eigenvector centrality Trafic moves along unrestricted walks, Trafic can effect subsequent nodes simultaneously, and eigenvector centrality measures long-term and indirect risks. Other measures may not have these properties, so it is probably best that they are not too closely correlated.

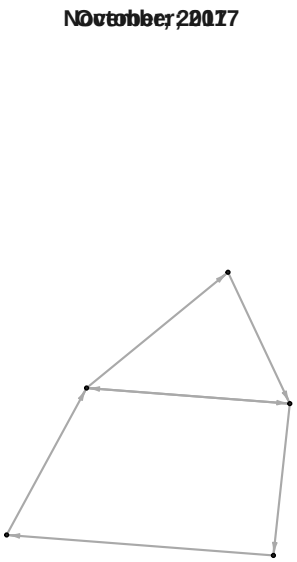
6.4.1 Data Viz

Perhaps now is a good opportunity to talk about data visualization. Using the data from the previous section on centrality: In this I will be using the Reddit data from "r/AskSocialScientists" from October and November. These will be ran separately, and I will look into their largest components. Then, I will plot the degrees in two different ways.

```
> # Using the data set up from the previous section:
> ##    Oct:
> library(igraph)
> g1 <- graph_from_edgelist(el)

## Error in graph_from_edgelist(el):  graph_from_edgelist expects a matrix
with two columns

> summary(g1) #

## IGRAPH d482d88 D--- 5 7 --

> #-------------------------------------------------------------------------
> ##   Nov:
> g12 <- graph_from_edgelist(el2)

## Error in graph_from_edgelist(el2):  object 'el2' not found

> summary(g12) #

## Error in h(simpleError(msg, call)):  error in evaluating the argument
'object' in selecting a method for function 'summary':  object 'g12' not
found
```

```
> set.seed(8675309)  # 1981 throwback seed
>
> par(mfrow=c(1,2))
> plot.igraph(g1,
+             vertex.color="gray15",
+             vertex.size=2.7,
+             vertex.label=NA,
+             vertex.shape="circle",
+             edge.arrow.size=0.4,
+             edge.arrow.width=.6,
+             edge.width=1.5)
> title("\n October; 2017",
+       cex.main=1, col.main="gray10")
>
> plot.igraph(g12,
+             vertex.color="gray15",
+             vertex.size=2.7,
+             vertex.label=NA,
+             vertex.shape="circle",
+             edge.arrow.size=0.4,
```

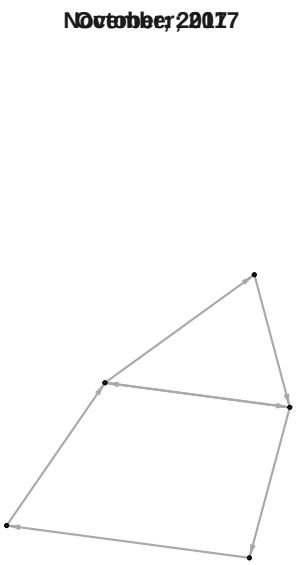
```
+              edge.arrow.width=.6,
+              edge.width=1.5)
 ## Error in plot.igraph(g12, vertex.color = "gray15", vertex.size = 2.7, :
                           object 'g12' not found

> title("\n November; 2017",
+       cex.main=1, col.main="gray10")
> par(mfrow=c(1,1)) # basic reset
```



Above we have both of our complete networks for both months. In these graphs we simply have users (nodes) and if they have commented on each other's posts (edges). This may include some information that is less interesting, like users that have no comments. Below we will take the two largest components of each network and plot it.

```
> giant.component <- function(graph) {
+      cl <- clusters(graph)
+      induced.subgraph(graph, which(cl$membership == which.max(cl$csize)))
+ }
```

```
>
> g2 <- giant.component(g1)
> g22 <- giant.component(g12)
      ## Error in "igraph" %in% class(graph):  object 'g12' not found

> par(mfrow=c(1,2))
> plot.igraph(g2,
+             vertex.color="gray15",
+             vertex.size=2.7,
+             vertex.label=NA,
+             vertex.shape="circle",
+             edge.arrow.size=0.4,
+             edge.arrow.width=.6,
+             edge.width=1.5)
> title("\n October; 2017",
+       cex.main=1, col.main="gray10")
>
> plot.igraph(g22,
+             vertex.color="gray15",
+             vertex.size=2.7,
+             vertex.label=NA,
+             vertex.shape="circle",
+             edge.arrow.size=0.4,
+             edge.arrow.width=.6,
+             edge.width=1.5)

 ## Error in plot.igraph(g22, vertex.color = "gray15", vertex.size = 2.7, :
                          object 'g22' not found

> title("\n November; 2017",
+       cex.main=1, col.main="gray10")
```



For the above, I have chosen to keep plots simple and not too complex to over complicate interpretation. In a paper these graphs would likely print and be easily read. Some things that I may consider changing is making the network bigger, and to have directionality more clear.

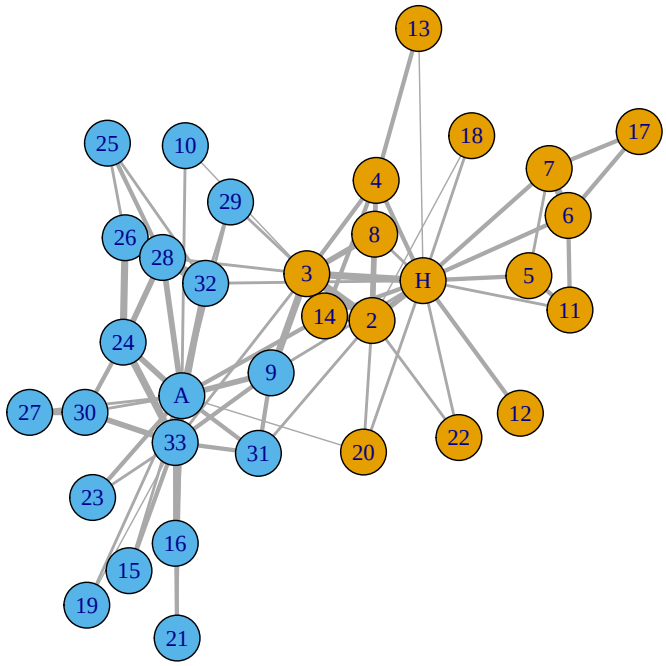
This is, of course, not the only information we can gain from a network. Below is a plot of the network degrees. A network "degree" is the number of connections each node has - a useful statistic. The following graphs intended purpose is to report the mean, the nodes, and edges in addition to the distribution of degrees in a network.

```
> Degree <- degree(g2)
 ## Error in FUN(X[[i]], ...):  as.edgelist.sna input must be an adjacency
 matrix/array, edgelist matrix, network, or sparse matrix, or list thereof.

> mean(Degree)

## Error in h(simpleError(msg, call)):  error in evaluating the argument 'x'
   in selecting a method for function 'mean':  object 'Degree' not found
```

```
> library(ggplot2)
> qplot(Degree) +
+      theme_bw() +
+      ylab("N") +
+      xlab("Degrees") +
+      # ggtitle("Degrees in Large Component of Network 1")+
+      ggtitle(substitute(
+           paste( "Large Component Degrees, with: ", N[V], "=", 231,
+                  " and ", N[E], "=", 404))) +
+      annotate("text", x = 35.4, y = 3.7, label = "35") +
+      geom_vline(xintercept=mean(Degree), linetype="dashed") +
+      annotate("text", x = 4.3, y = 71,
+               label = round(mean(Degree),3),
+               angle=-90)

  ## Error in eval_tidy(mapping$x, data, caller_env): object 'Degree' not
                                    found
```

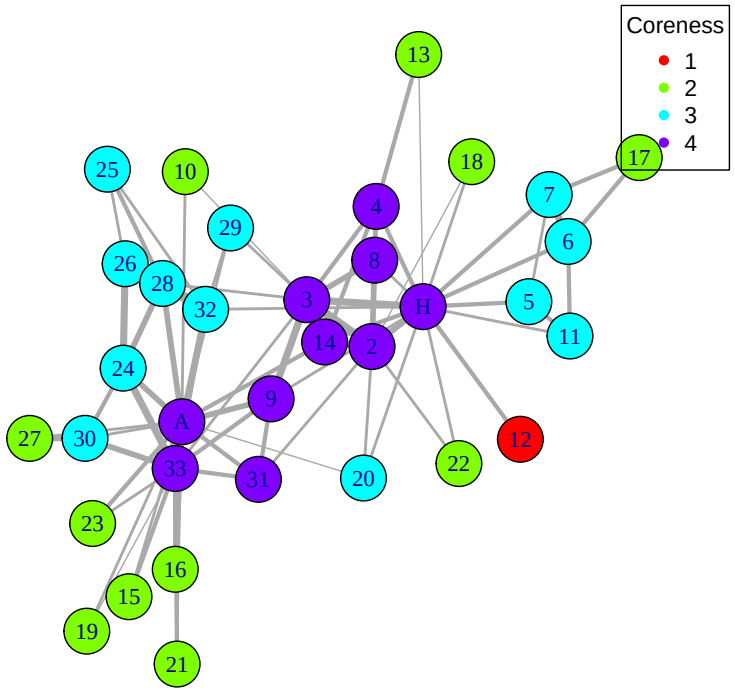
6.5 Assortativity and Community Structures

```
> # Let's expand on the homopholy measurements from Session_5.R
> # We'll use Zachary's karate club data again:
>
> library(igraph)
> library(igraphdata)
> library(ape)
> # this package will help us easily plot the dendrograms for community detection
> # algorithms that use hierarchical clustering
> library(Matrix)
> # this package gives us the image() function for visualizing adjacency matrices
>
>
> data("karate")
> summary(karate) #karate$Citation   is attached to this data remember

## IGRAPH 4b458a1 UNW- 34 78 -- Zachary's karate club network
## + attr: name (g/c), Citation (g/c), Author (g/c), Faction (v/n), name
## | (v/c), label (v/c), color (v/n), weight (e/n)

> frlay <- layout_with_fr(karate) # preset coordinates generated for us...
> # this will ensure that each time we plot the network,
> # the location of the nodes stays constant
> plot(karate, edge.width = E(karate)$weight, layout = frlay)
```




```
> ## Components
> #  Networks are made up of components, and isolated individuals.
> #  You can identify the components of a graph with decompose.graph()
> #  This gives a potentially long list of each component.
> #  So, table(sapply(decompose.graph(yournetwork),vcount)) will give you
> #  a quick summary of the size and number of components.
>
> is_connected(karate)  # There is only one component within this network.

## [1] TRUE

> ## Cores (k)
> #   We could peel away layers of a network, by those who are weekly laching
> #   on (degree of 1) all the way inwards (k-cores). We could consider these
> #   layers individually, or all at once by seeing the greatest k-core for
> #   each node
>
> coreness(karate)
```

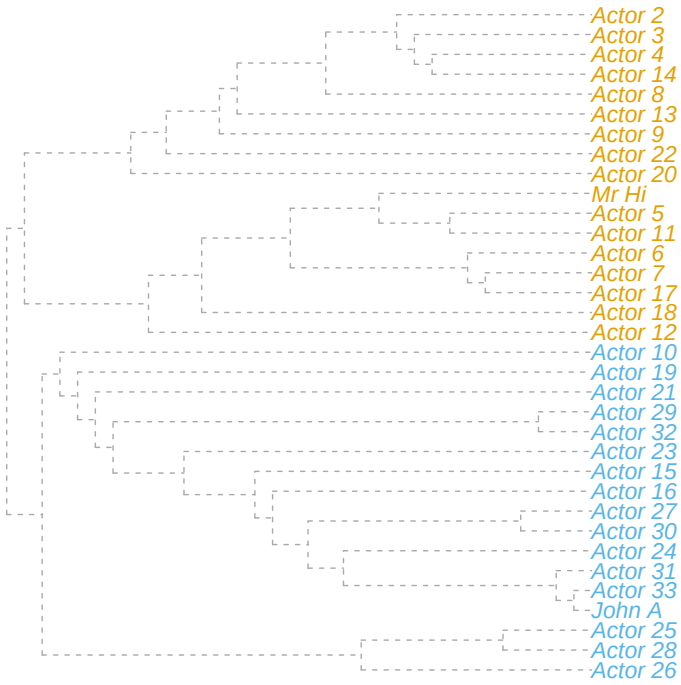
```
##    Mr Hi  Actor 2  Actor 3  Actor 4  Actor 5  Actor 6  Actor 7  Actor 8
##        4        4        4        4        3        3        3        4
##  Actor 9 Actor 10 Actor 11 Actor 12 Actor 13 Actor 14 Actor 15 Actor 16
##        4        2        3        1        2        4        2        2
## Actor 17 Actor 18 Actor 19 Actor 20 Actor 21 Actor 22 Actor 23 Actor 24
##        2        2        2        3        2        2        2        3
## Actor 25 Actor 26 Actor 27 Actor 28 Actor 29 Actor 30 Actor 31 Actor 32
##        3        3        2        3        3        3        4        3
## Actor 33   John A
##        4        4

> colors <- rainbow(max(4))
>
> plot(karate,
+      vertex.color = colors[coreness(karate)],
+      edge.width = E(karate)$weight,
+      layout = frlay)
> legend("topright",
+        legend=c(1:4),
+        col = colors,
+        pch = 16,
+        title = "Coreness")
```



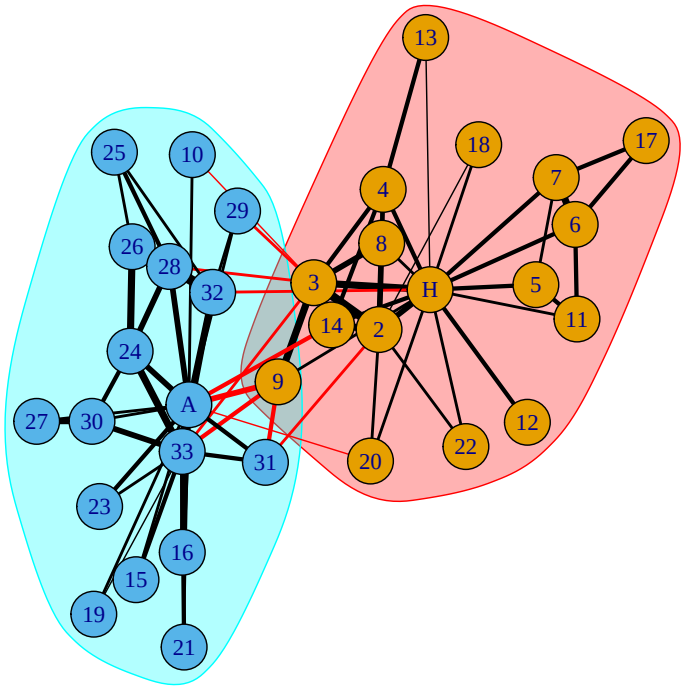

```
> ## Cliques
> #   These are the maximal connected subgraphs.
> #   Note: we can relax the definition of this by including non-connected nodes
> #   (but at short distance "n") igraph does not have this functionality (n=1).
> # Visual definition of Clique:
> #https://en.wikipedia.org/wiki/Clique_(graph_theory)#/media/File:VR_complex.svg
>
> head(cliques(karate, min=3)) # here we excluded cliques below a certain size.

## [[1]]
## + 3/34 vertices, named, from 4b458a1:
## [1] Mr Hi   Actor 2 Actor 3
##
## [[2]]
## + 3/34 vertices, named, from 4b458a1:
## [1] Actor 29 Actor 32 John A
##
## [[3]]
## + 3/34 vertices, named, from 4b458a1:
```

```
## [1] Actor 32 Actor 33 John A
##
## [[4]]
## + 3/34 vertices, named, from 4b458a1:
## [1] Actor 31 Actor 33 John A
##
## [[5]]
## + 3/34 vertices, named, from 4b458a1:
## [1] Actor 27 Actor 30 John A
##
## [[6]]
## + 3/34 vertices, named, from 4b458a1:
## [1] Actor 30 Actor 33 John A

> table(sapply(cliques(karate),length))

##
##  1  2  3  4  5
## 34 78 45 11  2

> # This summarizes the full enumeration of cliques, showing us the N of cliques
> largest_cliques(karate)

## [[1]]
## + 5/34 vertices, named, from 4b458a1:
## [1] Actor 2 Mr Hi   Actor 4 Actor 3 Actor 8
##
## [[2]]
## + 5/34 vertices, named, from 4b458a1:
## [1] Actor 2  Mr Hi    Actor 4  Actor 3  Actor 14

> #### Community detection
> #    There are so many ways to chop up a network into related nodes.
> #    Generally speaking, we want subsets of nodes that are well connected
> #    among them selves, but few connections from outsiders.
> #     There are mountains of ways to parse this out mathematically,
> #     but with R things can be easier.
>
> ### Hierarchical clustering (see Session 4 for some of this).
>
> ## Edge Betweenness
> #   Calculate betweenness (we did this in centrality session) of each edge,
> #   remove the edge (so divide the network along that edge) with the highest
> #   betweeness, recalculate betweenness, identify the new highest edge.. repeat
> #   Note:  edges with weights get interpreted as additional distance
```

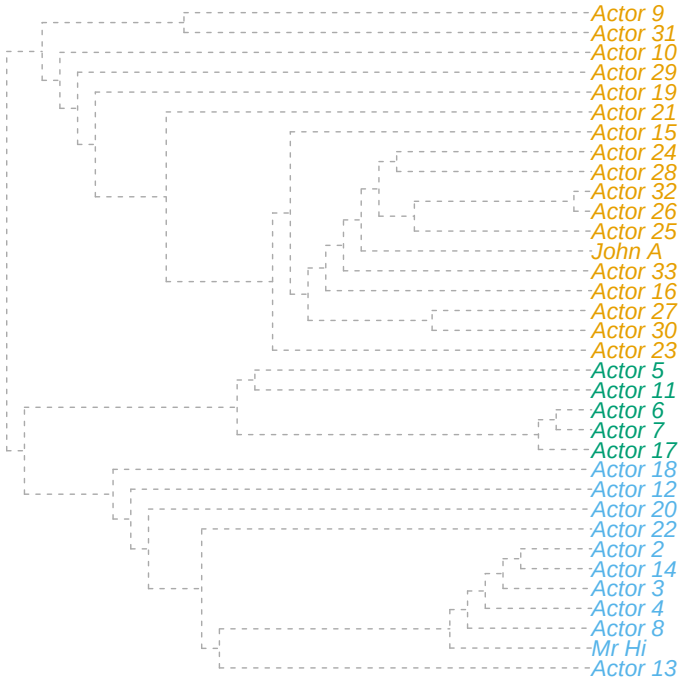
```
> #          (this is not the case in our network!) have to adjust this
> #          calculation to get the weights on the right emphasis.
>
> eb <- cluster_edge_betweenness(karate, weights = 1/E(karate)$weight)
> membership(eb)

##    Mr Hi  Actor 2  Actor 3  Actor 4  Actor 5  Actor 6  Actor 7  Actor 8
##        1        1        1        1        1        1        1        1
##  Actor 9 Actor 10 Actor 11 Actor 12 Actor 13 Actor 14 Actor 15 Actor 16
##        1        2        1        1        1        1        2        2
## Actor 17 Actor 18 Actor 19 Actor 20 Actor 21 Actor 22 Actor 23 Actor 24
##        1        1        2        1        2        1        2        2
## Actor 25 Actor 26 Actor 27 Actor 28 Actor 29 Actor 30 Actor 31 Actor 32
##        2        2        2        2        2        2        2        2
## Actor 33   John A
##        2        2

> sizes(eb)

## Community sizes
##  1  2
## 17 17

> # library(stats)-used for some alternative plotting options in our dendrograms
>
> dendPlot(eb, mode = "phylo")
```



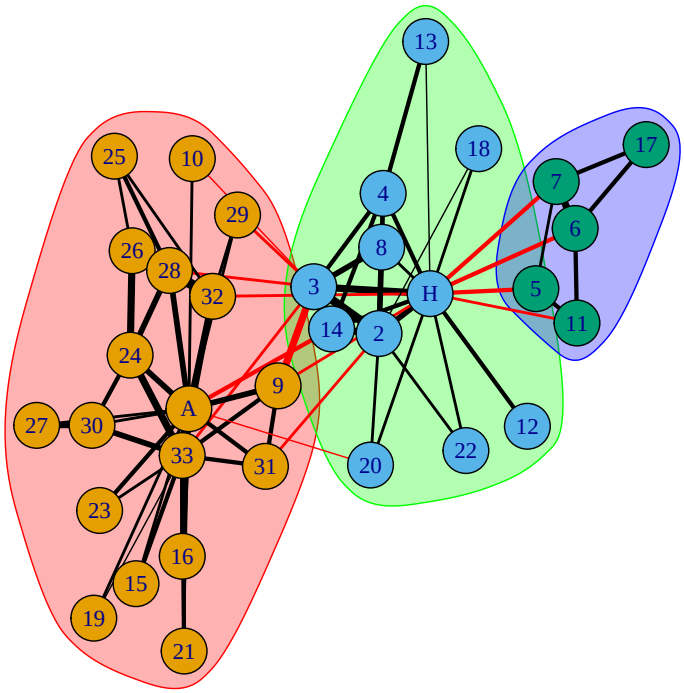

```
> plot(eb, karate, edge.width = E(karate)$weight, layout = frlay)
```



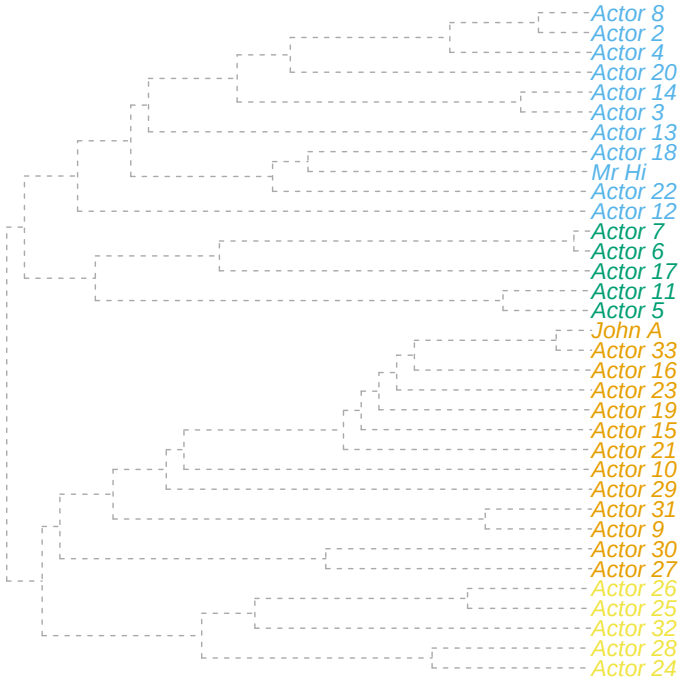

```
> # by putting the community object (here, "eb") in front of the graph object,
> #    igraph knows to plot both. igraph then colors the nodes by their community
> #    membership, draws bubbles around each community, and colors edges within
> #    communities black and between communities red.
>
>
> ## Fast Greedy (Modularity)
> #    Similar to "modularity" when we measured assorativity before, this can be
> #    used to partition communities. chopping groups up by their assortatitivy
> #    scores. Done within, then between all nodes. meaning there are more edges
> #    within groups and fewer between them -- then we find communities.
> fg <- cluster_fast_greedy(karate, weights = E(karate)$weight)
> sizes(fg)

## Community sizes
##  1  2  3
## 18 11  5

> dendPlot(fg, mode="phylo")
```



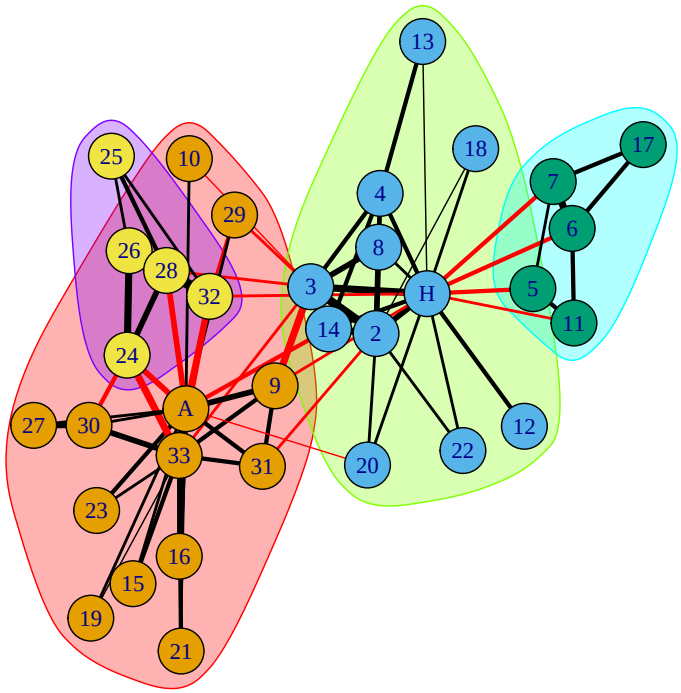

```
> plot(fg, karate, edge.width = E(karate)$weight, layout = frlay)
```



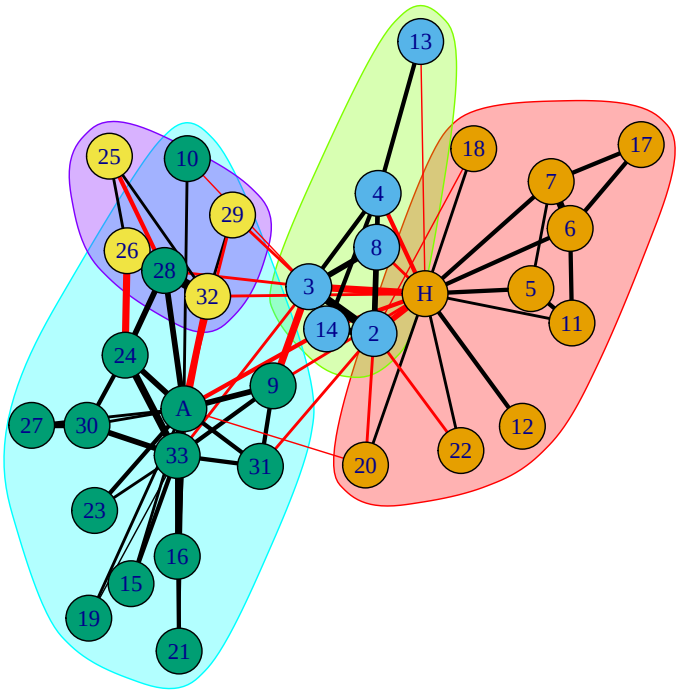

```
> ## Walktrap
> #   imagine doing many, many, short random walks across a network. You'll
> #   likely end up staying in a community on each random walk since
> #   between-communities ties should be rare to find. So we can use these random
> #   walks to find communities (on average) and use that measure as yet another
> #   hierarchical clustering process! [consider simulation, machine learning,
> #   and ways to take these random walks out of your hands]
>
> wt <- cluster_walktrap(karate, weights = E(karate)$weight) # uses edge weight
> dendPlot(wt)
```



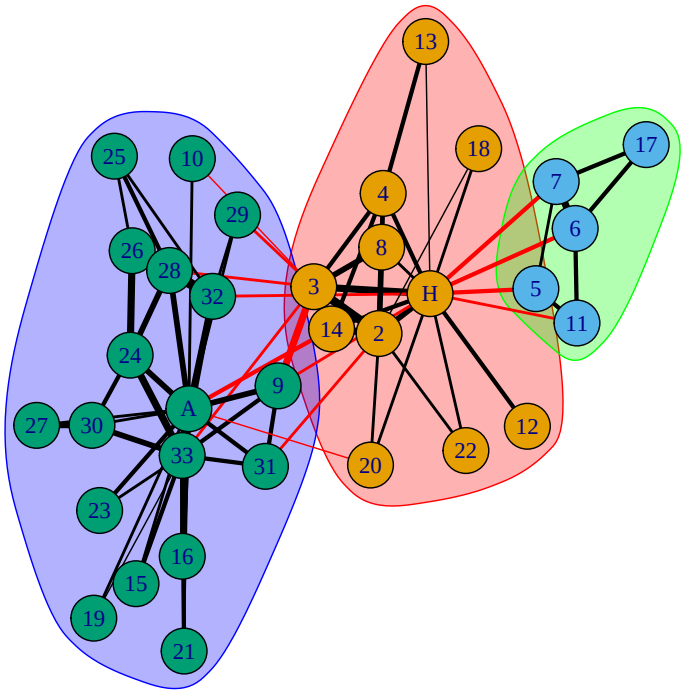

```
> plot(wt, karate, edge.width = E(karate)$weight, layout = frlay)
```



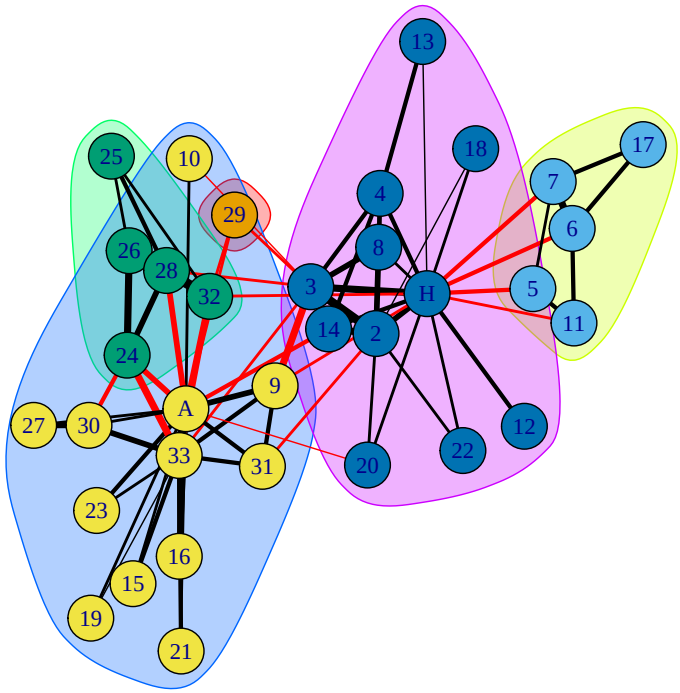

```
> ### Other approaches:
>
> ## Louvain (aka Multilevel)
> #   this one is also trying to quickly optimize modularity. Each node here is
> #   assigned to a different community, then a node is moved to join one of its
> #   neighbors with wich it increases the modularity score. This process of
> #   moving nodes around keeps going until modularity cannot be improved anymore.
> #   [see Yang et al 2016] - this performs really well.
> #   This doesn't classify as a heirarchical community deteticion because it is
> #   not strictly agglomerative. (bottom up) compared to "divisive" top-down
> #   methodologies.
> ml <- cluster_louvain(karate, weights = E(karate)$weight)
> plot(ml, karate, edge.width = E(karate)$weight, layout = frlay)
```




```
> ## InfoMap
> #    Improvement on walktrap - draws on information theory
> im <- cluster_infomap(karate, e.weights=E(karate)$weight)
> plot(im, karate, edge.width = E(karate)$weight, layout = frlay)
```




```
> ## Spinglass
> #     Drawing from physics, and magnetism - a spin glass type of magnet.
> #     Each node is similar to an atom, and we want to simulate the annealing
> #     process where we minimize energy and this happens when nodes are grouped
> #     in communities with the same spin - than different spin groups.
> sg <- cluster_spinglass(karate, weights=E(karate)$weight)
> plot(sg, karate, edge.width = E(karate)$weight, layout = frlay)
```




```
> ### Which one do I use?!
> #     often the "big picture" is the same, but often we'll find small changes
> #     between methods. When publishing, it can be better to publish many
> #     different methods in a table or easily reportable format.
> #     (See Seminar notes for some more information on each of these methods)
>
>
> ### Graph Partitions and "Ground Truth"
> #     So we can mathematically parse out groups - but how does this align with
> #     our theory, and other data? How does this match with our homophily?
>
> # First step: let's calculate assortativity
> #      (normalized ---  -1 (disassortive) :  1(assortive) [0 = no evidence])
>
> modularity(karate, V(karate)$Faction)

## [1] 0.3714661

> assortativity(karate, V(karate)$Faction)
```

```
## [1] 0.7434211

> #  So this assortativity measure is very high... and matches the karate club
> #  theory and our knowledge of it. And we've already eye-balled all the graphs,
> #  and things seem to line up to these factions.
>
> #  In igraph we can compare() different membership vectors using nmi or
> #  "normalized mutual information" 1 = perfect match!  0 = mutually
> #  uninformative of one another.
>
> # Compare "Truth" (faction)  with method:
> compare(V(karate)$Faction, eb, method = "nmi")

## [1] 0.8371695

> ## [1] 0.8371695
> compare(V(karate)$Faction, fg, method = "nmi")

## [1] 0.8255182

> ## [1] 0.8255182
> compare(V(karate)$Faction, wt, method = "nmi")

## [1] 0.6956222

> ## [1] 0.6956222
> compare(V(karate)$Faction, ml, method = "nmi")

## [1] 0.700314

> ## [1] 0.6872629
> compare(V(karate)$Faction, im, method = "nmi")

## [1] 0.8255182

> ## [1] 0.8255182
> compare(V(karate)$Faction, sg, method = "nmi")

## [1] 0.661137

> ## [1] 0.6543404
> table(V(karate)$Faction, membership(eb))

##
##      1  2
##   1 16  0
##   2  1 17
```

```
> table(V(karate)$Faction, membership(fg))

##
##      1  2  3
##   1  0 11  5
##   2 18  0  0

> table(V(karate)$Faction, membership(im))

##
##      1  2  3
##   1 11  5  0
##   2  0  0 18

> #  So if our theory is strong in that these factions matter:
> #     we see that edge betweenness does "best" with only one person
> #     assigned wrong faction. with fast_greedy and infomap coming close
> #     behind in faction membership detection.
>
>
> ## we can also compare different partitionings based on different
> #  community detection algorithms.
> compare(eb, sg, method = "nmi")

## [1] 0.5655214

> ## [1] 0.5622443
> compare(wt, sg, method = "nmi")

## [1] 0.9615489

> ## [1] 0.8953659
> compare(fg, im, method = "nmi")

## [1] 1

> ## we can see that these two community detection algorithms produce
> #  exactly the same partitioning of the graph
> ## [1] 1
> table(membership(fg),membership(im))

##
##      1  2  3
##   1  0  0 18
##   2 11  0  0
##   3  0  5  0
```

```
> ## Stochastic Block Models.
> #   Just create a model to calculate the probability of ties within each group
> #   (controlling for variables).
> #   (Have to swap packages out of igraph and data out of igraph)
>
> require(intergraph)
> detach(package:igraph)
> require(sna)
> knet <- asNetwork(karate)
> ## this is the function from intergraph that takes a igraph graph object and
> #      makes it into a statnet graph object
> blockmodel(knet, ec = knet %v% "Faction")

##
## Network Blockmodel:
##
## Block membership:
##
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
##  1  1  1  1  1  1  1  1  2  2  1  1  1  1  2  2  1  1  2  1  2  1  2  2  2  2
## 27 28 29 30 31 32 33 34
##  2  2  2  2  2  2  2  2
##
## Reduced form blockmodel:
##
##   1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
##            Block 1    Block 2
## Block 1 0.27500000 0.03472222
## Block 2 0.03472222 0.22875817

> ##            Block 1    Block 2
> ## Block 1 0.27500000 0.03472222
> ## Block 2 0.03472222 0.22875817
> #  We can see that a much higher probability of a tie are within each faction
> #                                                          than between them.
>
> kbm <- blockmodel(knet, ec = knet %v% "Faction")$block.model
>
> # (swap packages back)
> detach(package:sna)

  ## Error:  package 'sna' is required by 'statnet' so will not be detached

> detach(package:intergraph)
> require(igraph)
```

```
>
>
> #  BlockModels are mostly used for generative models of network formation.
> #    Now we'll use those block model results:
>
> g_kar <- sample_sbm(34, pref.matrix = kbm,
+                     block.sizes = c(16,18),
+                     directed = FALSE)
> assortativity(g_kar, c(rep(1,16), rep(2,18)))

## [1] 0.8284314

> ## This compares pretty closely to the assortativity of the real karate network
> #      that we calculated above.
> ## [1] 0.7456716
>
>
> # plot(g_kar, vertex.color = c(rep(1,16), rep(2,18)), main = "Fabricated")
> # plot(karate, main = "Real")
>
> # image(g_kar[]) ## quick way to visualize the adjacency matrix.
> #   Remember that graph[] calls the adjacency matrix
>
> # image(karate[][order(as.vector(order(V(karate)$Faction))),])
```

6.6 Network Modeling

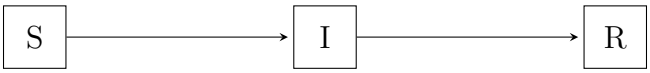
7 Stochastic Epidemic Modeling

Stochastic Epidemic Modeling, at it's core, is the ability to track, predict, and ultimiately intervene upon epidemics during their lifespan. This has obvious applications in public health, but could also be applied to the social sciences. Consider an example where voters who identify as "independent" are not "infected" with the Republican or Democrat "virus." Does framing public opinion this way yield interesting results? Perhaps! In the same way the H1N1 virus can only travel from person to person, geography too is important for political projections. Similarly for political communication online, the fact that we reference "viral" sensations and memes is forshadowing epidemic modeling's utility on the social sciences.

Certainly most epidemics are stochastic in nature, that is, acting with a degree of randomness and variation. Deterministic models have additional assumptions and too can be useful as a first-pass analysis. Epidemics usually start as a stochastic event in the first place, but at a larger volume they reach deterministic levels. However, if we wish to comment on the degree of uncertainty we have about an epidemic we requie stochastic models. For these purposes we will primarily focus on stochastic models, however the first subsection will include a deterministic model to outline some useful definitions and practices.

7.1 Deterministic Epidemic Modeling

Consider the following simplified epidemic model: the SIR model. This model assumes a simplified epidemic where susceptible (S) become infected (I) and then either become recovered or otherwise removed (R). This model can be complicated, adding conditional variables along the way (which would certainly be the case for the social sciences) but for now we will work with the following SIR model: a compartmented type model in theory.



The other thing to mention is that for these models we do not consider "time" in the same sense as timeseries or as we usually think of it. Instead, "time" passes as each generational infection occurs, someone could remain dormant with a virus for a long time, but all that matters is the conversion periods not the waiting periods inbetween. Similarly, these are Markov models which only care about the probability of infection between generations, so one's history doesn't matter.

7.1.1 Reed-Frost Model

The Reed-Frost Model is the simplest example of a deterministic epidemic model, and is typically appicable for small-n scenarios (like a small household infection for example).

Consider q to be the probability of escaping infection from a single infected individual by a given susceptible. X_i and Y_i are the number of susceptible and infectives at a given generational time.

The logic behind this model is to map out all possibilities that could occur for a particular n-size and then calculate the probability of any one possibility occurring. Or mathematically:

$$\begin{aligned}
&P(Y_{j+1} = y_{j+1} | X_0 = x_0, Y_0 = y_0, ..., X_j = x_j, Y_j = y_j) \\
&= P(Y_{j+1} = y_{j+1} | X_j = x_j, Y_j = y_j) \\
&= \binom{x_i}{y_{j+1}} (1 - q^{y_j})^{y_{j+1}} (q^{y_j})^{x_j - y_{j+1}}
\end{aligned}$$

The final result includes (in the first module) a binomial of individual infected, those who didn't escape infection $(1 - q)$ and then the compliment of this (q).

7.2 Stochastic SIR Models

8 Formal Modeling

8.1 How can Formal Modeling Help?

The late statistician George Box once said "All models are wrong, but some are useful." Formal Modeling (which I'll sometimes call game theory synonymously) can be used to spell out the the particular assumptions leading to an empirical test. Instead of writing long-hand the history, studies, and assumptions that go alongside our theory we explicitly state relevant "variables" in math notation leading right up to a logical empirical test. This has many obvious benefits, chiefly it disciplines one to be explicit with their assumptions (similar, but more involved than DAG plots from the causal inference chapter). Also, this allows for clear objections and revisions to be made by future scholarship. Indeed many models, such as Downs' "Median Voter Theorem" has been revised multiple times given the clear writing from the original mathematical formal modeling.

Of course Downs and others who use formal modeling are very frequently "wrong," but the purpose of formal modeling is to do things more systematically and with more academic rigor than prior. Works of this nature become foundational, having long shelf-lives, and become solid building blocks for future scholarship to sculpt.

8.2 Components of Game Theory

Utility, First Best, Nash Equilibrium, Comparative Static
To begin, utility is one's preferences for one thing over another. For example, preferring guns to butter:

$$u_i(Guns) = 10 \quad u_i(Butter) = 1$$

Which, note, is the same as this *utility function*:

$$u_i(Guns) = 8,675,309 \quad u_i(Butter) = 1,337$$

One's *expected utility* is the mathematical calculation of one's probability of each event happening multiplied by each utility per outcome.[15]
Considering the standard 2x2 matrix format:

		Player Y	
		A	B
Player X	A	(x, y)	(x, y)
	B	(x, y)	(x, y)

The *First Best* is the bolded sections given the best strategic decisions per potential choice by the other player.

The *Nash Equilibrium* is the "first best of the first bests" or where all players reach an equilibrium around one set of strategies – even though they may not have the highest

[15]pg. 339 of Bueno de Mesquita's "Political Economy for Public Policy has a great example of this.

		Country Y	
		Don't Arm	Arm
Country X	Don't Arm	**4, 4**	0,**3**
	Arm	**3**,0	1,1

pay-off or reward for a particular player. Note that the process here for solving a 3x3 to NxN matrix is similar to the standard 2x2. Also the Nash Equilibrium(a) can be solved algebraically as well; graphically it is the intersection of two lines which represent the strategic continuum a player has. A Nash Equilibrium(a) can be notated by any given variable: θ^*.

A variation of the Nash Equilibrium is the *subgame perfect Nash Equilibrium*, or the regular Nash Eqilibria per subgame of a longer, more dynamic, game.

Another term, the *comparative static* is the component of a model that connects our theory to our empiricism through game-theoretic reasoning. Typically the comparative static is deployed in papers to test hypotheses, arrive at novel findings, argue for an expansion of a particular model or theory, or just to describe a phenomenon with more precision than before.

8.3 Popular Games as Examples

9 Meta-Analysis

Increasingly popular in the social sciences (and beyond!) is the "analysis of analyses" or meta-analysis.[16] The practice, broadly speaking, involves the combination of many articles on a specific topic to see where the "jury stands" on the topic. This involves the combination of effect sizes, weighted by sample sizes and other important components of a papers methodology, controlling for any impending biases of the article (gender of the authors, or year the article was published as examples). While modern practice encourages the combination of effect sizes, early meta-analyses involved the combination of p-values to make theoretical statements about the given topic. These latter practices are still ongoing today, and are often met with criticism this is because of the popular misconception of p-values being a statement about seeing theory in data rather than the data being observed given the sharp null hypotheses.

That is,

$$\Pr(\text{data}|H_0\text{true}) = \text{p-values} \tag{48}$$

$$\Pr(H_0\text{true}|\text{data}) = / = \text{p-values} \tag{49}$$

Effect sizes will be calculated in the sections that follow.

9.1 Theory

In academia especially, articles are compared and contrasted against each other all time time. This usually occurs in an articles "literature review" section - often around a topic of a theoretically interesting puzzle or unanswered question that the research hopes to illuminate for the reader. However, these sorts of comparisons are often without recognition of bias, random sampling of selected readings, or without recognition of elite scholars being able to more easily market their results. In contrast to literary comparisons meta-analyses apply a more quantitative approach to article examination of a topic and often result in similar advantages.

To highlight this point, an article by Bushman and Rothstein (2012) notes that often biases occur when one group is given suggestive and attractive paper titles when others get dull scientific sounding titles. If students receive brief training in meta-analysis these biases go away. The article hits home the importance of empirical literature comparisons when something as small as the article title can bias the interpretation of an articles results.

9.2 Examples

[16] A term coined by Gene Glass as far back as 1976; however the practice of meta-analysis can be found as far back as 1904 with notable statisticians such as Pearson and Fisher.

10 Field Research Design: Qualitative Methods

10.1 How to Use Qualitative Evidence

10.2 Survey Design Psychology

10.2.1 Survey Experiments

10.2.2 The "Don't Know" Option:

10.2.3 Research on Survey Biases

10.3 Content Analysis

10.3.1 The Protocol

10.3.2 The Coding Sheet

10.3.3 Reliability Measurements

[Krippendorf's Alpha]

10.3.4 The Use of Drones

10.4 Advice, Tools, Resources

Index