Regression in R

Seth Margolis

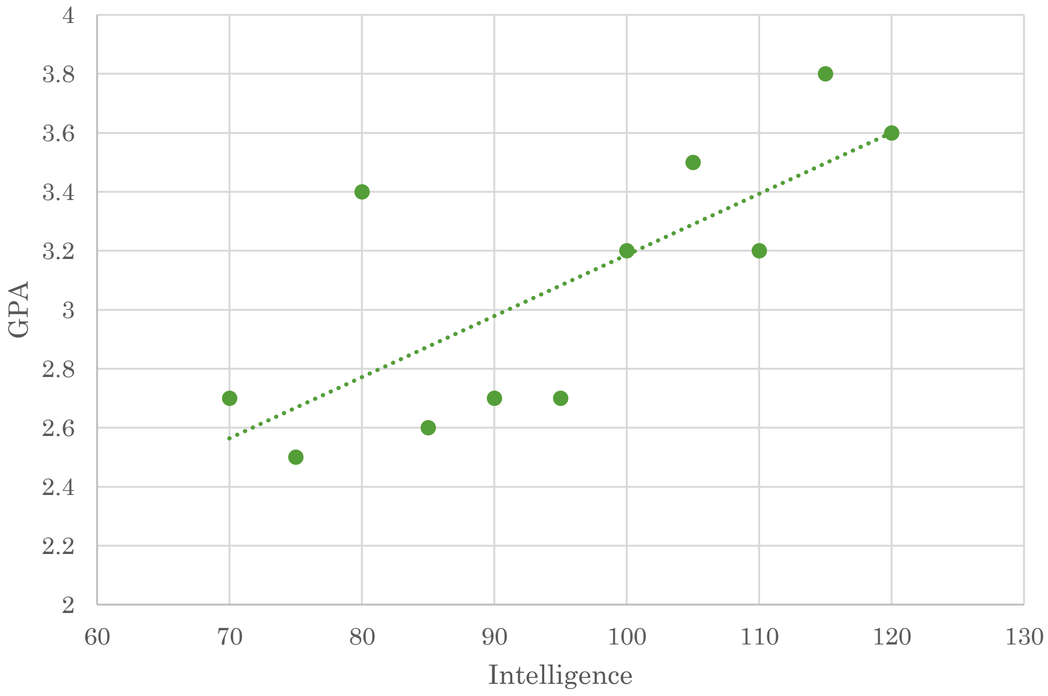
GradQuant

May 31, 2018

What is Regression Good For?

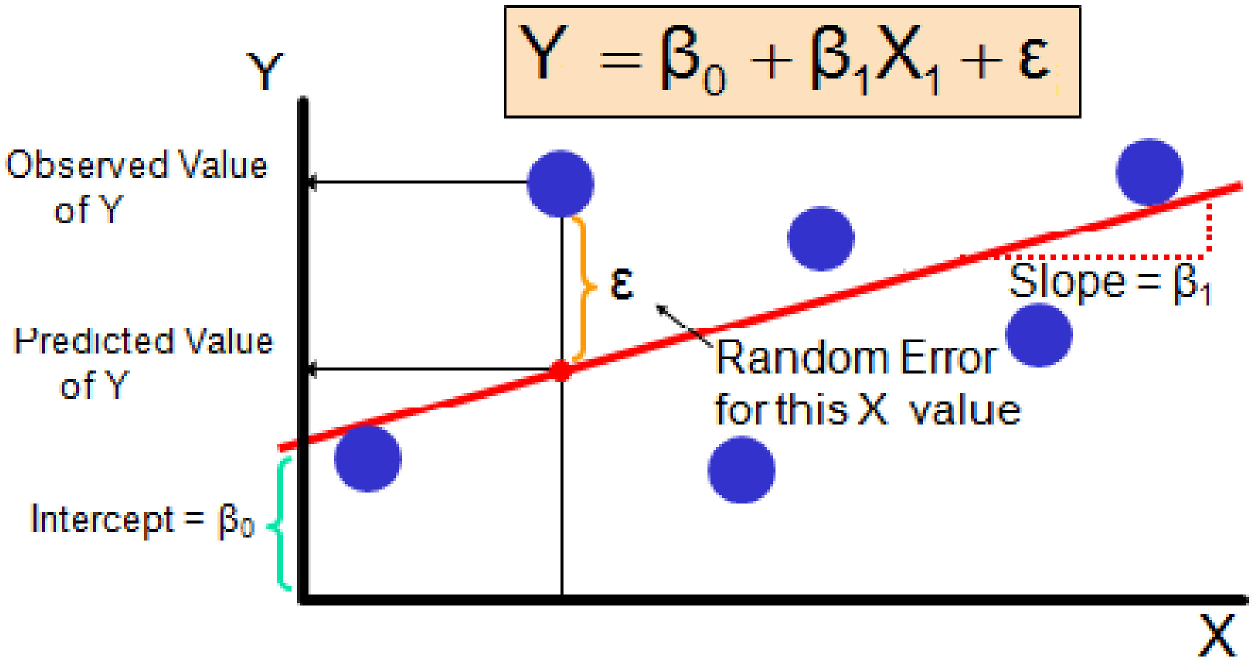
- Assessing relationships between variables
  - This probably covers most of what you do
  - Example:
    - What is the relationship between intelligence and GPA?
    - Intelligence is the independent variable, which we will call X
    - GPA is the dependent variable, which we will call Y

Person	Intelligence	GPA
1	70	2.7
2	75	2.5
3	80	3.4
4	85	2.6
5	90	2.7
6	95	2.7
7	100	3.2
8	105	3.5
9	110	3.2
10	115	3.8
11	120	3.6

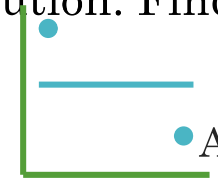


What is Regression?

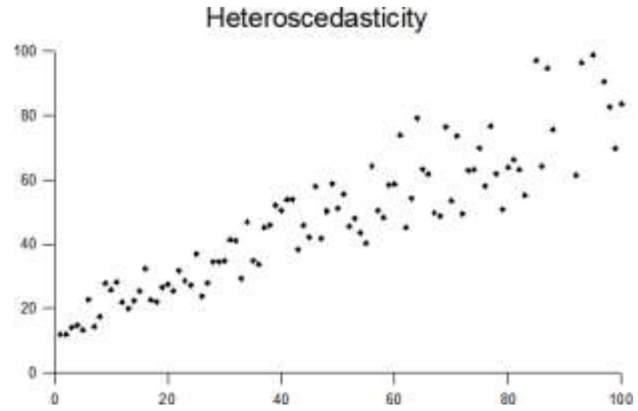
- Predicts an outcome (Y) from 1+ predictors (Xs)
- Fitting a line to data
  - Y = mx + b + error
  - Y = intercept + slope*X + error
- $Y = b_0 + b_1X_1 + \ldots + b_2X_2 + e$
- Y is one column of data
- Each X is a column of data
- b_x is the coefficient/weight for that x
- b_0 is the intercept
  - Prediction for Y when all Xs are 0
- e is error/residual
  - What is left over after prediction
  - $Y_{actual} - Y_{predicted}$
  - 1 value for each case (row of data)

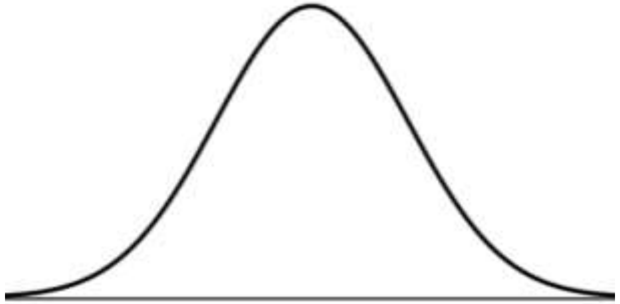


How does R select the best bs?

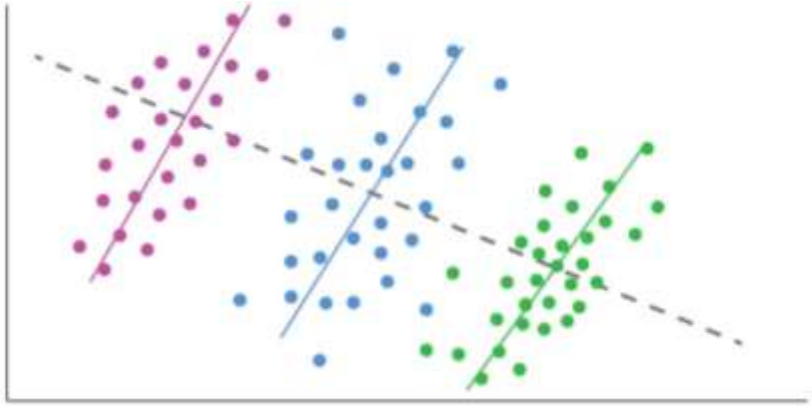
- $Y = b_0 + b_1X_1 + b_2X_2 + e$
- What would the best bs do?
- They would lead to predictions of Y that are closest to the actual Y values
- How close is one prediction from the actual value for that case (row in data)?
  - The residual/error
    - $Y_{actual}-Y_{predicted}$
  - Better bs will lead to smaller residuals/errors
- Proposed solution: Find bs that lead to the lowest average residuals/errors
- Problem:
  -  Average residual = 0
- Solution: Find bs that lead to lowest average *squared* residuals/errors
  - Ordinary *least squares* (OLS) regression
- R finds the bs that minimize the squared residuals/errors using matrix algebra

Assumptions

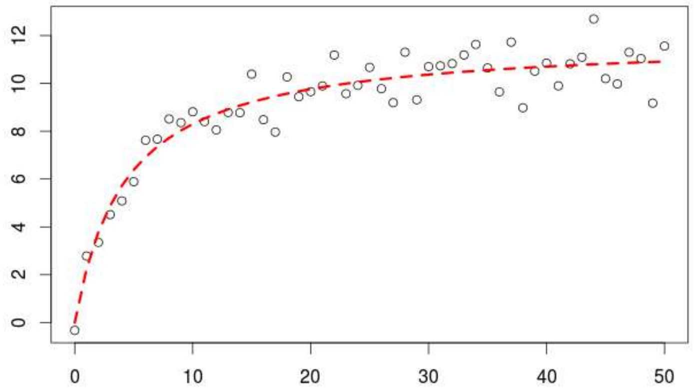




1. Homoscedasticity
   - Variance of residuals does not change at levels of X
   - When violated, bootstrap



2. Residuals/errors normally distributed
   - Can use histogram or P-P / Q-Q plot
   - When violated, bootstrap
3. Independence of residuals/errors
   - When violated, we can use multilevel modeling

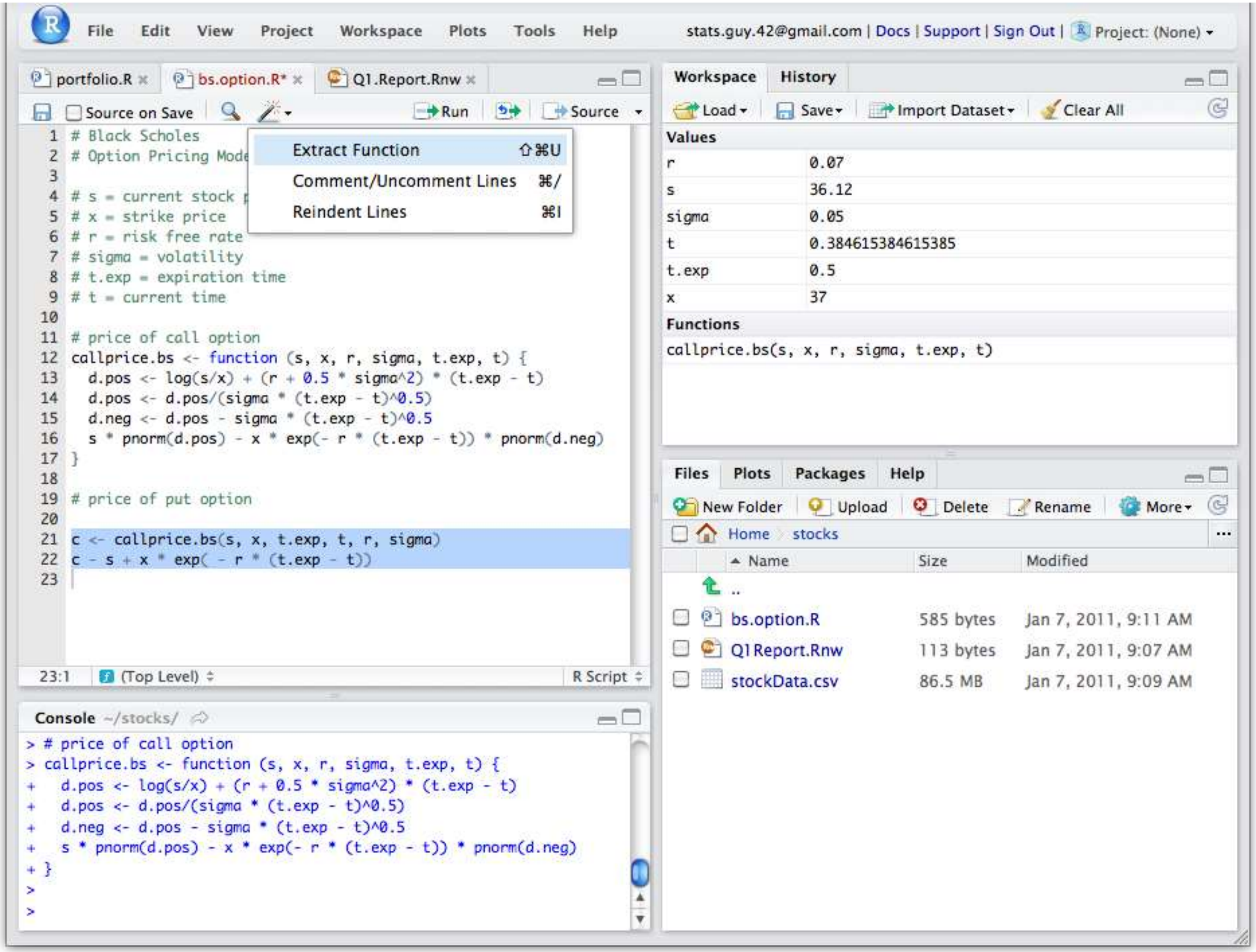


4. Linearity
   - When violated, transform Xs to meet this assumption

- Xs can have any distribution

R and RStudio

- Script
- Console
- Environment
- Object assignment
- Plotting
- Packages
- Help
- Commenting
- Data frames
- Functions and arguments



Load Data

- Usually will use
  - dataname = read.csv(“filepath”)
- We will use a dataset in the corrgram package about 322 Major League Baseball regular and substitute hitters in 1986

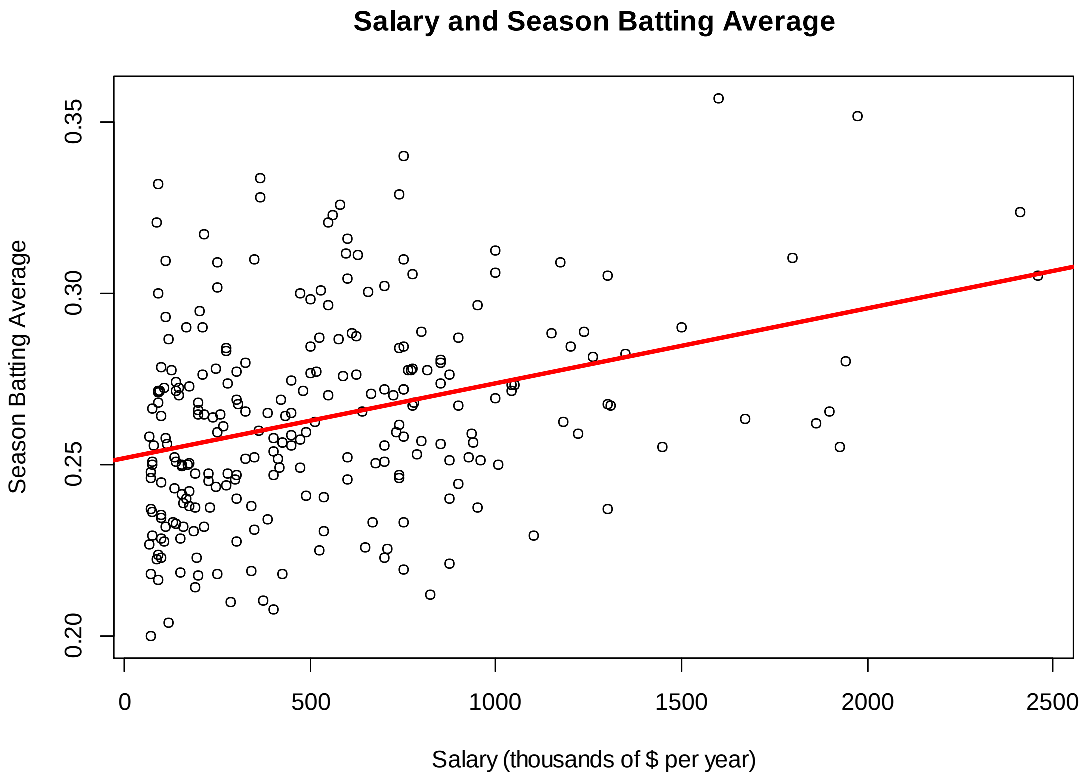
```
> #clear workspace
> #commented so that you don't accidentally do it
> # rm(list = ls())
> #load BaseballData from a package - usually will use read.csv to import a csv
> # install.packages("corrgram")
> library(corrgram)
> BaseballData = corrgram::baseball
> #this is baseball data on 322 Major League Baseball regular and substitute hitters in 1986
> #along with some statistics about their careers
```

Some Data Transformations – Not Focus of this Workshop

```
> #we now have an object in our environment called BaseballData
> dim(BaseballData)
[1] 322  22
> #this is a BaseballData frame with 322 rows (people) and 22 columns (variables)
> #let's subset the BaseballData to take only the columns we want
> colnames(BaseballData)
 [1] "Name"     "League"   "Team"     "Position" "Atbat"    "Hits"     "Homer"    "Runs"     "RBI"      "Walks"    "Years"    "Atbatc"   "Hi
tsc"
[14] "Homerc"   "Runsc"    "RBIc"     "Walksc"   "Putouts"  "Assists"  "Errors"   "Salary"   "logSal"
> BaseballData = BaseballData[,2:21]
> #renaming columns to be more interpretable
> colnames(BaseballData) = c("League", "Team", "Position", "SeasonAtBats", "SeasonHits", "SeasonHomeRuns",
+                            "SeasonRuns", "SeasonRBIs", "SeasonWalks", "CareerYears", "CareerAtBats", "CareerHits", "CareerHomeRuns",
+                            "CareerRuns", "CareerRBIs", "CareerWalks", "SeasonPutouts", "SeasonAssists", "SeasonErrors",
+                            "SeasonSalary")
> #let's remove people with less than 100 SeasonAtBats - some of them are outliers
> BaseballData = BaseballData[BaseballData$SeasonAtBats > 100,]
> #removing utility players - those who play multiple positions
> BaseballData = BaseballData[BaseballData$Position != "UT",]
> #creating new variables from other variables
> BaseballData$SeasonBattingAvg = BaseballData$SeasonHits / BaseballData$SeasonAtBats
> BaseballData$CareerHitsPerYear = BaseballData$CareerHits / BaseballData$CareerYears
> #let's remove cases with missing BaseballData
> BaseballData = na.omit(BaseballData)
> dim(BaseballData)
[1] 251  22
```

1 Continuous Predictor – Scatterplot

- $Y = b_0 + b_1X_1 + e$
- Scatterplot can be quite revealing
- Want to make sure the relationship looks linear
- Otherwise you will probably want to transform your predictor
  - Later in workshop




```
> #scatterplot
> plot(BaseballData$SeasonSalary, BaseballData$SeasonBattingAvg, main = "Salary and Season Batting Average",
+       xlab = "Salary (thousands of $ per year)", ylab = "Season Batting Average")
> #adding a line of best fit - the regression line
> abline(lm(SeasonBattingAvg ~ SeasonSalary, BaseballData), col="red", lwd = 3)
```

1 Continuous Predictor – Model

```
> #running regression model with one predictor
> SimpleModel = lm(SeasonBattingAvg ~ SeasonSalary, BaseballData)
> #getting most of the important information about our regression model
> summary(SimpleModel)

Call:
lm(formula = SeasonBattingAvg ~ SeasonSalary, data = BaseballData)

Residuals:
      Min        1Q    Median        3Q       Max 
-0.057810 -0.019429 -0.002813  0.016307  0.077840 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)  2.520e-01  2.731e-03  92.249  < 2e-16 ***
SeasonSalary 2.179e-05  3.912e-06   5.571 6.55e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.02768 on 249 degrees of freedom
Multiple R-squared:  0.1108,	Adjusted R-squared:  0.1073 
F-statistic: 31.04 on 1 and 249 DF,  p-value: 6.552e-08
```

For each 1-unit increase in SeasonSalary, the expected SeasonBattingAvg increases by .000022

- Estimate = b = regression coefficient
- Standard error = standard deviation of sampling distribution
  - Provides confidence intervals
  - Most affected by N (sample size)
- t = b / se
- t and df -> p
  - df = N-k-1
  - N = sample size
  - k = number of predictors
- Multiple R^2 = correlation of predicted and actual values squared
  - p-value is p from F (lower right)
- Adjusted R^2 adjusts R^2 based on number of predictors

Standardized Coefficients

- What if all values were standardized before entering model?
- Z = (X – Mean) / SD
- βs can be interpreted like correlations
  - -1 to 1
  - Called standardized regression coefficients

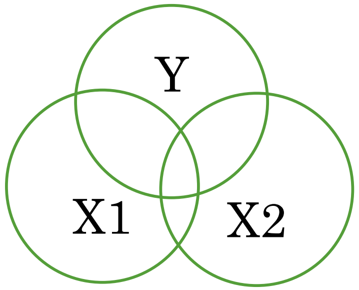
```
> #but let's get the betas too
> # install.packages("QuantPsyc")
> library(QuantPsyc)
> lm.beta(SimpleModel)
SeasonSalary
   0.3329203
> #correlation
> cor.test(BaseballData$SeasonSalary, BaseballData$SeasonBattingAvg)

        Pearson's product-moment correlation

data:  BaseballData$SeasonSalary and BaseballData$SeasonBattingAvg
t = 5.5712, df = 249, p-value = 6.552e-08
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.2180913 0.4386572
sample estimates:
      cor
0.3329203
```

For each 1 SD increase in SeasonSalary, SeasonBattingAvg increases by .33 SDs

Multiple Continuous Predictors

```
> #running regression model with multiple predictors
> TwoPredModel = lm(SeasonBattingAvg ~ CareerYears + CareerHitsPerYear, BaseballData)
> summary(TwoPredModel)

Call:
lm(formula = SeasonBattingAvg ~ CareerYears + CareerHitsPerYear,
    data = BaseballData)

Residuals:
      Min        1Q    Median        3Q       Max 
-0.062431 -0.018775 -0.000904  0.016265  0.079860 

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)    
(Intercept)        0.2378233  0.0044869  53.004  < 2e-16 ***
CareerYears       -0.0006345  0.0003846  -1.650      0.1    
CareerHitsPerYear  0.0003341  0.0000461   7.247 5.38e-12 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.02668 on 248 degrees of freedom
Multiple R-squared:  0.1768,	Adjusted R-squared:  0.1702 
F-statistic: 26.64 on 2 and 248 DF,  p-value: 3.319e-11

> lm.beta(TwoPredModel)
      CareerYears CareerHitsPerYear 
       -0.1010549         0.4439864 
```

- Coefficients are partial coefficients
  - Contribution of that variable holding other variables constant
  - Unique contribution of that variable over and above other variables



Adding a Predictor

```
> #adding a predictor
> ThreePredModel = lm(SeasonBattingAvg ~ CareerYears + CareerHitsPerYear + SeasonSalary, BaseballData)
> summary(ThreePredModel)

Call:
lm(formula = SeasonBattingAvg ~ CareerYears + CareerHitsPerYear + 
    SeasonSalary, data = BaseballData)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.05366 -0.01960 -0.00071  0.01537  0.08364 

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)    
(Intercept)        2.405e-01  4.572e-03  52.611  < 2e-16 ***
CareerYears       -9.911e-04  4.069e-04  -2.436   0.0156 *  
CareerHitsPerYear  2.577e-04  5.503e-05   4.682 4.69e-06 ***
SeasonSalary       1.268e-05  5.106e-06   2.483   0.0137 *  
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.02641 on 247 degrees of freedom
Multiple R-squared:  0.1969,	Adjusted R-squared:  0.1871 
F-statistic: 20.18 on 3 and 247 DF,  p-value: 9.884e-12

> lm.beta(ThreePredModel)
      CareerYears CareerHitsPerYear      SeasonSalary 
       -0.1578539         0.3424447         0.1936466 
```

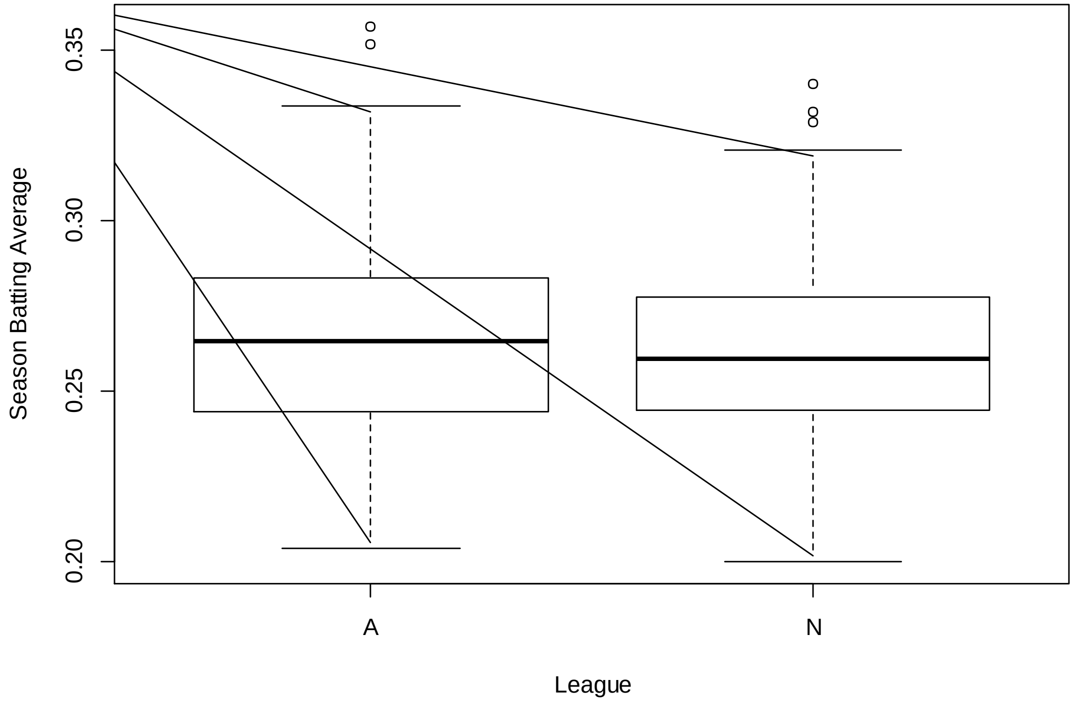
For each 1 unit increase in CareerYears, the expected SeasonBattingAvg increases by -.00099, holding other variables constant

Adding a Predictor

```
> #let's see how much R-squared increased by adding a predictor
> summary(ThreePredModel)$r.squared - summary(TwoPredModel)$r.squared
[1] 0.02004029
> #may be more interpetable to square-root that number: the semi-partial correlation
> sqrt(summary(ThreePredModel)$r.squared - summary(TwoPredModel)$r.squared)
[1] 0.1415637
> #model comparison
> anova(TwoPredModel, ThreePredModel)
Analysis of Variance Table

Model 1: SeasonBattingAvg ~ CareerYears + CareerHitsPerYear
Model 2: SeasonBattingAvg ~ CareerYears + CareerHitsPerYear + SeasonSalary
  Res.Df     RSS Df Sum of Sq      F  Pr(>F)
1    248 0.17658
2    247 0.17229  1  0.004299 6.1633 0.01371 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
> #same p-value of individual predictor
```

Categorical Predictors: Dichotomous

```
> #One dichotomous predictor: equivalent to a t-test
> #boxplot
> plot(BaseballData$League, BaseballData$SeasonBattingAvg, xlab = "League", ylab = "Season Batting Average")
> #regresion model
> DichotPredModel = lm(SeasonBattingAvg ~ League, BaseballData)
> summary(DichotPredModel)

Call:
lm(formula = SeasonBattingAvg ~ League, data = BaseballData)

Residuals:
      Min        1Q    Median        3Q       Max 
-0.061725 -0.020409 -0.000931  0.016290  0.091568 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept)  0.265329   0.002531 104.840   <2e-16 ***
LeagueN     -0.003604   0.003707  -0.972    0.332    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.0293 on 249 degrees of freedom
Multiple R-squared:  0.003783,	Adjusted R-squared:  -0.0002182 
F-statistic: 0.9455 on 1 and 249 DF,  p-value: 0.3318

> lm.beta(DichotPredModel)
     League 
-0.06150317 
Warning message:
In var(if (is.vector(x) || is.factor(x)) x else as.double(x), na.rm = na.rm) :
  Calling var(x) on a factor x is deprecated and will become an error.
  Use something like 'all(duplicated(x)[-1L])' to test for a constant vector.
> #this is the correlation, which is a measure of effect size/consistency
```



The NL has a batting average .0036 lower than the AL

Categorical Predictors: 3+ Levels

- Dummy Coding
  - Pick a reference group
  - Estimate that group's level of Y as the intercept
  - For all other groups, estimate how different their level of Y is, compared to the dummy group as the bs
  - I usually prefer this over effects coding
- Effects Coding
  - Estimate the grand mean (mean of all groups) as the intercept
  - For all other groups (except 1), estimate how different their level of Y is, compared to the grand mean as the bs

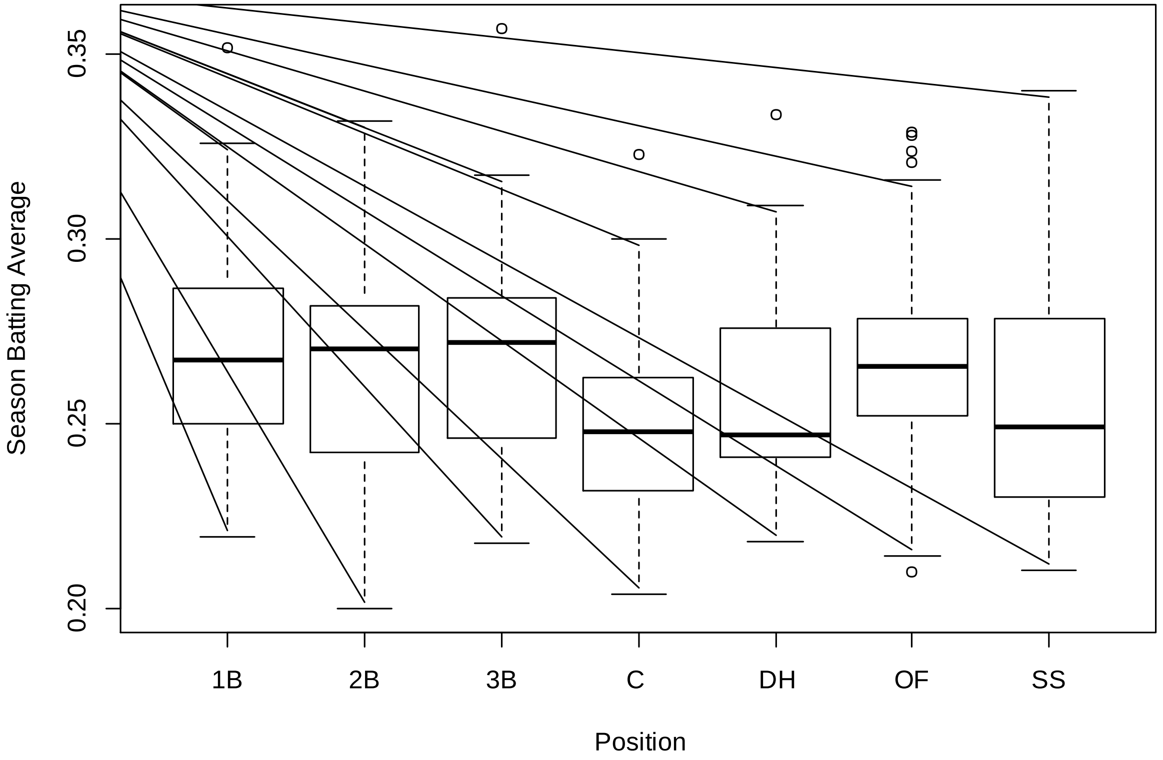
- Dummy Coding
  - 1 2 3 4 5 6
  - 1B 1 0 0 0 0 0
  - 2B 0 1 0 0 0 0
  - 3B 0 0 1 0 0 0
  - C 0 0 0 1 0 0
  - DH 0 0 0 0 1 0
  - OF 0 0 0 0 0 0
  - SS 0 0 0 0 0 1
- Effects Coding
  - 1 2 3 4 5 6
  - 1B 1 0 0 0 0 0
  - 2B 0 1 0 0 0 0
  - 3B 0 0 1 0 0 0
  - C 0 0 0 1 0 0
  - DH 0 0 0 0 1 0
  - OF -1-1-1-1-1-1
  - SS 0 0 0 0 0 1

Categorical Predictors: 3+ Levels

```
> #multiple-categorical predictor
> #what values could Position have?
> levels(BaseballData$Position)
[1] "1B" "2B" "3B" "C " "OF" "DH" "SS" "UT"
> #why is UT still there? We deleted the data but the column can still take that value
> #can fix by converting to character vector then back to factor
> BaseballData$Position = as.factor(as.character(BaseballData$Position))
> levels(BaseballData$Position)
[1] "1B" "2B" "3B" "C " "DH" "OF" "SS"
> #see fixed
> #now let's see how many people are at each position
> # install.packages("plyr")
> library(plyr)
> count(BaseballData$Position)
   x freq
1 1B   25
2 2B   28
3 3B   34
4 C    30
5 DH   13
6 OF   94
7 SS   27
> #what coding scheme is it using
> contrasts(BaseballData$Position)
   2B 3B C  DH OF SS
1B  0  0  0  0  0  0
2B  1  0  0  0  0  0
3B  0  1  0  0  0  0
C   0  0  1  0  0  0
DH  0  0  0  1  0  0
OF  0  0  0  0  1  0
SS  0  0  0  0  0  1
> #creating dummy codes with OF as reference group
> contrasts(BaseballData$Position) = contr.treatment(7, base = 6)
```

Categorical Predictors: 3+ Levels

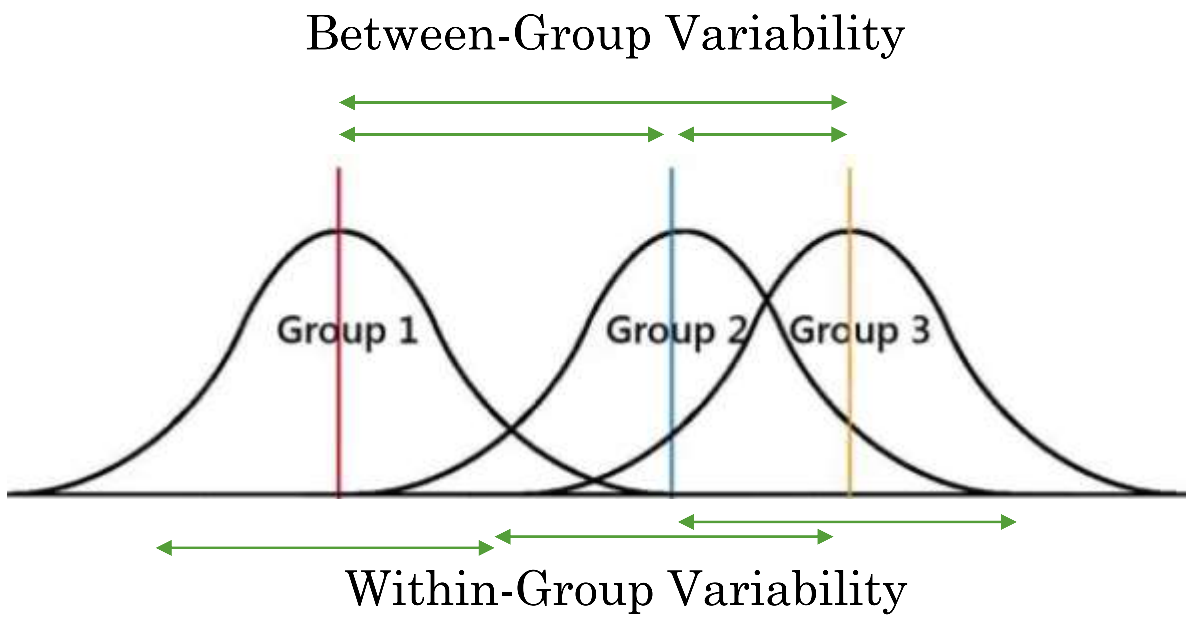
```
> #boxplot
> plot(BaseballData$Position, BaseballData$SeasonBattingAvg, xlab = "Position", ylab = "Season Batting Average")
> #regression model
> CategoricalModel = lm(SeasonBattingAvg ~ Position, BaseballData)
> summary(CategoricalModel)

Call:
lm(formula = SeasonBattingAvg ~ Position, data = BaseballData)

Residuals:
      Min        1Q    Median        3Q       Max
-0.064216 -0.018283 -0.002458  0.015893  0.087049

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.267538   0.002963  90.303  < 2e-16 ***
Position1    0.001983   0.006464   0.307  0.75928
Position2   -0.003321   0.006184  -0.537  0.59172
Position3    0.002309   0.005748   0.402  0.68821
Position4   -0.019317   0.006023  -3.207  0.00152 **
Position5   -0.005433   0.008500  -0.639  0.52332
Position7   -0.013372   0.006272  -2.132  0.03400 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.02872 on 244 degrees of freedom
Multiple R-squared:  0.06153,	Adjusted R-squared:  0.03845
F-statistic: 2.666 on 6 and 244 DF,  p-value: 0.01594

> lm.beta(CategoricalModel)
 Position1  Position2  Position3  Position4  Position5  Position7
 0.1318510 -0.2208422  0.1535659 -1.2844380 -0.3612325 -0.8891539
Warning message:
In var(if (is.vector(x) || is.factor(x)) x else as.double(x), na.rm = na.rm) :
  Calling var(x) on a factor x is deprecated and will become an error.
  Use something like 'all(duplicated(x)[-1L])' to test for a constant vector.
> #betas with multi-categorical predictors not very useful
```



Catchers have a batting average .019 lower than outfielders

Comparing to ANOVA

```
> #anova model
> CategoricalANOVA = aov(SeasonBattingAvg ~ Position, BaseballData)
> summary(CategoricalANOVA)
             Df Sum Sq   Mean Sq F value Pr(>F)  
Position      6 0.0132 0.0021998   2.666 0.0159 *
Residuals   244 0.2013 0.0008251                 
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
> # install.packages("sjstats")
> library(sjstats)
> eta_sq(CategoricalANOVA)
# A tibble: 1 x 2
  term      etasq
  <chr>     <dbl>
1 Position 0.0615
> #they are the same!
```



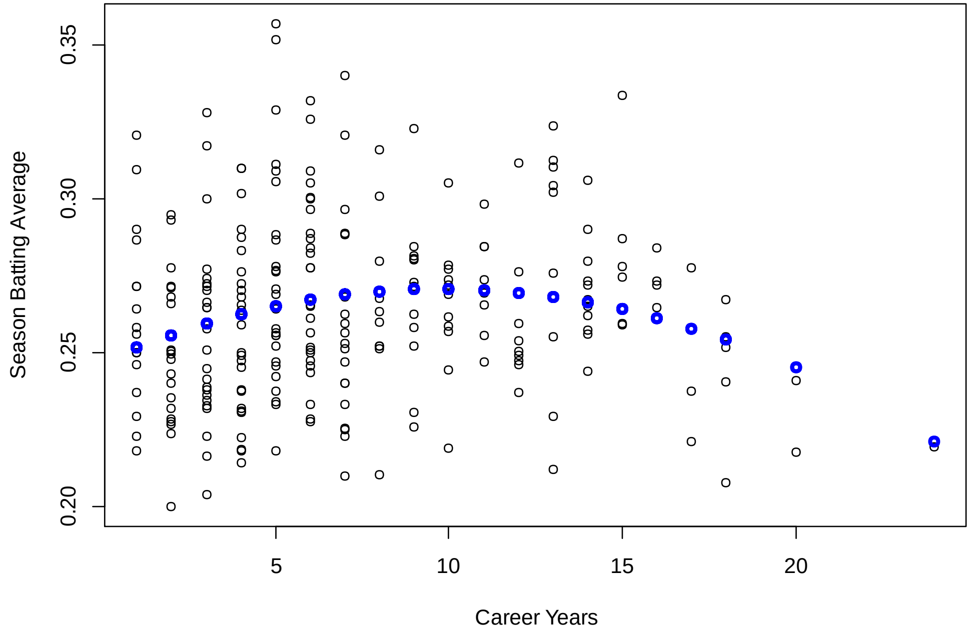
ANCOVA – Categorical and Continuous Predictors

```
> #ANCOVA
> ANCOVAModel = lm(SeasonBattingAvg ~ CareerYears + CareerHitsPerYear + SeasonSalary + League + Position, BaseballData)
> summary(ANCOVAModel)

Call:
lm(formula = SeasonBattingAvg ~ CareerYears + CareerHitsPerYear +
    SeasonSalary + League + Position, data = BaseballData)

Residuals:
      Min        1Q    Median        3Q       Max
-0.049853 -0.020151 -0.001458  0.015171  0.078672

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)        2.470e-01  5.492e-03  44.977  < 2e-16 ***
CareerYears       -1.100e-03  4.350e-04  -2.529  0.01207 *
CareerHitsPerYear  2.274e-04  5.816e-05   3.910  0.00012 ***
SeasonSalary       1.415e-05  5.195e-06   2.725  0.00691 **
LeagueN           -2.579e-03  3.418e-03  -0.755  0.45119
Position1         -2.551e-03  5.974e-03  -0.427  0.66982
Position2         -3.387e-03  5.673e-03  -0.597  0.55109
Position3          3.745e-03  5.329e-03   0.703  0.48284
Position4         -1.031e-02  5.822e-03  -1.771  0.07780 .
Position5          1.864e-03  8.393e-03   0.222  0.82442
Position7         -1.193e-02  5.754e-03  -2.073  0.03923 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.02627 on 240 degrees of freedom
Multiple R-squared:  0.2278,	Adjusted R-squared:  0.1956
F-statistic: 7.078 on 10 and 240 DF,  p-value: 9.614e-10

> lm.beta(ANCOVAModel)
      CareerYears CareerHitsPerYear      SeasonSalary           LeagueN         Position1         Position2         Position3         Positi
on4
      -0.17523338        0.30223085        0.21624455       -0.04400971       -0.16959836       -0.53939126        4.97741143      -157.53274
505
        Position5         Position7
       0.03181022       -0.79315660
Warning messages:
1: In var(if (is.vector(x) || is.factor(x)) x else as.double(x), na.rm = na.rm) :
  Calling var(x) on a factor x is deprecated and will become an error.
  Use something like 'all(duplicated(x)[-1L])' to test for a constant vector.
2: In var(if (is.vector(x) || is.factor(x)) x else as.double(x), na.rm = na.rm) :
  Calling var(x) on a factor x is deprecated and will become an error.
  Use something like 'all(duplicated(x)[-1L])' to test for a constant vector.
```

Catchers have a batting average .010 lower than outfielders, holding other variables constant

Non-linear Relationships

```
> #non-linear relationships
> plot(BaseballData$CareerYears, BaseballData$SeasonBattingAvg, xlab = "Career Years", ylab = "Season Batting Average")
> lines(lowess(BaseballData$CareerYears, BaseballData$SeasonBattingAvg), col="blue", lwd = 3)
> BaseballData$CareerYearsCentered = scale(BaseballData$CareerYears, scale = F)
> BaseballData$CareerYearsCenteredSq = BaseballData$CareerYearsCentered^2
> QuadraticModel = lm(SeasonBattingAvg ~ CareerYearsCentered + CareerYearsCenteredSq, BaseballData)
> summary(QuadraticModel)

Call:
lm(formula = SeasonBattingAvg ~ CareerYearsCentered + CareerYearsCenteredSq,
    data = BaseballData)

Residuals:
      Min        1Q    Median        3Q       Max
-0.059578 -0.020068 -0.001751  0.014445  0.091863

Coefficients:
                        Estimate Std. Error t value Pr(>|t|)
(Intercept)            2.690e-01  2.373e-03 113.330  < 2e-16 ***
CareerYearsCentered    1.291e-03  4.805e-04   2.687 0.007687 **
CareerYearsCenteredSq -2.452e-04  7.077e-05  -3.466 0.000623 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.02869 on 248 degrees of freedom
Multiple R-squared:  0.04857,   Adjusted R-squared:  0.04089
F-statistic: 6.329 on 2 and 248 DF,  p-value: 0.002085

> lm.beta(QuadraticModel)
  CareerYearsCentered CareerYearsCenteredSq
            0.2056725            -0.2652163
> plot(BaseballData$CareerYears, BaseballData$SeasonBattingAvg, xlab = "Career Years", ylab = "Season Batting Average")
> points(BaseballData$CareerYears, QuadraticModel$fitted.values, col="blue", lwd = 3)
```

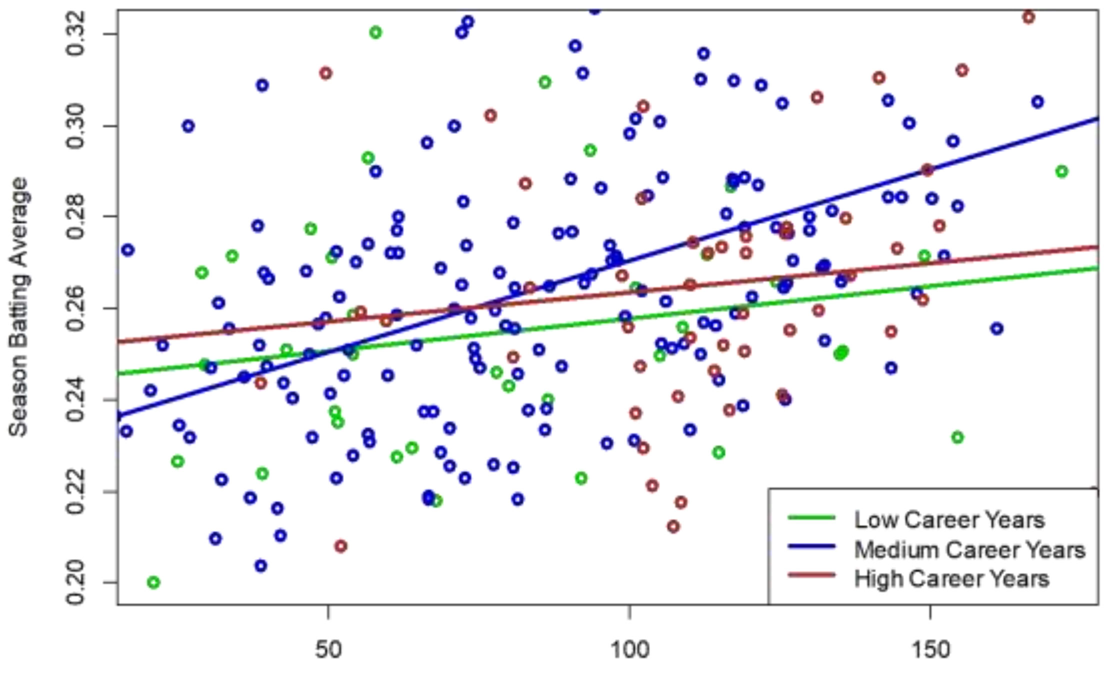


Interactions

```
> #interactions
> BaseballData$CareerHitsPerYearCentered = scale(BaseballData$CareerHitsPerYear, scale = F)
> InteractionModel = lm(SeasonBattingAvg ~ CareerYears*CareerHitsPerYearCentered, BaseballData)
> summary(InteractionModel)

Call:
lm(formula = SeasonBattingAvg ~ CareerYears * CareerHitsPerYearCentered,
    data = BaseballData)

Residuals:
     Min       1Q   Median       3Q      Max
-0.06182 -0.01856 -0.00062  0.01613  0.07297

Coefficients:
                                        Estimate Std. Error t value Pr(>|t|)
(Intercept)                            2.672e-01  3.238e-03  82.499  < 2e-16 ***
CareerYears                           -3.064e-04  4.107e-04  -0.746   0.4564
CareerHitsPerYearCentered              4.667e-04  7.637e-05   6.111 3.84e-09 ***
CareerYears:CareerHitsPerYearCentered -2.163e-05  9.975e-06  -2.168   0.0311 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.02649 on 247 degrees of freedom
Multiple R-squared:  0.1922,	Adjusted R-squared:  0.1824
F-statistic: 19.59 on 3 and 247 DF,  p-value: 1.997e-11

> lm.beta(InteractionModel)
                          CareerYears             CareerHitsPerYearCentered CareerYears:CareerHitsPerYearCentered
                         -0.048793187                           0.620179291                          -0.003445064
Warning message:
In b * sx : longer object length is not a multiple of shorter object length
```

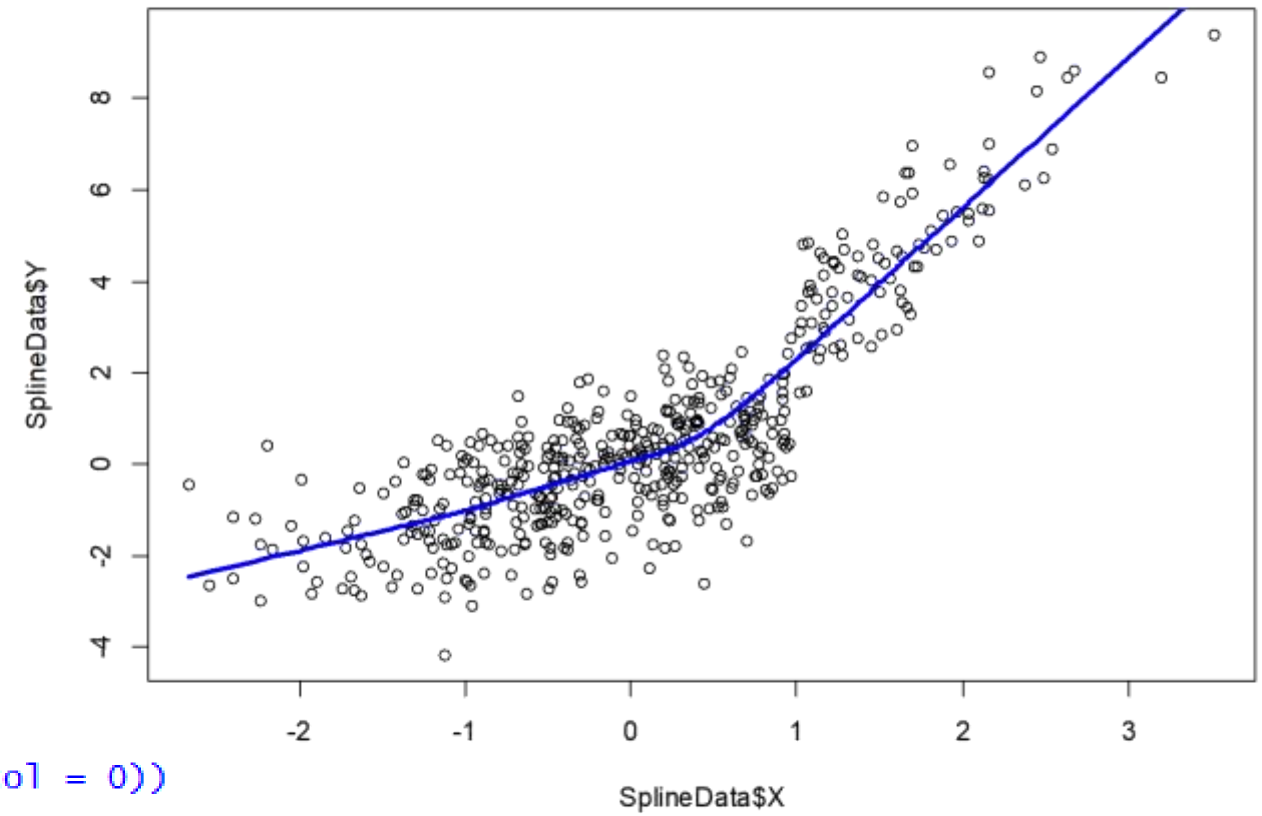
- Interaction = effect of X_1 depends on level of X_2
  - If that is true, opposite is true too
- $Y = b_0 + b_1X_1 + b_2X_2 + b_3X_1X_2$
- $Y = b_0 + b_1X_1 + (b_2 + b_3X_1)X_2$
  - b_3 = For each 1-unit increase in X how much does b_2 increase
  - And vice-versa
- Can have 3-way, 4-way, etc. interactions

Interaction Plot




```
> #interaction plot
> InteractionData = BaseballData[,c("SeasonBattingAvg", "CareerYears", "CareerHitsPerYear")]
> InteractionData$CareerYearsCat = "medium"
> InteractionData$CareerYearsCat[InteractionData$CareerYears < mean(InteractionData$CareerYears)-sd(InteractionData$CareerYears)] = "low"
> InteractionData$CareerYearsCat[InteractionData$CareerYears > mean(InteractionData$CareerYears)+sd(InteractionData$CareerYears)] = "high"
> plot(InteractionData$CareerHitsPerYear[InteractionData$CareerYearsCat == "low"],
+      InteractionData$SeasonBattingAvg[InteractionData$CareerYearsCat == "low"], col = "green3", lwd = 3,
+      xlab = "Career Hits Per Year", ylab = "Season Batting Average")
> abline(lm(InteractionData$SeasonBattingAvg[InteractionData$CareerYearsCat == "low"] ~ InteractionData$CareerHitsPerYear[InteractionData$CareerYearsCat == "low"]),
+        col = "green3", lwd = 3)
> points(InteractionData$CareerHitsPerYear[InteractionData$CareerYearsCat == "medium"], InteractionData$SeasonBattingAvg[InteractionData$CareerYearsCat == "medium"],
+        col = "blue3", lwd = 3)
> abline(lm(InteractionData$SeasonBattingAvg[InteractionData$CareerYearsCat == "medium"] ~ InteractionData$CareerHitsPerYear[InteractionData$CareerYearsCat == "medium"]),
+        col = "blue3", lwd = 3)
> points(InteractionData$CareerHitsPerYear[InteractionData$CareerYearsCat == "high"], InteractionData$SeasonBattingAvg[InteractionData$CareerYearsCat == "high"],
+        col = "brown", lwd = 3)
> abline(lm(InteractionData$SeasonBattingAvg[InteractionData$CareerYearsCat == "high"] ~ InteractionData$CareerHitsPerYear[InteractionData$CareerYearsCat == "high"]),
+        col = "brown", lwd = 3)
> legend("bottomright", legend=c("Low Career Years", "Medium Career Years", "High Career Years"), col=c("green3", "blue3", "brown"),
+        lty = 1, lwd = 3)
```

Spline Regression – Making Data




```
> #creating some data
> set.seed(pi)
> SplineData = data.frame(matrix(nrow = 500, ncol = 0))
> SplineData$X = rnorm(500)
> SplineData$Y = SplineData$X + rnorm(500)
> SplineData$Y[SplineData$X>1] = SplineData$Y[SplineData$X>1] + 2*SplineData$X[SplineData$X>1]
> #plotting that data with a lowess curve
> plot(SplineData$X, SplineData$Y)
> lines(lowess(SplineData$X, SplineData$Y), col="blue", lwd = 3)
> #knot is at 1 so will make two variables (because two spline lines) based on that info
> SplineData$X1 = SplineData$X
> SplineData$X2 = SplineData$X
> #if a value of X1 is above the knot, make it the knot value
> SplineData$X1[SplineData$X1 > 1] = 1
> #if a value of X2 is below the knot, make it 0
> SplineData$X2[SplineData$X2 < 1] = 0
> #if a value of X2 is above the knot, subtract out the knot value
> SplineData$X2[SplineData$X2 > 1] = SplineData$X2[SplineData$X2 > 1]-1
```

Spline Regression – Model

```
> #regression model
> SplineModel = lm(Y ~ X1 + X2, SplineData)
> summary(SplineModel)

Call:
lm(formula = Y ~ X1 + X2, data = SplineData)

Residuals:
    Min      1Q  Median      3Q     Max 
-3.3407 -0.6900  0.0126  0.6683  3.2474 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  0.18755    0.05179   3.621 0.000323 ***
X1           1.22687    0.06155  19.932  < 2e-16 ***
X2           4.27701    0.16688  25.630  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.072 on 497 degrees of freedom
Multiple R-squared:  0.7831,    Adjusted R-squared:  0.7822 
F-statistic: 897.1 on 2 and 497 DF,  p-value: < 2.2e-16

> lm.beta(SplineModel)
       X1        X2 
0.4583233 0.5893404 
> #betas not very useful
```

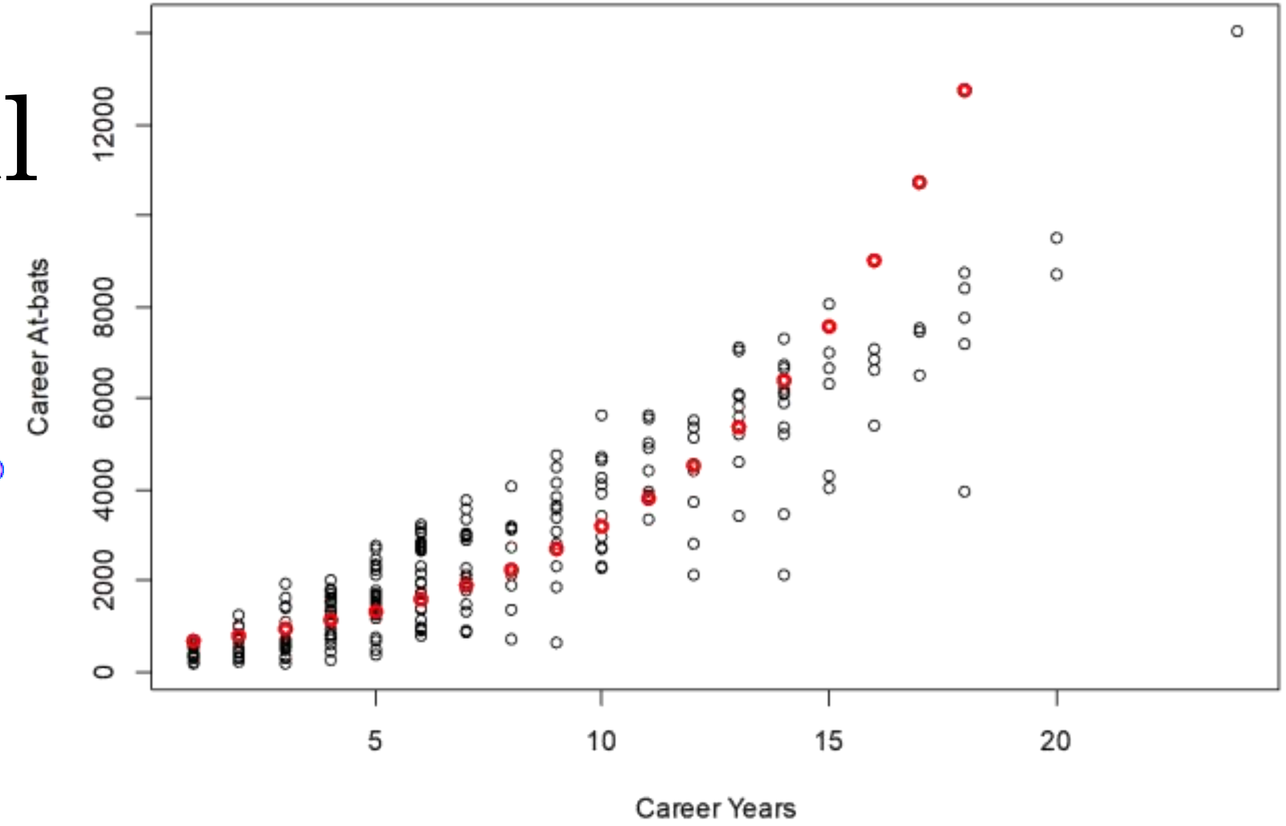
bs are slopes before and after knot

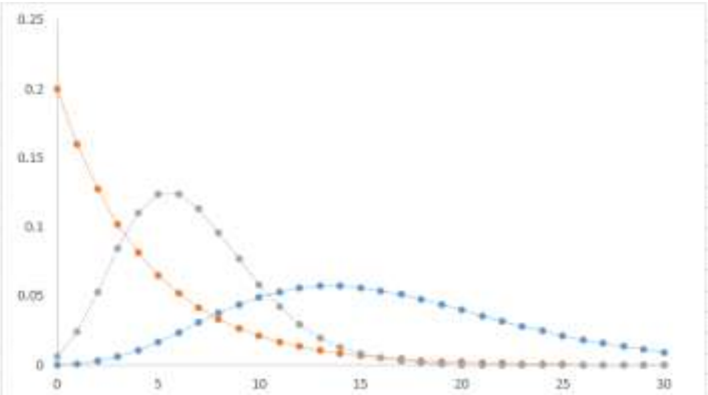
Count Outcome

```
> #count outcome - negative binomial regression
> # install.packages("psych")
> library(psych)
> describe(BaseballData)
                        vars   n    mean      sd  median trimmed     mad    min      max    range  skew kurtosis     se
League*                    1 251    1.47    0.50    1.00    1.46    0.00   1.00     2.00     1.00  0.13    -1.99   0.03
Team*                      2 251   13.79    7.47   14.00   13.84   10.38   1.00    26.00    25.00 -0.01    -1.25   0.47
Position*                  3 251    4.47    1.95    5.00    4.58    1.48   1.00     7.00     6.00 -0.43    -1.22   0.12
SeasonAtBats               4 251  412.91  142.71  419.00  415.48  171.98 126.00   687.00   561.00 -0.13    -1.13   9.01
SeasonHits                 5 251  110.35   44.18  110.00  108.98   50.41  27.00   238.00   211.00  0.26    -0.65   2.79
SeasonHomeRuns             6 251   11.99    8.78   10.00   11.14    8.90   0.00    40.00    40.00  0.73    -0.30   0.55
SeasonRuns                 7 251   56.10   25.16   54.00   55.03   29.65   8.00   130.00   122.00  0.35    -0.67   1.59
SeasonRBIs                 8 251   52.89   25.57   48.00   51.15   26.69   8.00   121.00   113.00  0.54    -0.49   1.61
SeasonWalks                9 251   42.14   21.50   38.00   40.69   23.72   5.00   105.00   100.00  0.54    -0.51   1.36
CareerYears               10 251    7.16    4.67    6.00    6.71    4.45   1.00    24.00    23.00  0.85     0.10   0.29
CareerAtBats              11 251 2619.59 2256.24 1928.00 2300.86 1857.70 181.00 14053.00 13872.00  1.35     2.20 142.41
CareerHits                12 251  712.29  640.56  506.00  616.95  512.98  42.00  4256.00  4214.00  1.50     3.23  40.43
CareerHomeRuns            13 251   69.75   82.99   40.00   52.73   45.96   0.00   548.00   548.00  2.17     5.77   5.24
CareerRuns                14 251  358.20  330.74  247.00  307.06  244.63  16.00  2165.00  2149.00  1.56     3.27  20.88
CareerRBIs                15 251  327.55  320.24  226.00  269.90  223.87   9.00  1659.00  1650.00  1.52     1.88  20.21
CareerWalks               16 251  259.15  264.64  172.00  210.97  170.50   8.00  1566.00  1558.00  1.88     4.14  16.70
SeasonPutouts             17 251  295.84  283.18  227.00  239.00  155.67   0.00  1377.00  1377.00  2.03     3.92  17.87
SeasonAssists             18 251  119.46  147.69   43.00   94.19   60.79   0.00   492.00   492.00  1.14    -0.05   9.32
SeasonErrors              19 251    8.67    6.70    7.00    7.95    5.93   0.00    32.00    32.00  0.92     0.21   0.42
SeasonSalary              20 251  536.71  447.50  425.00  468.12  407.71  68.00  2460.00  2392.00  1.51     2.77  28.25
SeasonBattingAvg          21 251    0.26    0.03    0.26    0.26    0.03   0.20     0.36     0.16  0.45     0.06   0.00
CareerHitsPerYear         22 251   90.90   38.93   92.40   90.44   45.57  12.25   195.67   183.42  0.09    -0.71   2.46
CareerYearsCentered       23 251    0.00    4.67   -1.16   -0.45    4.45  -6.16    16.84    23.00  0.85     0.10   0.29
CareerYearsCenteredSq     24 251   21.68   31.68    9.98   15.31   12.81   0.03   283.61   283.58  3.77    21.30   2.00
CareerHitsPerYearCentered 25 251    0.00   38.93    1.50   -0.46   45.57 -78.65   104.77   183.42  0.09    -0.71   2.46
> #could use poisson regression because means and standard devations look equal
> #but why have an assumption that can never really be true
```

Count Outcome – Negative Binomial Model

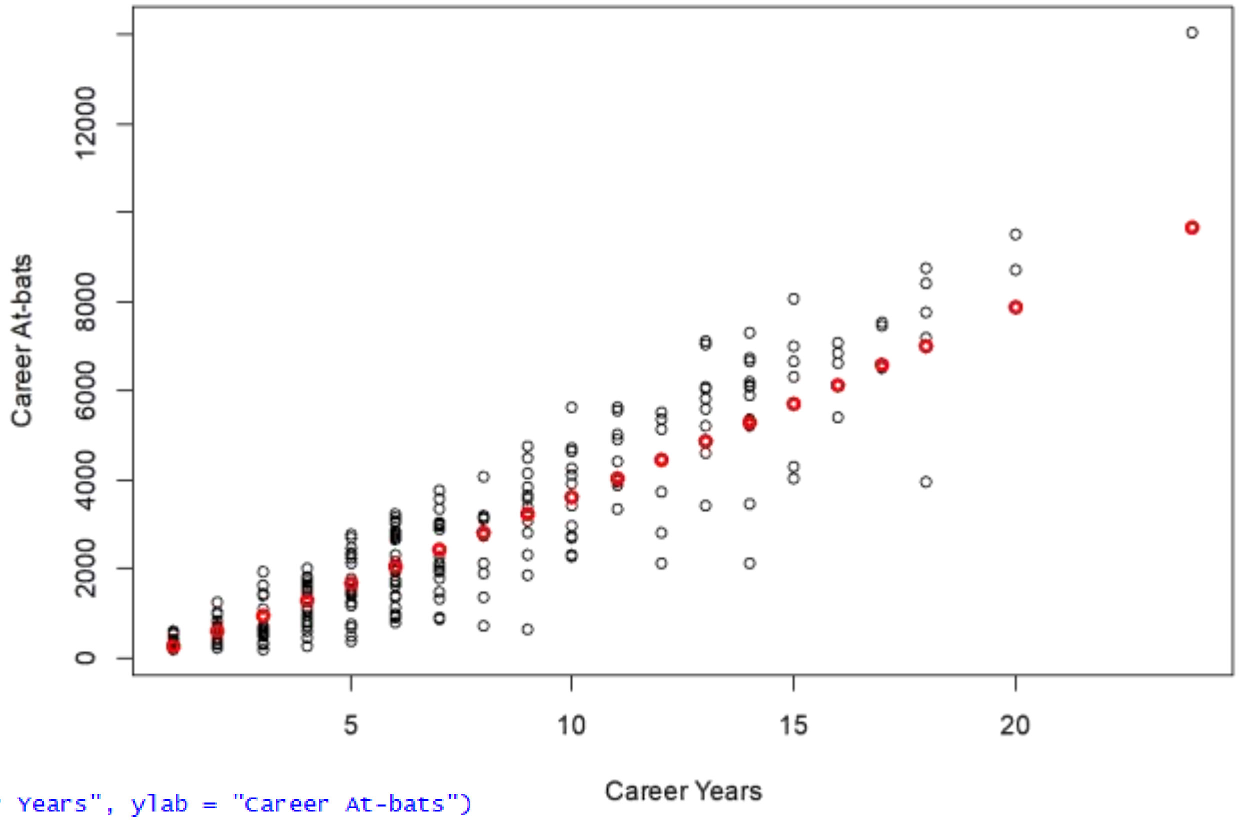
```
> #so we will use negative binomial regression
> # install.packages("MASS")
> library(MASS)
> NegBinomModel = glm.nb(CareerAtBats ~ CareerYears, BaseballData)
> summary(NegBinomModel)

Call:
glm.nb(formula = CareerAtBats ~ CareerYears, data = BaseballData,
    init.theta = 4.456280624, link = log)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.63070  -0.91779  -0.06258   0.58988   1.74574

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) 6.335415   0.054953  115.29   <2e-16 ***
CareerYears 0.173228   0.006429   26.95   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for Negative Binomial(4.4563) family taken to be 1)

    Null deviance: 919.60  on 250  degrees of freedom
Residual deviance: 260.53  on 249  degrees of freedom
AIC: 4107.3

Number of Fisher Scoring iterations: 1


              Theta:  4.456
          Std. Err.:  0.386

 2 x log-likelihood:  -4101.330
> plot(BaseballData$CareerYears, BaseballData$CareerAtBats, xlab = "Career Years", ylab = "Career At-bats")
> points(BaseballData$CareerYears, NegBinomModel$fitted.values, col = "red", lwd = 3)
```





Count Outcome – Negative Binomial Model

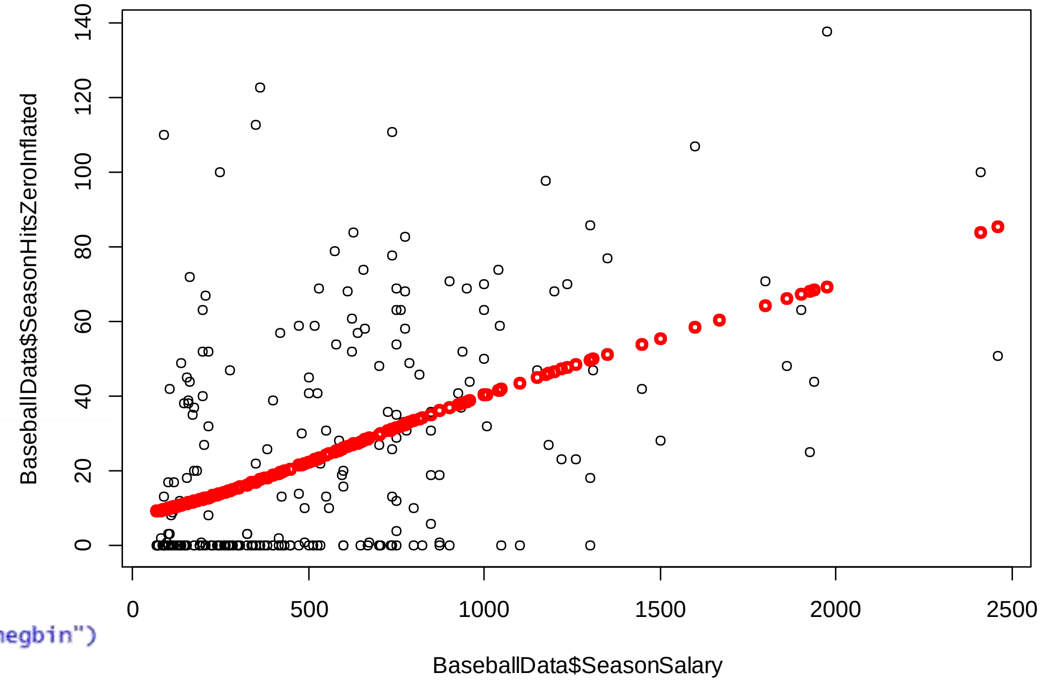
```
> #will want to log predictor if relationship looks linear with raw variables because y is log transformed
> NegBinomModel2 = glm.nb(CareerAtBats ~ log(CareerYears), BaseballData)
> summary(NegBinomModel2)

Call:
glm.nb(formula = CareerAtBats ~ log(CareerYears), data = BaseballData,
    init.theta = 5.847400845, link = log)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.1820  -0.7821   0.0449   0.5562   2.1368

Coefficients:
                 Estimate Std. Error z value Pr(>|z|)
(Intercept)       5.60893    0.06592   85.08   <2e-16 ***
log(CareerYears)  1.12293    0.03499   32.09   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for Negative Binomial(5.8474) family taken to be 1)

    Null deviance: 1205.5  on 250  degrees of freedom
Residual deviance:  258.5  on 249  degrees of freedom
AIC: 4035.2

Number of Fisher Scoring iterations: 1


              Theta:  5.847
          Std. Err.:  0.511

 2 x log-likelihood:  -4029.214
> plot(BaseballData$CareerYears, BaseballData$CareerAtBats, xlab = "Career Years", ylab = "Career At-bats")
> points(BaseballData$CareerYears, NegBinomModel2$fitted.values, col = "red", lwd = 3)
> anova(NegBinomModel, NegBinomModel2)
Likelihood ratio tests of Negative Binomial Models

Response: CareerAtBats
            Model    theta Resid. df    2 x log-lik.   Test    df LR stat. Pr(Chi)
1     CareerYears 4.456281       249       -4101.330
2 log(CareerYears) 5.847401       249       -4029.214 1 vs 2     0 72.11665       0
```



For each 1-unit increase in log(CareerYears), the expected log(CareerAtBats) increases by 1.12

Count Outcome – Zero-Inflated Model

```
> #zero-inflated model
> BaseballData$SeasonHitsZeroInflated= BaseballData$SeasonHits - 100
> BaseballData$SeasonHitsZeroInflated[BaseballData$SeasonHitsZeroInflated < 0] = 0
> sum(BaseballData$SeasonHitsZeroInflated == 0)
[1] 109
> plot(BaseballData$SeasonSalary, BaseballData$SeasonHitsZeroInflated)
> # install.packages("pscl")
> library(pscl)
> ZeroInflatedModel = zeroinfl(SeasonHitsZeroInflated ~ SeasonSalary, BaseballData, dist = "negbin")
> summary(ZeroInflatedModel)

Call:
zeroinfl(formula = SeasonHitsZeroInflated ~ SeasonSalary, data = BaseballData, dist = "negbin")

Pearson residuals:
    Min      1Q  Median      3Q     Max 
-1.2043 -0.6032 -0.4699  0.4174  4.9919 

Count model coefficients (negbin with log link):
              Estimate Std. Error z value Pr(>|z|)    
(Intercept)  3.4352636  0.1062513  32.331  < 2e-16 ***
SeasonSalary 0.0004130  0.0001099   3.758 0.000171 ***
Log(theta)   0.5731038  0.1320986   4.338 1.43e-05 ***

Zero-inflation model coefficients (binomial with logit link):
               Estimate Std. Error z value Pr(>|z|)    
(Intercept)   1.1073460  0.2425845   4.565 5.00e-06 ***
SeasonSalary -0.0029248  0.0004704  -6.217 5.06e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 

Theta = 1.7738 
Number of iterations in BFGS optimization: 13 
Log-likelihood: -805.9 on 5 Df
> plot(BaseballData$SeasonSalary, BaseballData$SeasonHitsZeroInflated)
> points(BaseballData$SeasonSalary, ZeroInflatedModel$fitted.values, col = "red", lwd = 3)
```

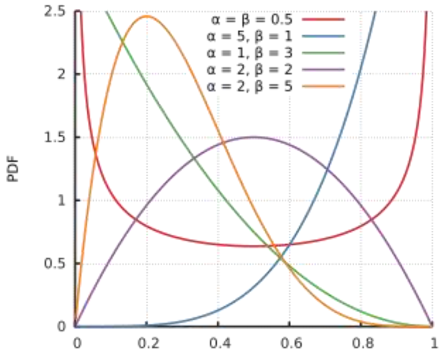


Use when many observations have a 0 on the Y

For each 1-unit increase in SeasonSalary, the expected log(SeasonHits) increases by .0004

Proportional Outcome – Beta Regression

```
> #proportional outcome - beta regresion
> #we probably should have been modeling season batting average as a proportion this whole time...
> #beta distribution is bounded by 0 and 1 just as a proportion is
> #install.packages("betareg")
> library(betareg)
> BetaModel = betareg(SeasonBattingAvg ~ SeasonSalary, data = BaseballData)
> summary(BetaModel)

Call:
betareg(formula = SeasonBattingAvg ~ SeasonSalary, data = BaseballData)

Standardized weighted residuals 2:
    Min      1Q  Median      3Q     Max 
-2.2240 -0.6885 -0.0761  0.6190  2.6723 

Coefficients (mean model with logit link):
               Estimate Std. Error z value Pr(>|z|)    
(Intercept)  -1.087e+00  1.392e-02 -78.055  < 2e-16 ***
SeasonSalary  1.104e-04  1.958e-05   5.638 1.72e-08 ***

Phi coefficients (precision model with identity link):
      Estimate Std. Error z value Pr(>|z|)    
(phi)   258.25      23.02   11.22   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 

Type of estimator: ML (maximum likelihood)
Log-likelihood: 547.5 on 3 Df
Pseudo R-squared: 0.1104
Number of iterations: 15 (BFGS) + 2 (Fisher scoring)
> plot(BaseballData$SeasonSalary, BaseballData$SeasonBattingAvg, xlab = "Season Salary", ylab = "Season Batting Average")
> points(sort(BaseballData$SeasonSalary), sort(BetaModel$fitted.values), lwd = 3, type = "l", col = "red")
```



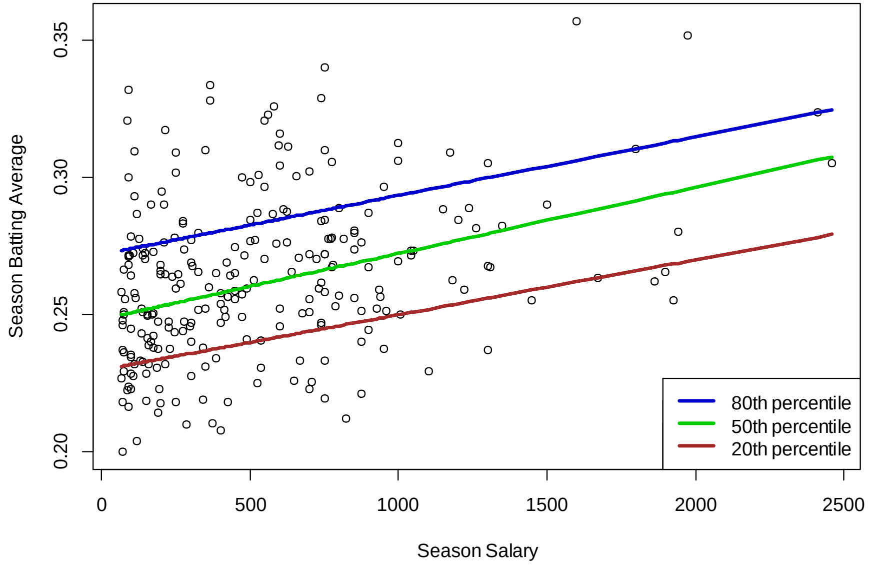
For each 1-unit increase in SeasonSalary, the log odds of SeasonBatting Avg increases by .00011

Log odds = log(probability/1-probability)
Can do algebra to get probability at each X

Quantile (Percentile) Regression

```
> #quantile (percentile) regression
> #more robust to outliers and heterscedasticity
> #install.packages("quantreg")
> library(quantreg)
> #start with predicting median Y at each X (rather than mean in OLS regression)
> QuantileModel.5 = rq(SeasonBattingAvg ~ SeasonSalary, data = BaseballData, tau = 0.5)
> summary(QuantileModel.5)

Call: rq(formula = SeasonBattingAvg ~ SeasonSalary, tau = 0.5, data = BaseballData)

tau: [1] 0.5

Coefficients:
             coefficients lower bd upper bd
(Intercept)  0.24820      0.24403  0.25483
SeasonSalary 0.00002      0.00001  0.00003
> plot(BaseballData$SeasonSalary, BaseballData$SeasonBattingAvg, xlab = "Season Salary", ylab = "Season Batting Average")
> points(sort(BaseballData$SeasonSalary), sort(QuantileModel.5$fitted.values), lwd = 3, type = "l", col = "green3")
> legend("bottomright", legend="50th percentile", col="green3",
+        lty = 1, lwd = 3)
> #can predict 80th percentile of Y at each X
> QuantileModel.8 = rq(SeasonBattingAvg ~ SeasonSalary, data = BaseballData, tau = 0.8)
> summary(QuantileModel.8)

Call: rq(formula = SeasonBattingAvg ~ SeasonSalary, tau = 0.8, data = BaseballData)

tau: [1] 0.8

Coefficients:
             coefficients lower bd upper bd
(Intercept)  0.27185      0.26824  0.27674
SeasonSalary 0.00002      0.00001  0.00004
> points(sort(BaseballData$SeasonSalary), sort(QuantileModel.8$fitted.values), lwd = 3, type = "l", col = "blue3")
> legend("bottomright", legend=c("80th percentile", "50th percentile"), col=c("blue3", "green3"),
+        lty = 1, lwd = 3)
> #can predict 20th percentile of Y at each X
> QuantileModel.2 = rq(SeasonBattingAvg ~ SeasonSalary, data = BaseballData, tau = 0.2)
> summary(QuantileModel.2)

Call: rq(formula = SeasonBattingAvg ~ SeasonSalary, tau = 0.2, data = BaseballData)

tau: [1] 0.2

Coefficients:
             coefficients lower bd upper bd
(Intercept)  0.22981      0.22491  0.23376
SeasonSalary 0.00002      0.00002  0.00003
> points(sort(BaseballData$SeasonSalary), sort(QuantileModel.2$fitted.values), lwd = 3, type = "l", col = "brown")
> legend("bottomright", legend=c("80th percentile", "50th percentile", "20th percentile"), col=c("blue3", "green3", "brown"),
+        lty = 1, lwd = 3)
```

Predict a certain percentile at each X

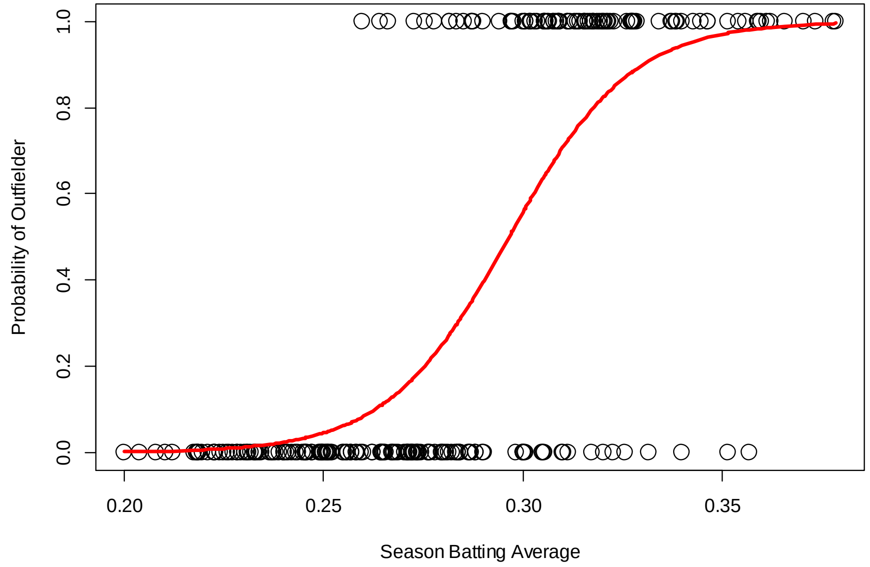


Dichotomous Outcome – Logistic Regression

odds ratio = 2 -> probability of .66

For each 1-unit increase in SeasonBattingAvg, the log odds of being an outfielder increases by 65

```
> #dichotomous outcome - logistic regression
> #outcome will be infielder vs. outfielder
> #removing designated hitters as they don't have a position
> BaseballData = BaseballData[BaseballData$Position != "DH",]
> #must convert position factor to a character vector so it can take different values
> BaseballData$Position = as.character(BaseballData$Position)
> #making new variable for infielder vs. outfielder
> BaseballData$DichotomousPosition = 0
> #if they are an outfielder, they are coded as 1 (otherwise, 0, by defualt)
> BaseballData$DichotomousPosition[BaseballData$Position == "OF"] = 1
> #converting column from character vector to numeric vector
> BaseballData$SeasonBattingAvg[BaseballData$DichotomousPosition == 1] = BaseballData$SeasonBattingAvg[BaseballData$DichotomousPosition == 1
] + .05
> LogisticModel = glm(DichotomousPosition ~ SeasonBattingAvg, data = BaseballData, family = binomial)
> summary(LogisticModel)

Call:
glm(formula = DichotomousPosition ~ SeasonBattingAvg, family = binomial,
    data = BaseballData)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.8144  -0.5251  -0.1656   0.5997   2.2300

Coefficients:
                 Estimate Std. Error z value Pr(>|z|)
(Intercept)       -19.355      2.443  -7.923 2.32e-15 ***
SeasonBattingAvg   65.273      8.293   7.871 3.52e-15 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 319.36  on 237  degrees of freedom
Residual deviance: 172.70  on 236  degrees of freedom
AIC: 176.7

Number of Fisher Scoring iterations: 6

> plot(BaseballData$SeasonBattingAvg, BaseballData$DichotomousPosition, cex = 2, xlab = "Season Batting Average", ylab = "Probability of Out
fielder")
> points(sort(BaseballData$SeasonBattingAvg), sort(LogisticModel$fitted.values), lwd = 3, type = "l", col = "red")
```



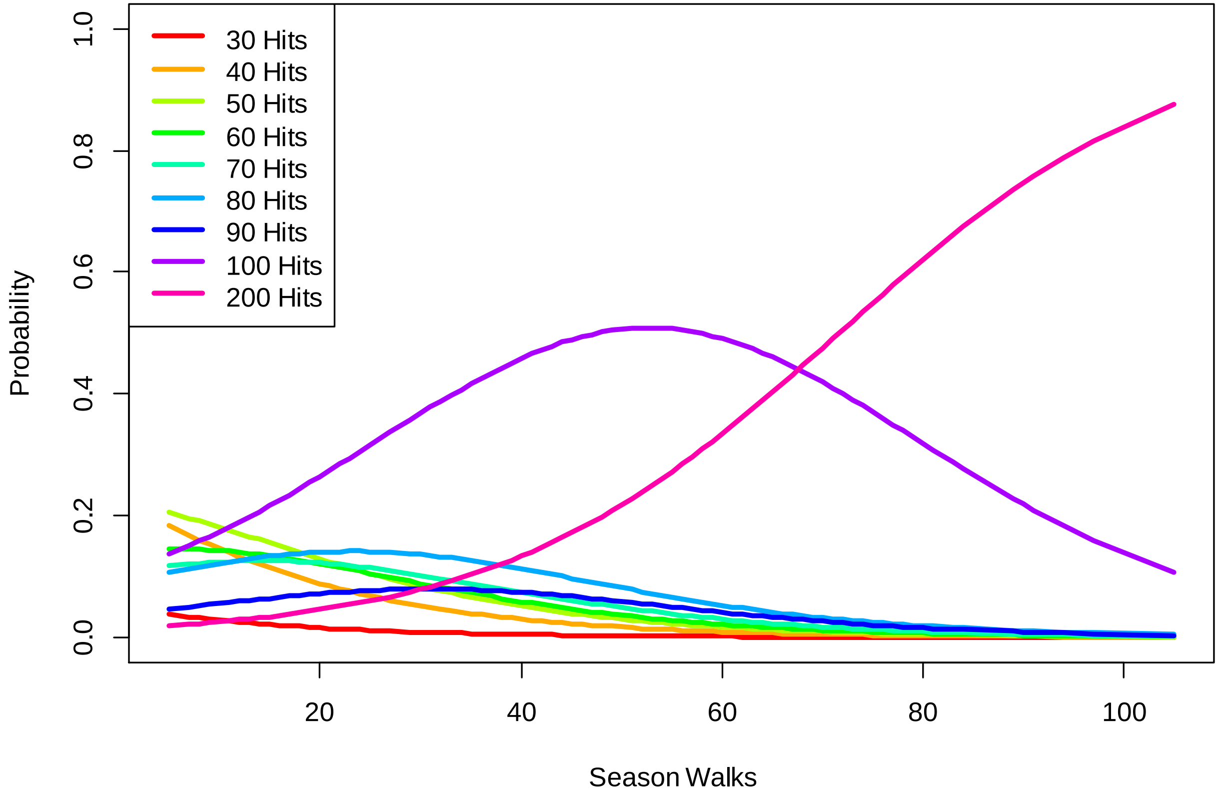
Ordinal Outcome - Ordered Logistic Regression

```
> #ordinal outcome - ordered logistic regression
> #making an orindal variable out of season hits by only allowing for 1 signficant digit
> BaseballData$SeasonHits1SignificantDigit = as.factor(signif(BaseballData$SeasonHits, 1))
> count(BaseballData$SeasonHits1SignificantDigit)
    x freq
1  30    2
2  40   11
3  50   16
4  60   15
5  70   16
6  80   21
7  90   13
8 100   93
9 200   51
> library(MASS)
> OrdinalLogisticModel = polr(SeasonHits1SignificantDigit ~ SeasonWalks, data = BaseballData, Hess = T)
> summary(OrdinalLogisticModel)
Call:
polr(formula = SeasonHits1SignificantDigit ~ SeasonWalks, data = BaseballData,
    Hess = T)

Coefficients:
              Value Std. Error t value
SeasonWalks 0.05893   0.006848   8.605

Intercepts:
        Value   Std. Error t value
30|40   -2.9434  0.7357    -4.0005
40|50   -0.9638  0.3489    -2.7622
50|60   -0.0035  0.2904    -0.0121
60|70    0.5847  0.2808     2.0825
70|80    1.0914  0.2838     3.8463
80|90    1.6614  0.2952     5.6279
90|100   1.9804  0.3036     6.5237
100|200  4.2228  0.3952    10.6847

Residual Deviance: 762.4758
AIC: 780.4758
> #intercepts not very useful
> exp(coef(OrdinalLogisticModel))
SeasonWalks
   1.060697
> #for each extra Season Walk, the odds of Season Hits being in a category vs. the category below it are multiplied by 1.06
> #does not provide p values but does provide t value and could convert that to a p value
> pt(summary(OrdinalLogisticModel)$coefficients[1,3], nrow(BaseballData)-2, lower.tail = F) * 2
[1] 1.085732e-15
```

Ordinal Outcome - Ordered Logistic Regression

```
> #can extract probability of being in each ordinal category based on SeasonWalks
> OrdinalLogisticProbabilities = data.frame(BaseballData$SeasonWalks[!duplicated(BaseballData$SeasonWalks)],
+                                           predict(OrdinalLogisticModel,
+                                                   BaseballData[!duplicated(BaseballData$SeasonWalks),],
+                                                   type = "probs"))
> colnames(OrdinalLogisticProbabilities) = c("SeasonWalks", "P30Hits", "P40Hits", "P50Hits", "P60Hits",
+                                            "P70Hits", "P80Hits", "P90Hits", "P100Hits", "P200Hits")
> OrdinalLogisticProbabilities = OrdinalLogisticProbabilities[order(OrdinalLogisticProbabilities$SeasonWalks),]
> OrdinalLogisticProbabilities
    SeasonWalks      P30Hits      P40Hits     P50Hits     P60Hits     P70Hits     P80Hits     P90Hits  P100Hits   P200Hits
262           5 0.0377605468 0.1834902077 0.204758938 0.146003763 0.117276300 0.107560247 0.046821822 0.1370289 0.01929932
8             7 0.0337041103 0.1679087659 0.195860907 0.145474859 0.120555476 0.113600083 0.050387582 0.1508473 0.02166094
9             8 0.0318368502 0.1604575527 0.191156093 0.144842517 0.121930825 0.116507239 0.052186095 0.1581373 0.02294552
48            9 0.0300698199 0.1532379191 0.186310718 0.143969992 0.123117963 0.119320357 0.053987443 0.1656814 0.02430439
```

Ordinal Outcome – Ordered Logistic Regression




```
> plot(OrdinalLogisticProbabilities$SeasonWalks, OrdinalLogisticProbabilities$P30Hits, xlab = "Season Walks",
+      ylab = "Probability", lwd = 3, type = "l", col = rainbow(9)[1], ylim = c(0, 1))
> points(OrdinalLogisticProbabilities$SeasonWalks, OrdinalLogisticProbabilities$P40Hits, lwd = 3, type = "l", col = rainbow(9)[2])
> points(OrdinalLogisticProbabilities$SeasonWalks, OrdinalLogisticProbabilities$P50Hits, lwd = 3, type = "l", col = rainbow(9)[3])
> points(OrdinalLogisticProbabilities$SeasonWalks, OrdinalLogisticProbabilities$P60Hits, lwd = 3, type = "l", col = rainbow(9)[4])
> points(OrdinalLogisticProbabilities$SeasonWalks, OrdinalLogisticProbabilities$P70Hits, lwd = 3, type = "l", col = rainbow(9)[5])
> points(OrdinalLogisticProbabilities$SeasonWalks, OrdinalLogisticProbabilities$P80Hits, lwd = 3, type = "l", col = rainbow(9)[6])
> points(OrdinalLogisticProbabilities$SeasonWalks, OrdinalLogisticProbabilities$P90Hits, lwd = 3, type = "l", col = rainbow(9)[7])
> points(OrdinalLogisticProbabilities$SeasonWalks, OrdinalLogisticProbabilities$P100Hits, lwd = 3, type = "l", col = rainbow(9)[8])
> points(OrdinalLogisticProbabilities$SeasonWalks, OrdinalLogisticProbabilities$P200Hits, lwd = 3, type = "l", col = rainbow(9)[9])
> legend("topleft", legend=c("30 Hits", "40 Hits", "50 Hits", "60 Hits", "70 Hits",
+                           "80 Hits", "90 Hits", "100 Hits", "200 Hits"),
+        col=rainbow(9), lty = 1, lwd = 3)
```

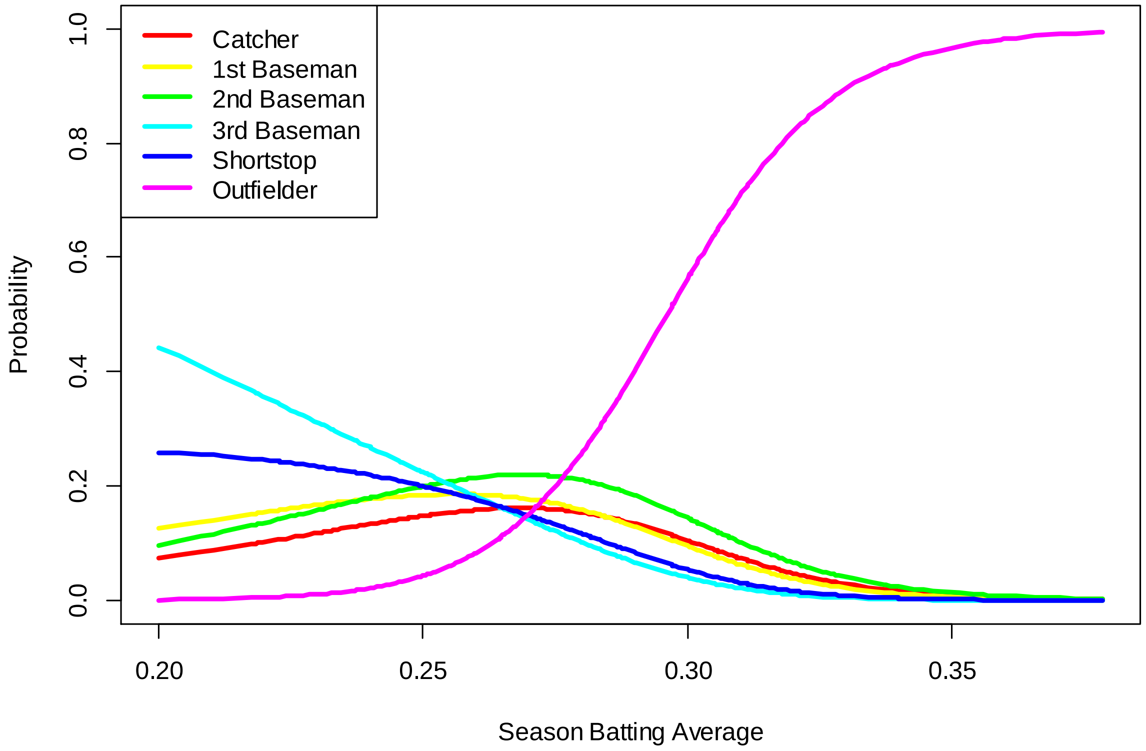
Categorical Outcome – Multinomial Logistic Regression

```
> #categorical outcome - multinomial logistic regression
> #we will predict position from batting average
> #install.packages("nnet")
> library(nnet)
> #need outcome to be a factor
> BaseballData$Position = as.factor(BaseballData$Position)
> levels(BaseballData$Position) = c("C ", "1B", "2B", "3B",  "OF", "SS")
> #makes catchers our baseline group
> MultinomialLogisticModel = multinom(Position ~ SeasonBattingAvg, data = BaseballData)
# weights:  18 (10 variable)
initial  value 426.438754
iter  10 value 316.517898
iter  20 value 310.600403
iter  30 value 310.592334
iter  40 value 310.592019
final  value 310.591908
converged
> summary(MultinomialLogisticModel)
Call:
multinom(formula = Position ~ SeasonBattingAvg, data = BaseballData)

Coefficients:
   (Intercept) SeasonBattingAvg
1B   1.8120254       -6.3658618
2B   0.2003393        0.3972404
3B   7.3283761      -27.6491492
OF -15.7669377       58.1507833
SS   5.1022752      -19.2047561

Std. Errors:
   (Intercept) SeasonBattingAvg
1B    2.568260         9.564138
2B    2.460470         9.072218
3B    2.663592        10.239017
OF    2.922198        10.155475
SS    2.648406        10.062783

Residual Deviance: 621.1838
AIC: 641.1838
> #intercepts not very useful
```

Categorical Outcome – Multinomial Logistic Regression

```
> #intercepts not very useful
> exp(summary(MultinomialLogisticModel)$coefficients)
    (Intercept) SeasonBattingAvg
1B 6.122836e+00     1.719259e-03
2B 1.221817e+00     1.487714e+00
3B 1.522907e+03     9.820352e-13
OF 1.420712e-07     1.797067e+25
SS 1.643955e+02     4.565416e-09
> #as batting average increases by  1 (a lot!), the odds of being a 2nd baseman vs a catcher is multiplied by 1.5
> #no significance testing provided but the coeffecients divided by their standard errors provide t values,
> #which you could convert to p values
> ts = summary(MultinomialLogisticModel)$coefficients/summary(MultinomialLogisticModel)$standard.errors
> ts
   (Intercept) SeasonBattingAvg
1B   0.7055459      -0.66559700
2B   0.0814232       0.04378648
3B   2.7513137      -2.70037157
OF  -5.3955744       5.72605276
SS   1.9265459      -1.90849343
> pt(ts, nrow(BaseballData)-2, lower.tail = F) * 2
   (Intercept) SeasonBattingAvg
1B 0.481166396     1.493681e+00
2B 0.935174412     9.651116e-01
3B 0.006396257     1.992572e+00
OF 1.999999834     3.111102e-08
SS 0.055237183     1.942459e+00
> #can extract probability of being in each position based on SeasonWalks
> MultinomialLogisticProbabilities = data.frame(BaseballData$SeasonBattingAvg[!duplicated(BaseballData$SeasonBattingAvg)],
+                                               predict(MultinomialLogisticModel,
+                                                       BaseballData[!duplicated(BaseballData$SeasonBattingAvg),],
+                                                       type = "probs"))
> colnames(MultinomialLogisticProbabilities) = c("SeasonBattingAvg", "Catcher", "FirstBaseman", "SecondBaseman",
+                                                   "ThirdBaseman", "Outfielder", "Shortstop")
> MultinomialLogisticProbabilities = MultinomialLogisticProbabilities[order(MultinomialLogisticProbabilities$SeasonBattingAvg),]
> MultinomialLogisticProbabilities[100:234,]
    SeasonBattingAvg   Catcher FirstBaseman SecondBaseman ThirdBaseman Outfielder    Shortstop
244        0.2734375 0.160422738  0.172288871   0.218496987 1.272145e-01  0.1833652 0.1382117823
149        0.2736419 0.160306098  0.171939785   0.218355846 1.264057e-01  0.1854222 0.1375703333
155        0.2737430 0.160246485  0.171765193   0.218283418 1.260058e-01  0.1864469 0.1372522539
266        0.2740385 0.160065276  0.171248578   0.218062172 1.248393e-01  0.1894633 0.1363213691
```

Categorical Outcome – Multinomial Logistic Regression




```
> plot(MultinomialLogisticProbabilities$SeasonBattingAvg, MultinomialLogisticProbabilities$Catcher, xlab = "Season Batting Average",
+      ylab = "Probability", lwd = 3, type = "l", col = rainbow(6)[1], ylim = c(0, 1))
> points(MultinomialLogisticProbabilities$SeasonBattingAvg,
+        MultinomialLogisticProbabilities$FirstBaseman, lwd = 3, type = "l", col = rainbow(6)[2])
> points(MultinomialLogisticProbabilities$SeasonBattingAvg,
+        MultinomialLogisticProbabilities$SecondBaseman, lwd = 3, type = "l", col = rainbow(6)[3])
> points(MultinomialLogisticProbabilities$SeasonBattingAvg,
+        MultinomialLogisticProbabilities$ThirdBaseman, lwd = 3, type = "l", col = rainbow(6)[4])
> points(MultinomialLogisticProbabilities$SeasonBattingAvg,
+        MultinomialLogisticProbabilities$Shortstop, lwd = 3, type = "l", col = rainbow(6)[5])
> points(MultinomialLogisticProbabilities$SeasonBattingAvg,
+        MultinomialLogisticProbabilities$Outfielder, lwd = 3, type = "l", col = rainbow(6)[6])
> legend("topleft", legend=c("Catcher", "1st Baseman", "2nd Baseman",
+                            "3rd Baseman", "Shortstop", "Outfielder"),
+        col=rainbow(6), lty = 1, lwd = 3)
```

Nested Data – Multilevel Modeling

- Assumption of regression: independence of residuals/errors
  - When violated, we can use multilevel modeling
- Dependence of residuals/errors usually results from grouping/nesting in the measured DV
  - Students nested in classrooms/teachers
  - Observations nested within participants (i.e., repeated measures)
  - Participants nested in countries
- Conceptually, it’s like running separate regression models for each classroom and then aggregating them
- Can have student-level and classroom-level predictors
- Can have more than 2 levels

Multilevel Equations

- Example with level-1 and level-2 predictors:
  - Level-1 Model:
    - $Y_{ij} = \beta_{0j} + \beta_{1j}X_{ij} + r_{ij}$
  - Level-2 Model:
    - $\beta_{0j} = \gamma_{00} + \gamma_{01}W_j + u_{0j}$
    - $\beta_{1j} = \gamma_{10} + \gamma_{11}W_j + u_{1j}$
  - Combined Model:
    - $Y_{ij} = \gamma_{00} + \gamma_{01}W_j + \gamma_{10}X_{ij} + \gamma_{11}W_jX_{ij} + u_{0j} + u_{1j}X_{ij} + r_{ij}$
- $\mathrm{Var}(r_{ij}) = \sigma^2$
- $\mathrm{Var}(u_{0j}) = \tau_{00}$
- $\mathrm{Var}(u_{1j}) = \tau_{11}$
- $\mathrm{Cov}(u_{0j}, u_{1j}) = \tau_{01}$

Nested Data – Multilevel Modeling

```
> #multilevel modeling
> #install.packages("lme4")
> library(lme4)
> #treating players as nested within teams
> #predcting batting average from home runs within teams
> MultilevelModel = lmer(SeasonSalary ~ SeasonHomeRuns + (SeasonHomeRuns | Team),
+                        data = BaseballData, REML = F)
> summary(MultilevelModel)
Linear mixed model fit by maximum likelihood
t-tests use  Satterthwaite approximations to degrees of freedom ['lmerMod']
Formula: SeasonSalary ~ SeasonHomeRuns + (SeasonHomeRuns | Team)
   Data: BaseballData

     AIC      BIC   logLik deviance df.resid
  3550.0   3570.9  -1769.0   3538.0      232

Scaled residuals:
    Min      1Q  Median      3Q     Max
-2.6855 -0.6923 -0.1894  0.5151  4.1844

Random effects:
 Groups   Name           Variance Std.Dev. Corr
 Team     (Intercept)      1560.0  39.50
          SeasonHomeRuns    168.7  12.99   -0.71
 Residual                150331.9 387.73
Number of obs: 238, groups:  Team, 26

Fixed effects:
               Estimate Std. Error      df t value Pr(>|t|)
(Intercept)     313.932     43.897  30.537   7.152 5.32e-08 ***
SeasonHomeRuns   19.349      4.023  21.473   4.809 8.90e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Correlation of Fixed Effects:
            (Intr)
SeasonHmRns -0.693
> #no p values - what are the degrees of freedom?
```

- Level-1 Model:
  - $Salary_{ij} = \beta_{0j} + \beta_{1j}SeasonHomeRuns_{ij} + r_{ij}$
- Level-2 Model:
  - $\beta_{0j} = 313.9 + u_{0j}$
  - $\beta_{1j} = 19.3 + u_{1j}$
- $Var(r_{ij}) = \sigma^2 = 150331.9$
- $Var(u_{0j}) = \tau_{00} = 1560.0$
- $Var(u_{1j}) = \tau_{11} = 168.7$
- $Cor(u_{0j}, u_{1j}) = -.71$

Within teams, a 1-unit increase in SeasonHomeRuns the expected Salary increases by 19.3 units

Nested Data – Multilevel Modeling

```
> #can get rough estimates of degrees of freedom and p-values from the lmerTest package
> #uses the the Satterthwaite approximation
> #install.packages("lmerTest")
> library(lmerTest)
> MultilevelModel = lmer(SeasonSalary ~ SeasonHomeRuns + (SeasonHomeRuns | Team),
+                        data = BaseballData, REML = F)
> summary(MultilevelModel)
Linear mixed model fit by maximum likelihood
t-tests use  Satterthwaite approximations to degrees of freedom ['lmerMod']
Formula: SeasonSalary ~ SeasonHomeRuns + (SeasonHomeRuns | Team)
   Data: BaseballData

     AIC      BIC   logLik deviance df.resid
  3550.0   3570.9  -1769.0   3538.0      232

Scaled residuals:
    Min      1Q  Median      3Q     Max
-2.6855 -0.6923 -0.1894  0.5151  4.1844

Random effects:
 Groups   Name           Variance Std.Dev. Corr
 Team     (Intercept)      1560.0  39.50
          SeasonHomeRuns    168.7  12.99   -0.71
 Residual                150331.9 387.73
Number of obs: 238, groups:  Team, 26

Fixed effects:
               Estimate Std. Error      df t value Pr(>|t|)
(Intercept)     313.932     43.897  30.537   7.152 5.32e-08 ***
SeasonHomeRuns   19.349      4.023  21.473   4.809 8.90e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Correlation of Fixed Effects:
            (Intr)
SeasonHmRns -0.693
```

Nested Data – Multilevel Modeling

```
> #but can do model comparison (using liklihood ratios for more accurate results)
> MultilevelBaseModel = lmer(SeasonSalary ~ 1 + (SeasonHomeRuns | Team),
+                        data = BaseballData, REML = F)
> summary(MultilevelBaseModel)
summary from lme4 is returned
some computational error has occurred in lmerTest
Linear mixed model fit by maximum likelihood  ['lmerMod']
Formula: SeasonSalary ~ 1 + (SeasonHomeRuns | Team)
   Data: BaseballData

     AIC      BIC   logLik deviance df.resid
  3564.5   3581.9  -1777.2   3554.5      233

Scaled residuals:
    Min      1Q  Median      3Q     Max
-2.7005 -0.7289 -0.2136  0.5258  3.9716

Random effects:
 Groups   Name           Variance Std.Dev. Corr
 Team     (Intercept)     19699.7 140.4
          SeasonHomeRuns    561.7  23.7    -0.97
 Residual                152038.5 389.9
Number of obs: 238, groups:  Team, 26

Fixed effects:
            Estimate Std. Error t value
(Intercept)   451.85      30.26   14.93
> anova(MultilevelBaseModel, MultilevelModel)
Data: BaseballData
Models:
object: SeasonSalary ~ 1 + (SeasonHomeRuns | Team)
..1: SeasonSalary ~ SeasonHomeRuns + (SeasonHomeRuns | Team)
       Df    AIC    BIC  logLik deviance  Chisq Chi Df Pr(>Chisq)
object  5 3564.5 3581.9 -1777.2   3554.5
..1     6 3550.0 3570.9 -1769.0   3538.0 16.462      1  4.964e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Nested Data – Multilevel Modeling

```
> #because proportion of reduction in error variance is a pseudo-Rsquared
> PseudoRSquared = (1.497e-03 - 1.459e-03) / 1.497e-03
> PseudoRSquared
[1] 0.0253841
> PseudoR = sqrt(PseudoRSquared)
> PseudoR
[1] 0.1593239
```

Regularization

- To prevent overfitting, take parsimony into account
- Ridge (L_2)
  - Causes regression coefficients to shrink
- Lasso (L_1)
  - Causes some regression coefficients to become 0
- Elastic Net
  - Hybrid of other 2

- L_2 regularization (a.k.a. ridge regression). Find β which minimizes:

$$\sum_{j=1}^{N}(y_j - \sum_{i=0}^{d}\beta_i \cdot x_i)^2 + \lambda\sum_{i=1}^{d}\beta_i^2$$

  - λ is the regularization parameter: bigger λ imposes more constraint

- L_1 regularization (a.k.a. lasso). Find β which minimizes:

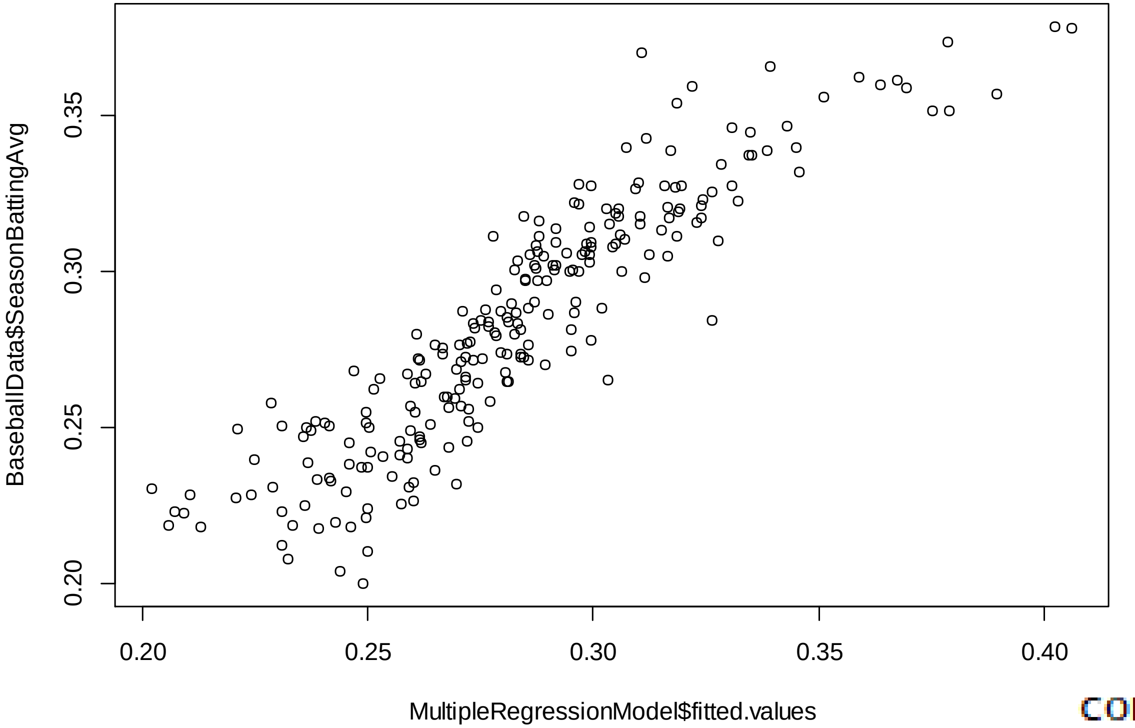
$$\sum_{j=1}^{N}(y_j - \sum_{i=0}^{d}\beta_i \cdot x_i)^2 + \lambda\sum_{i=1}^{d}|\beta_i|$$

Penalty = $(1-\alpha)\,|\boldsymbol{\beta}|_1 + \alpha\,|\boldsymbol{\beta}|^2$

= $\lambda_2\,|\boldsymbol{\beta}|^2 + \lambda_1\,|\boldsymbol{\beta}|_1$ where $\alpha = \frac{\lambda_2}{\lambda_1+\lambda_2}$

Overfit Multiple Regression Model

```
> #multiple regression model
> #output is Batting Average but just going to use linear regression
> MultipleRegressionModel = lm(SeasonBattingAvg ~ SeasonAtBats + SeasonHits + SeasonHomeRuns + SeasonRuns +
+                               SeasonRBIs + SeasonWalks + SeasonPutouts + SeasonAssists + SeasonErrors +
+                               SeasonSalary, data = BaseballData)
> summary(MultipleRegressionModel)

Call:
lm(formula = SeasonBattingAvg ~ SeasonAtBats + SeasonHits + SeasonHomeRuns +
    SeasonRuns + SeasonRBIs + SeasonWalks + SeasonPutouts + SeasonAssists +
    SeasonErrors + SeasonSalary, data = BaseballData)

Residuals:
      Min        1Q    Median        3Q       Max
-0.048949 -0.011248  0.002318  0.010838  0.059542

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)     2.773e-01  3.832e-03  72.375   <2e-16 ***
SeasonAtBats   -4.474e-04  3.267e-05 -13.697   <2e-16 ***
SeasonHits      1.932e-03  1.156e-04  16.711   <2e-16 ***
SeasonHomeRuns -2.335e-04  2.987e-04  -0.782   0.4351
SeasonRuns      2.618e-04  1.382e-04   1.895   0.0593 .
SeasonRBIs      1.340e-05  1.284e-04   0.104   0.9170
SeasonWalks     2.243e-05  8.154e-05   0.275   0.7835
SeasonPutouts  -4.005e-05  4.441e-06  -9.019   <2e-16 ***
SeasonAssists  -1.242e-04  1.190e-05 -10.435   <2e-16 ***
SeasonErrors   -6.122e-04  2.435e-04  -2.514   0.0126 *
SeasonSalary   -5.167e-06  3.126e-06  -1.653   0.0998 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.01711 on 227 degrees of freedom
Multiple R-squared:  0.8243,    Adjusted R-squared:  0.8166
F-statistic: 106.5 on 10 and 227 DF,  p-value: < 2.2e-16

> lm.beta(MultipleRegressionModel)
  SeasonAtBats     SeasonHits SeasonHomeRuns     SeasonRuns     SeasonRBIs    SeasonWalks  SeasonPutouts  SeasonAssists
  -1.608113299    2.161510790   -0.051156533    0.166163097    0.008660163    0.011914773   -0.283032481   -0.462840432
  SeasonErrors   SeasonSalary
  -0.100641931   -0.058839852
```



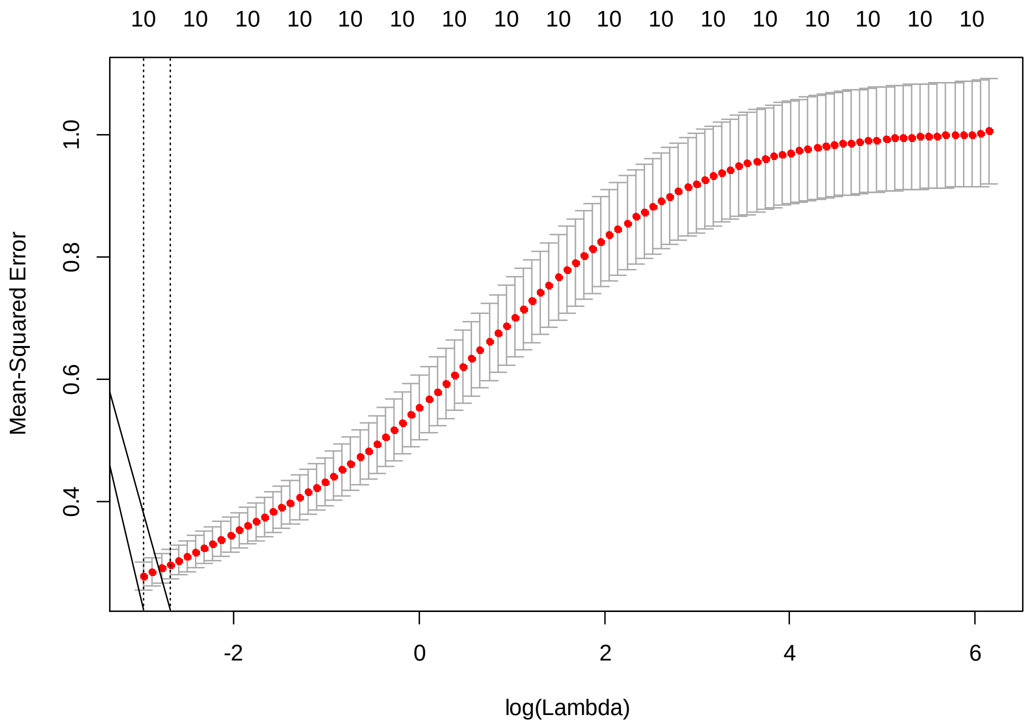
cor
0.9079237

Multicollinearity

```
> #install.packages("car")
> library(car)
> vif(MultipleRegressionModel)
  SeasonAtBats     SeasonHits SeasonHomeRuns     SeasonRuns     SeasonRBIs    SeasonWalks  SeasonPutouts  SeasonAssists 
     17.810640      21.617820       5.531555       9.931990       8.901544       2.424003       1.272567       2.542178 
  SeasonErrors   SeasonSalary 
      2.070586       1.637749 
> sqrt(vif(MultipleRegressionModel))
  SeasonAtBats     SeasonHits SeasonHomeRuns     SeasonRuns     SeasonRBIs    SeasonWalks  SeasonPutouts  SeasonAssists 
      4.220265       4.649497       2.351926       3.151506       2.983546       1.556921       1.128081       1.594421 
  SeasonErrors   SeasonSalary 
      1.438953       1.279746 
> #The square root of the variance inflation factor indicates how much larger the standard error is,
> #compared with what it would be if that variable were uncorrelated with the other predictor variables in the model.
```

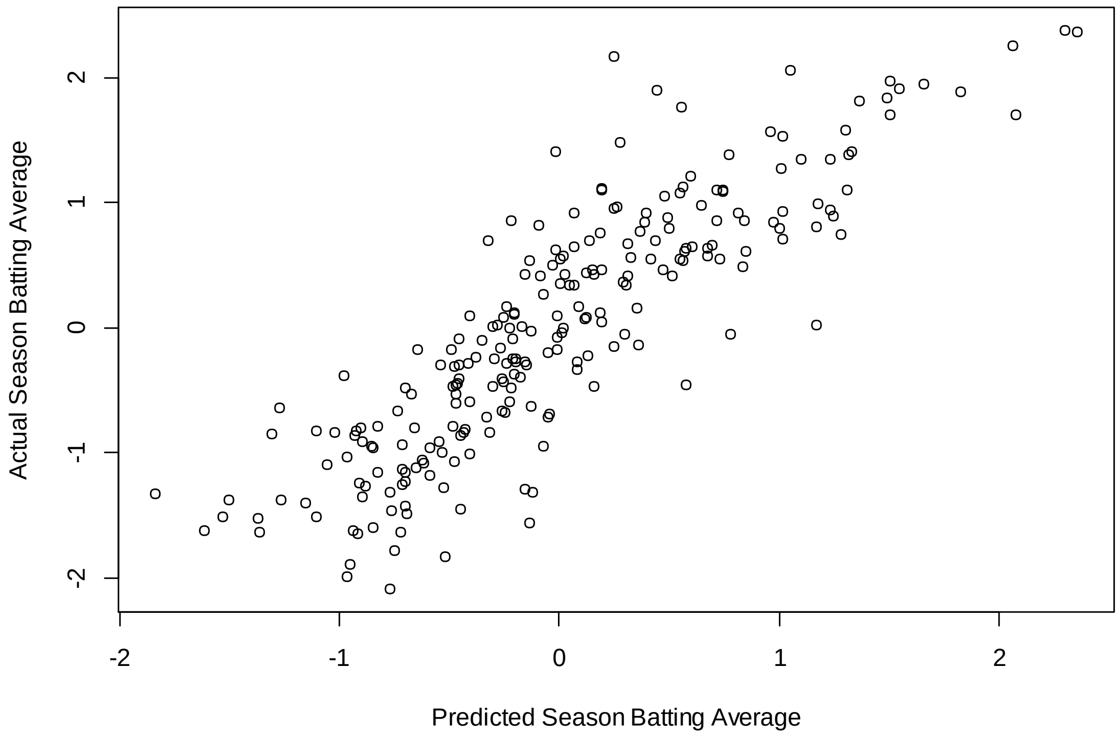
Ridge Regression

```
> #ridge regression
> #going to standardize variables
> BaseballData[,grep("^Season", colnames(BaseballData))[1:10]] = scale(BaseballData[,grep("^Season", colnames(BaseballData))[1:10]])
> BaseballData$SeasonBattingAvg = scale(BaseballData$SeasonBattingAvg)
> #install.packages("glmnet")
> library(glmnet)
> #use 10-fold cross validation to choose the best lambda (how much of a penalty for coeffecients)
> RidgeCV = cv.glmnet(as.matrix(BaseballData[,grep("^Season", colnames(BaseballData))[1:10]]),
+                     BaseballData$SeasonBattingAvg, alpha = 0)
> plot(RidgeCV)
> RidgeModel = glmnet(as.matrix(scale(BaseballData[,grep("^Season", colnames(BaseballData))[1:10]])),
+                     BaseballData$SeasonBattingAvg, alpha = 0, lambda = RidgeCV$lambda.min)
> coef(RidgeModel)
11 x 1 sparse Matrix of class "dgCMatrix"
                          s0
(Intercept)    -5.116049e-17
SeasonAtBats   -5.058454e-01
SeasonHits      9.359258e-01
SeasonHomeRuns -1.310218e-01
SeasonRuns      3.212652e-01
SeasonRBIs      6.499673e-02
SeasonWalks    -9.224874e-02
SeasonPutouts  -2.806091e-01
SeasonAssists  -4.449597e-01
SeasonErrors   -1.607571e-01
SeasonSalary    2.856858e-02
```



Ridge Regression

```
> plot(predict(RidgeModel, newx = as.matrix(BaseballData[,grep("^Season", colnames(BaseballData))[1:10]])),
+      BaseballData$SeasonBattingAvg, xlab = "Predicted Season Batting Average",
+      ylab = "Actual Season Batting Average")
> cor.test(predict(RidgeModel, newx = as.matrix(BaseballData[,grep("^Season", colnames(BaseballData))[1:10]])),
+          BaseballData$SeasonBattingAvg)

        Pearson's product-moment correlation

data:  predict(RidgeModel, newx = as.matrix(BaseballData[, grep("^Season",  and BaseballData$SeasonBattingAvg    colnames(BaseballData))[1:1
0]])) and BaseballData$SeasonBattingAvg
t = 26.727, df = 236, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.8314997 0.8954324
sample estimates:
      cor
0.8669903

> #no standard errors so no confidence intervals and p values :(
```



Lasso Regression

```
> #lasso regression
> #use 10-fold cross validation to choose the best lambda (how much of a penalty for coeffecients)
> LassoCV = cv.glmnet(as.matrix(BaseballData[,grep("^Season", colnames(BaseballData))[1:10]]),
+                     BaseballData$SeasonBattingAvg, alpha = 1)
> plot(LassoCV)
> #run lasso regression with best lambda penalty
> LassoModel = glmnet(as.matrix(BaseballData[,grep("^Season", colnames(BaseballData))[1:10]]),
+                     BaseballData$SeasonBattingAvg, alpha = 1, lambda = LassoCV$lambda.min)
> coef(LassoModel)
11 x 1 sparse Matrix of class "dgCMatrix"
                           s0
(Intercept)     -7.118814e-17
SeasonAtBats    -1.478604e+00
SeasonHits       2.045501e+00
SeasonHomeRuns  -4.251243e-02
SeasonRuns       1.540756e-01
SeasonRBIs       .
SeasonWalks      .
SeasonPutouts   -2.823383e-01
SeasonAssists   -4.614148e-01
SeasonErrors    -1.039151e-01
SeasonSalary    -4.336433e-02
> plot(predict(LassoModel, newx = as.matrix(BaseballData[,grep("^Season", colnames(BaseballData))[1:10]])),
+      BaseballData$SeasonBattingAvg, xlab = "Predicted Season Batting Average",
+      ylab = "Actual Season Batting Average")
> cor.test(predict(LassoModel, newx = as.matrix(BaseballData[,grep("^Season", colnames(BaseballData))[1:10]])),
+          BaseballData$SeasonBattingAvg)

        Pearson's product-moment correlation

data:  predict(LassoModel, newx = as.matrix(BaseballData[, grep("^Season",  and BaseballData$SeasonBattingAvg    colnames(BaseballData))[1:1
0]])) and BaseballData$SeasonBattingAvg
t = 33.196, df = 236, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.8821736 0.9276392
sample estimates:
      cor
0.9075298

> #no standard errors so no confidence intervals and p values :(
```

Elastic Net Regression

```
> #elastic net regression
> #use 10-fold cross validation to choose the best lambda (how much of a penalty for coeffecients)
> ElasticNetCV = cv.glmnet(as.matrix(BaseballData[,grep("^Season", colnames(BaseballData))[1:10]]),
+                          BaseballData$SeasonBattingAvg, alpha = .5)
> plot(ElasticNetCV)
> #run elastic net regression with best lambda penalty
> ElasticNetModel = glmnet(as.matrix(BaseballData[,grep("^Season", colnames(BaseballData))[1:10]]),
+                          BaseballData$SeasonBattingAvg, alpha = .5, lambda = ElasticNetCV$lambda.min)
> coef(ElasticNetModel)
11 x 1 sparse Matrix of class "dgCMatrix"
                          s0
(Intercept)    -7.215433e-17
SeasonAtBats   -1.522572e+00
SeasonHits      2.078355e+00
SeasonHomeRuns -4.919470e-02
SeasonRuns      1.724985e-01
SeasonRBIs      2.645930e-04
SeasonWalks     2.456380e-03
SeasonPutouts  -2.831747e-01
SeasonAssists  -4.634619e-01
SeasonErrors   -1.035497e-01
SeasonSalary   -5.010126e-02
> plot(predict(ElasticNetModel, newx = as.matrix(BaseballData[,grep("^Season", colnames(BaseballData))[1:10]])),
+      BaseballData$SeasonBattingAvg, xlab = "Predicted Season Batting Average",
+      ylab = "Actual Season Batting Average")
> cor.test(predict(ElasticNetModel, newx = as.matrix(BaseballData[,grep("^Season", colnames(BaseballData))[1:10]])),
+          BaseballData$SeasonBattingAvg)

        Pearson's product-moment correlation

data:  predict(ElasticNetModel, newx = as.matrix(BaseballData[, grep("^Season",  and BaseballData$SeasonBattingAvg    colnames(BaseballData)
)[1:10]])) and BaseballData$SeasonBattingAvg
t = 33.241, df = 236, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.8824501 0.9278131
sample estimates:
      cor
0.9077497

> #no standard errors so no confidence intervals and p values :(
```

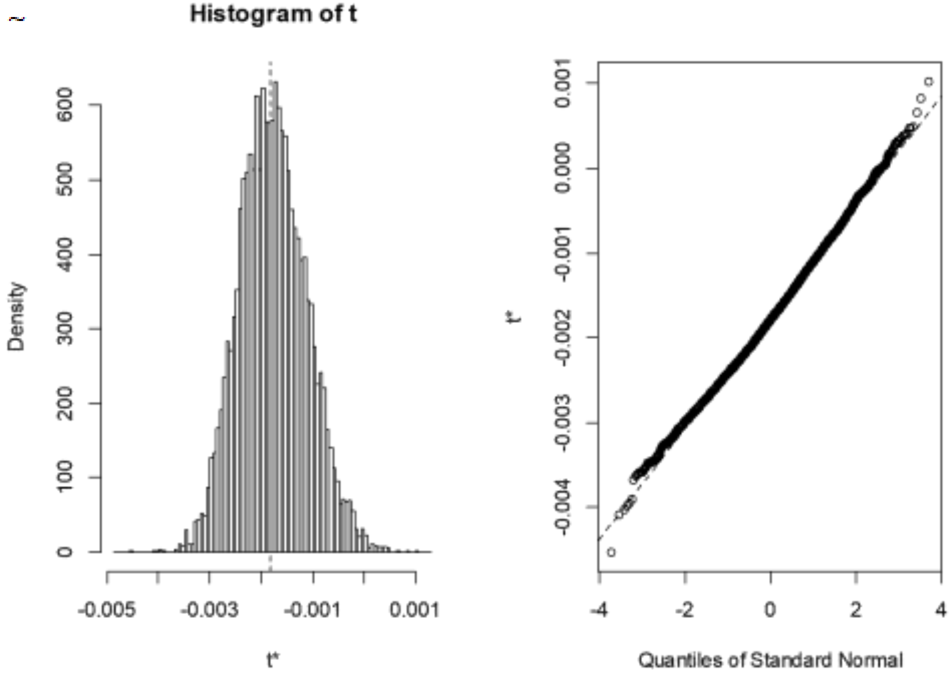
Bootstrapping

- Two assumptions of regression:
  1. Homoscedasticity
     - Variance of residuals does not change at levels of X
  2. Residuals/errors normally distributed
     - Can use histogram or P-P / Q-Q plot

- Can solve each with bootstrapping
  - Imagine a dataset with N rows
  - Could sample rows with replacement N times
  - Run model with that new dataset
  - Record results
  - Repeat 10,000 times
  - Provides distribution of results with a mean and standard deviation/error

Bootstrapping Three-Predictor Model

```
> #bootstrapping
> #install.packages("boot")
> library(boot)
> # function to obtain regression coeffecients
> bs <- function(formula, data, indices) {
+   d <- data[indices,] # allows boot to select sample
+   fit <- lm(formula, data=d)
+   return(coef(fit))
+ }
> BootResults = boot(data = BaseballData, statistic = bs, R = 10000,
+                 formula = SeasonBattingAvg ~ CareerYears + CareerHitsPerYear + SeasonSalary)
> BootResults

ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = BaseballData, statistic = bs, R = 10000, formula = SeasonBattingAvg ~
    CareerYears + CareerHitsPerYear + SeasonSalary)


Bootstrap Statistics :
         original        bias     std. error
t1*  2.584571e-01 -2.428137e-04 6.885982e-03
t2* -1.807435e-03  3.845195e-05 6.535281e-04
t3*  3.292326e-04  1.368640e-06 7.924091e-05
t4*  1.383175e-05 -2.519730e-07 7.020067e-06
> plot(BootResults, index=2)
> boot.ci(BootResults, type="basic", index=2)
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 10000 bootstrap replicates

CALL :
boot.ci(boot.out = BootResults, type = "basic", index = 2)

Intervals :
Level      Basic
95%   (-0.0032, -0.0006 )
Calculations and Intervals on Original Scale
```



Causal Claims

- Correctly specified model
  - Regression coefficients can be interpreted as causal effects if model is "correctly specified"
    - Other models still valid prediction models
  - All causes of Y that are correlated with any Xs in the model are in the model
    - Rare, except for…
- Random assignment
  - Creates a correctly specified model
  - If randomly assigned condition is a predictor, nothing is correlated to it (assuming large enough N)
    - So model is correctly specified
  - If add covariates, can still interpret effects of condition causally
  - Must be able to manipulate IV
    - If cannot, try to make model as "correct" as possible

Future Directions

- Structural Equation Modeling (SEM)
  - Path analysis
    - Mediation
  - Latent variables – model measurement error
- Factor Analysis
- Longitudinal data analysis
  - Regressed change
    - Predict t_2 from t_1 and other variables
  - Difference scores
    - Outcome is t_2-t_1
  - MLM
  - SEM
    - Latent growth models
    - Cross-lagged models
  - Time series analyses
- Machine Learning