

Model_Building_Examples

James D. Wilson

April 29, 2017

Case Study I: Why are low-quality diamonds more expensive? Linear Models

Here, we demonstrate an example of model building based on the `diamonds` data set in the `ggplot2` package. This code is borrowed from Chapter 24.2 in the *R for Data Science* book.

Load all needed packages

```
install.packages(c("ggplot2", "tidyverse", "modelr", "hexbin"), repos = "https://cloud.r-project.org")

## Installing packages into '/Users/jdwilson4/Library/R/3.3/library'
## (as 'lib' is unspecified)

##
## The downloaded binary packages are in
## /var/folders/hm/8gnvskgx0rb1c11fzmz7sgf82j1yqg/T//RtmpXUU4z3/downloaded_packages
library(ggplot2, quiet = TRUE)
library(tidyverse, quiet = TRUE)

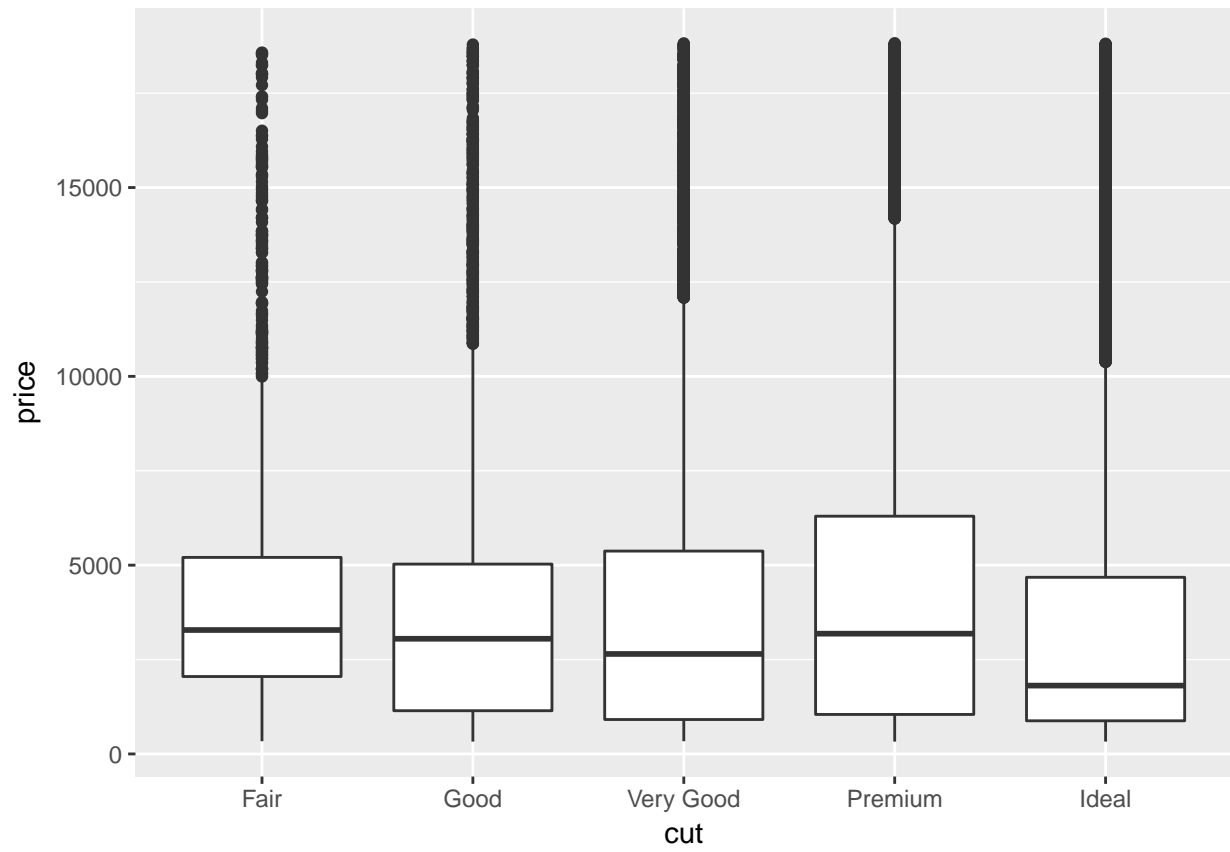
## Loading tidyverse: tibble
## Loading tidyverse: tidyr
## Loading tidyverse: readr
## Loading tidyverse: purrr
## Loading tidyverse: dplyr

## Conflicts with tidy packages -----
## filter(): dplyr, stats
## lag():    dplyr, stats
library(modelr, quiet = TRUE)
library(hexbin, quiet = TRUE)
```

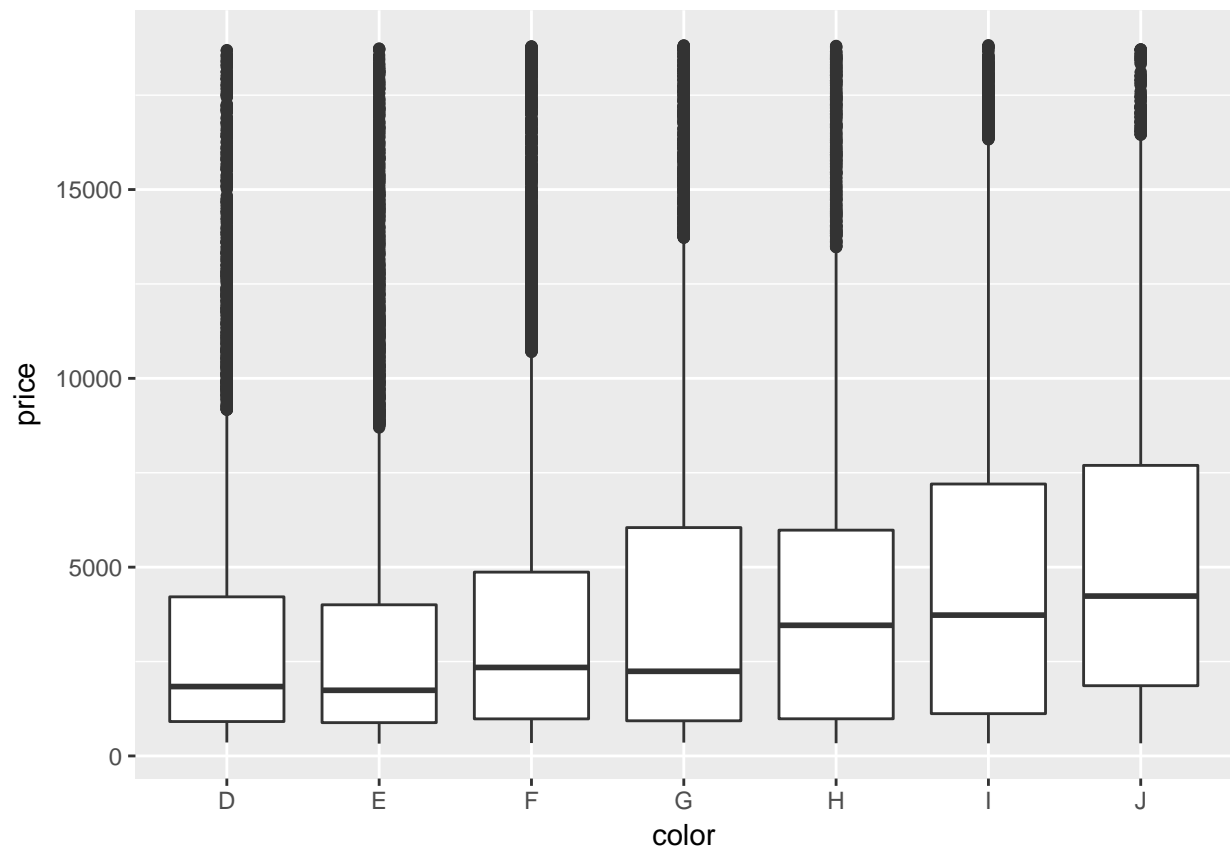
Exploratory Analysis of diamond quality and cost

We want to investigate the relationship between diamond quality and cost so that we can a model that quantifies their relationship. To begin, we look at boxplots of the cost of diamonds of varying qualities. In particular, we investigate the relationship between `cost` and `cut`, `color` and `clarity` of the diamond.

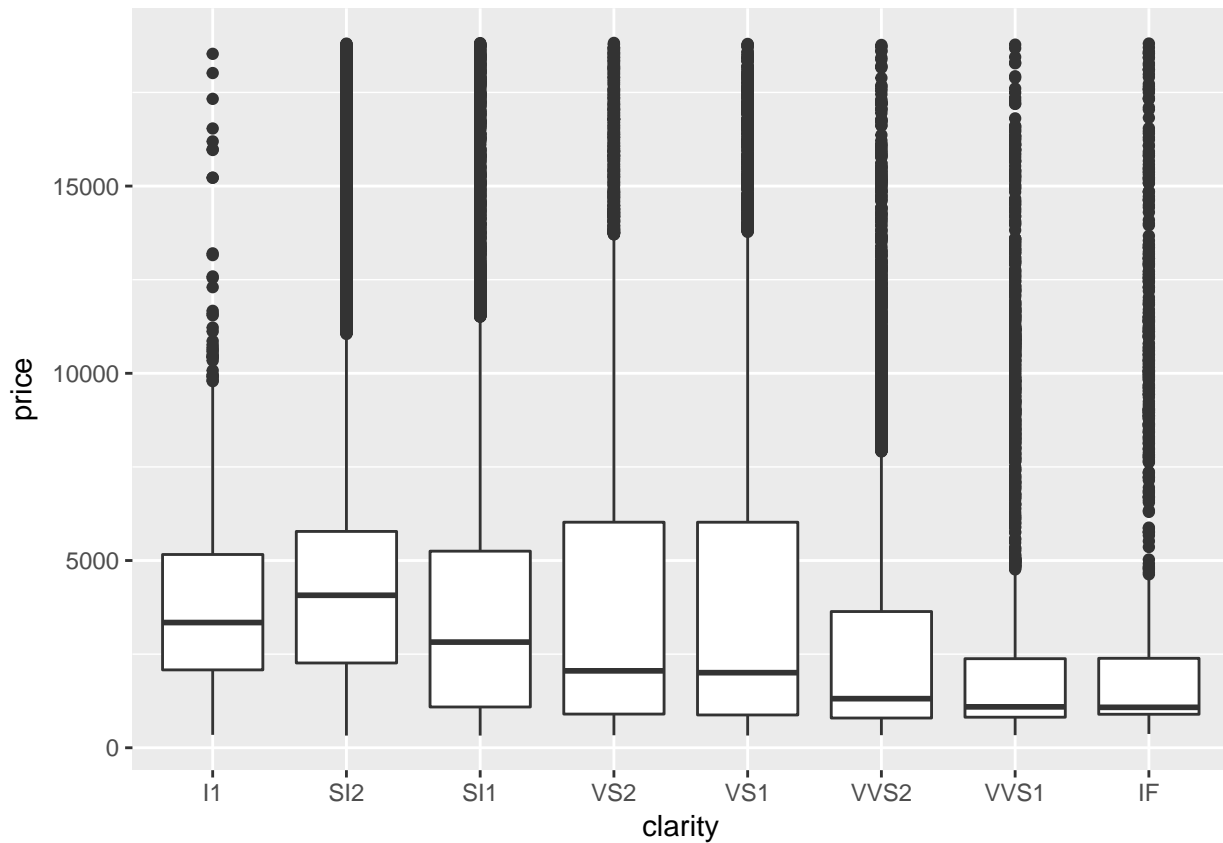
```
ggplot(diamonds, aes(cut, price)) + geom_boxplot()
```



```
ggplot(diamonds, aes(color, price)) + geom_boxplot()
```

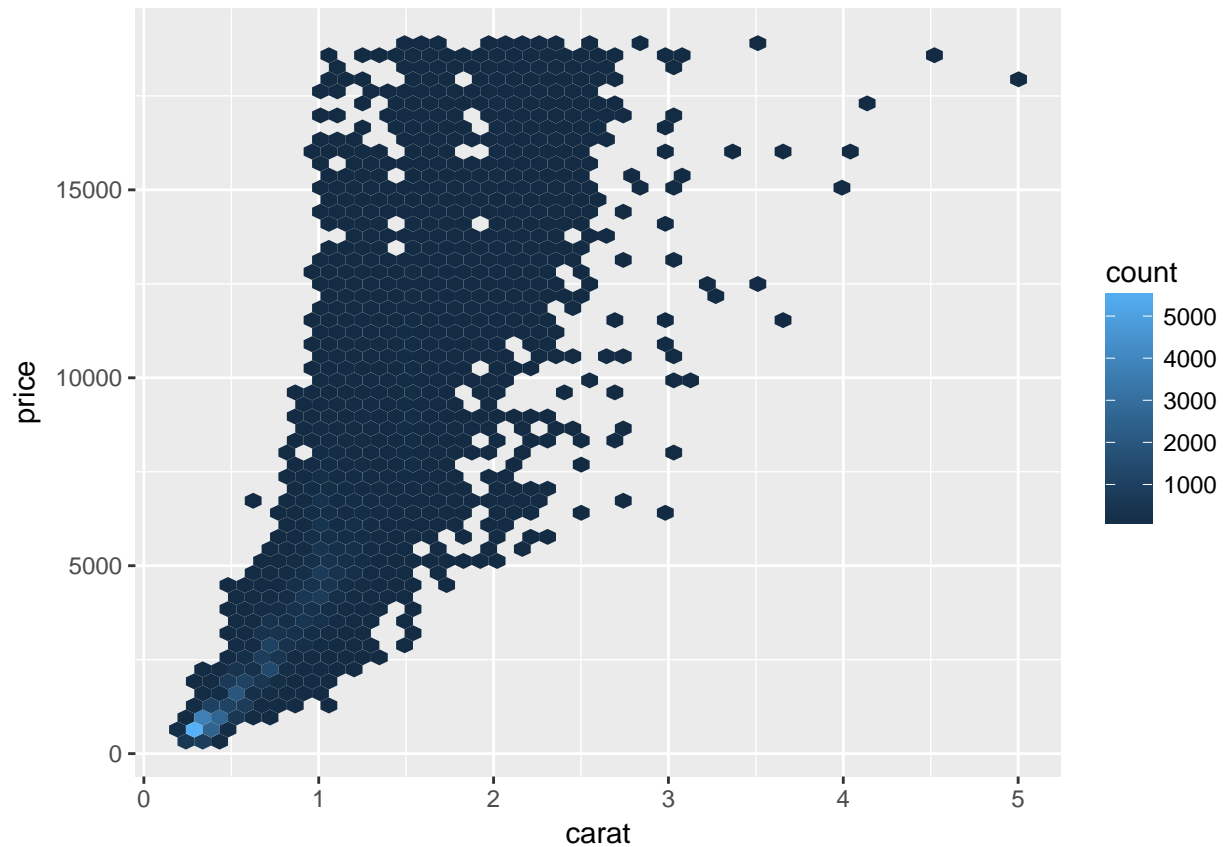


```
ggplot(diamonds, aes(clarity, price)) + geom_boxplot()
```



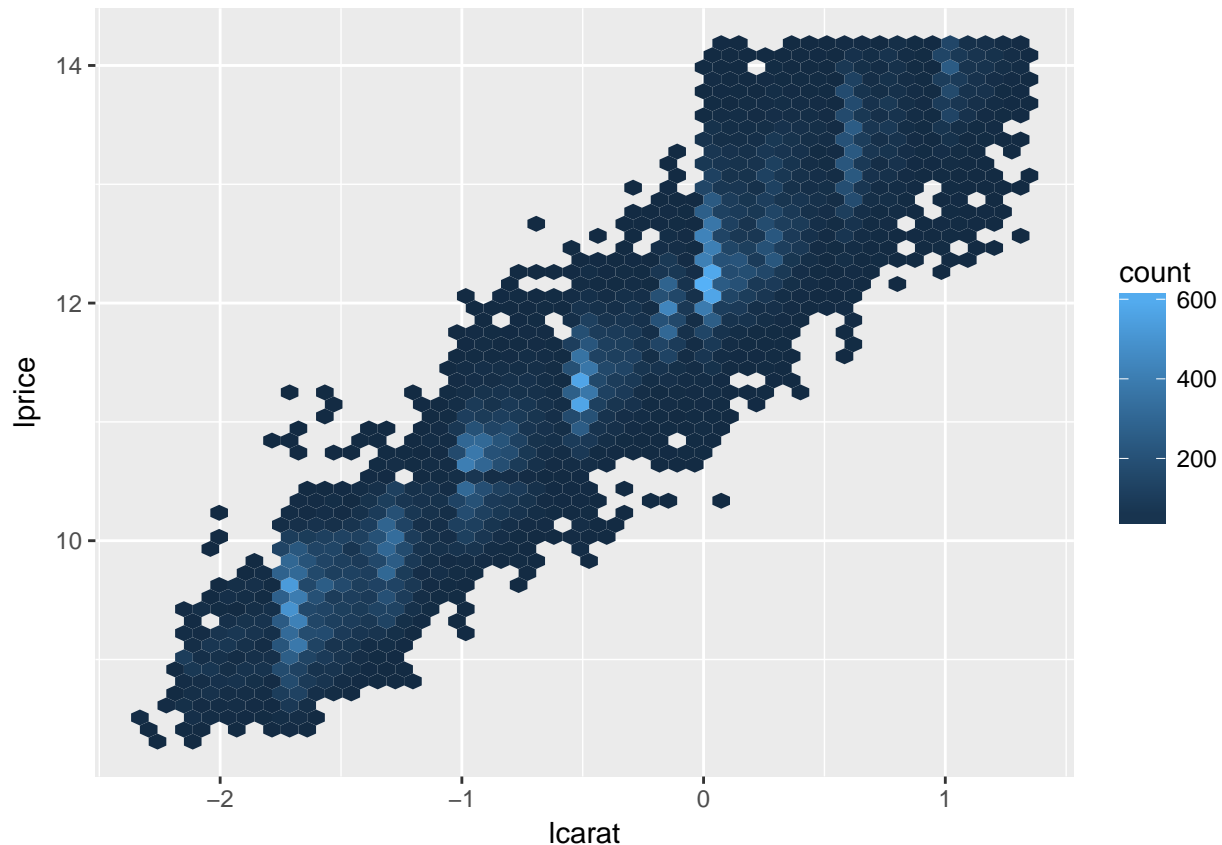
It looks like lower quality diamonds have higher prices because there is an important confounding variable: the weight `carat` of the diamond. The weight of the diamond is the single most important factor for determining the price of the diamond, and lower quality diamonds tend to be larger. To investigate this further, we try a slightly different visualization.

```
ggplot(diamonds, aes(carat, price)) + geom_hex(bins = 50)
```



We see two valuable trends in the above data. First, it appears that the `cost` is exponentially growing across the value of `carat`. This suggests that we could take a `log()` transform of both variable to obtain a linear relationship between these variables. We do this and also focus our attention on the diamonds that are 2.5 carats or more in weight. We then plot our transformed variables.

```
diamonds2 <- diamonds %>%  
  filter(carat <= 2.5) %>%  
  mutate(lprice = log2(price), lcarat = log2(carat))  
  
ggplot(diamonds2, aes(lcarat, lprice)) + geom_hex(bins = 50)
```



We see that this transformation did a really good job at providing a **linear** relationship between these two variables! All we have to do now is fit a linear model. This is readily done with the `lm()` function as follows:

Fitting a Linear Model

The `lm()` function

Let's first look at the help file for the `lm()` function. This function is the function to use if you are interested in fitting a **linear model**.

```
?lm
```

Important Arguments

- *x*: input design matrix
- *y*: response variable

We wish to model the response `lprice` against the predictor `lcarat` in the following linear model:

$$\log(\text{price}) = \beta_0 + \beta_1 \log(\text{carat}) + \epsilon$$

where above β_0 and β_1 are unknown **coefficient** values. The aim of the `lm` function is to estimate or fit these coefficient values. We see that next.

```
mod_diamond <- lm(lprice ~ lcarat, data = diamonds2)
```

`mod_diamond` above is a `lm` object, and contains a lot of information that can be used for inference, such as coefficient values, p-values that can be used to assess whether or not coefficients are statistically significant. To look at the coefficients, we can try a few of the following functions:

```
#get a summary of the fit
summary(mod_diamond)
```

```
##
## Call:
## lm(formula = lprice ~ lcarat, data = diamonds2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.96407 -0.24549 -0.00844  0.23930  1.93486
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 12.193863   0.001969   6194.5  <2e-16 ***
## lcarat       1.681371   0.001936    868.5  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3767 on 53812 degrees of freedom
## Multiple R-squared:  0.9334, Adjusted R-squared:  0.9334
## F-statistic: 7.542e+05 on 1 and 53812 DF,  p-value: < 2.2e-16
```

```
#get the coefficients of the fit
coef(mod_diamond)
```

```
## (Intercept)      lcarat
##  12.193863    1.681371
```

So, this is saying that the model we fit is given by the following linear regression:

$$\log(\text{price}) = 12.1939 + 1.6813 \log(\text{carat})$$

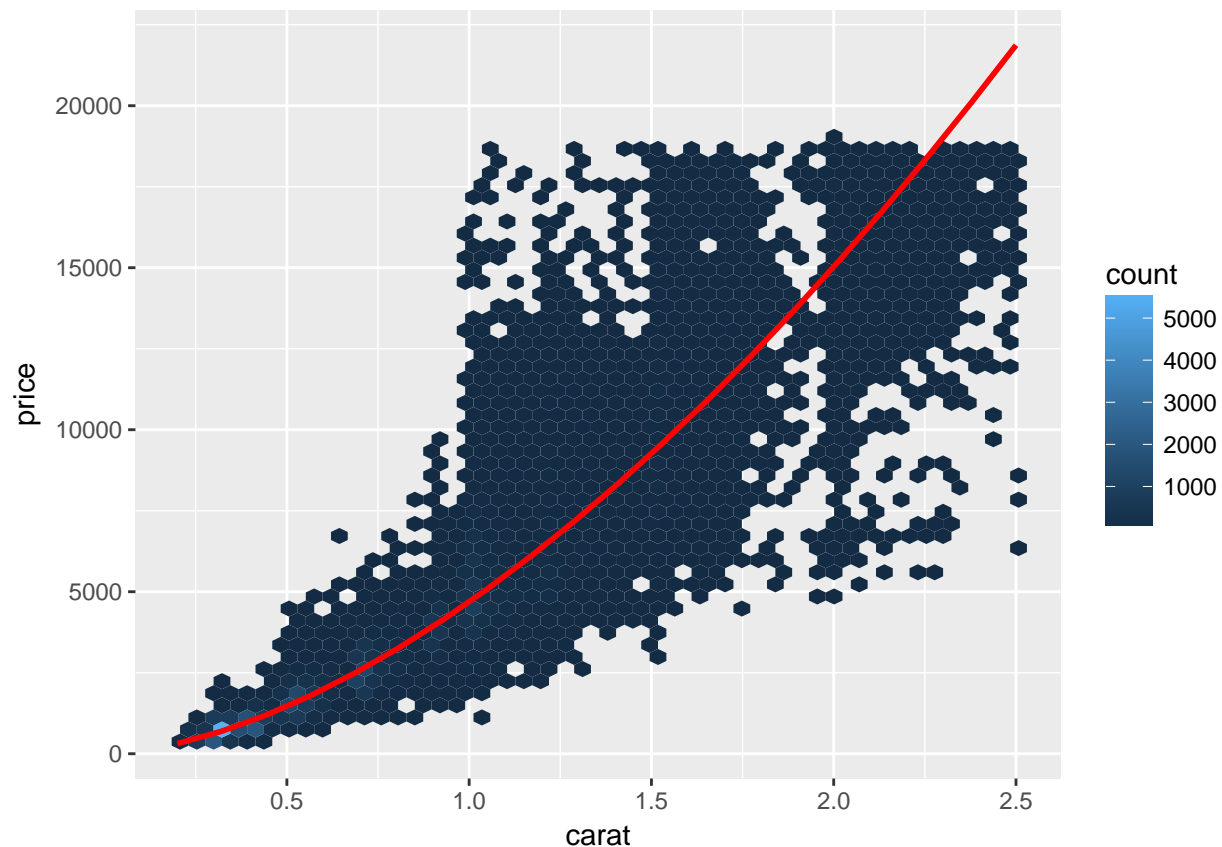
Also, from the `summary` results, we see that the p-values for both coefficients (represented by `Pr(>|t|)`) are less than $2e-16$, suggesting that each coefficient is statistically significant.

Secondly, we see that we get an Adjusted R^2 of 0.9334, meaning that the variable `carat` explains 93% of the variability of the price.

Visualize the fitted model to the data

```
#make predictions on for each carat value within the values that we observed (this is interpolation!) C
grid <- diamonds2 %>%
  data_grid(carat = seq_range(carat, 20)) %>%
  mutate(lcarat = log2(carat)) %>%
  add_predictions(mod_diamond, "lprice") %>%
  mutate(price = 2 ^ lprice)

#Plot the predictions (grid) on the original scale of the carat and price
ggplot(diamonds2, aes(carat, price)) +
  geom_hex(bins = 50) +
  geom_line(data = grid, colour = "red", size = 1)
```



Importantly, we see that our fitted model was a linear model on $\log(\text{price})$ against $\log(\text{carat})$. We interpolate values of `grid` by fitting our fitted model on the original `price` and `carat` scale, which it turns out, is an exponential model.

Residual Analysis

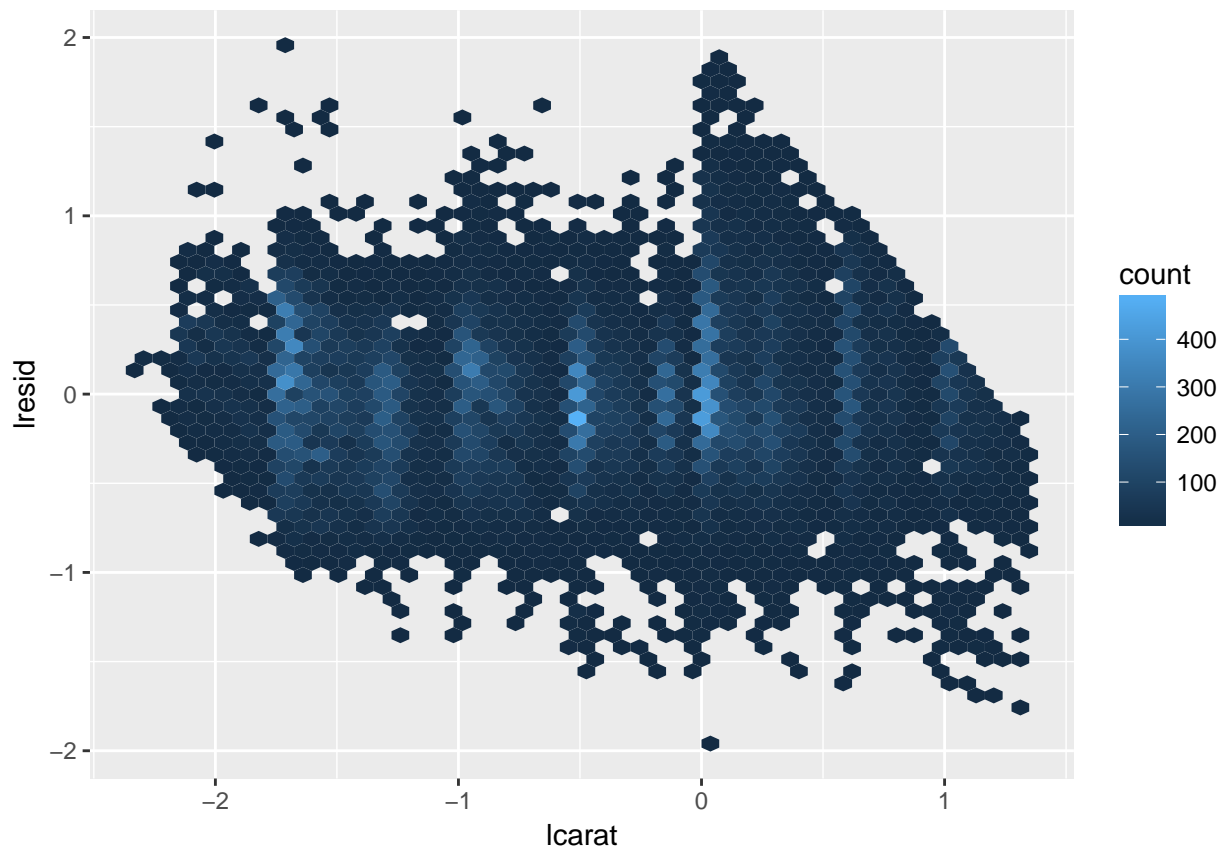
Once you've fit a model, you obtain predicted values \hat{y}_i for each response observation y_i . An important value to investigate is the **residual**, given by $\hat{y}_i - y_i$.

There are a lot of things we could look into here, but essentially, a good rule of thumb is: *your model is a good fit if your residuals are randomly scattered about 0*. That is, if you see any trend in the residuals, you may need to include another predictor to capture this variability.

To obtain and plot the residuals of this fitted model, we can use the following script.

```
diamonds2 <- diamonds2 %>%
  add_residuals(mod_diamond, "lresid")

ggplot(diamonds2, aes(lcarat, lresid)) +
  geom_hex(bins = 50)
```

This plot shows that there are some diamonds that have quite large residuals. It is advisable to dive more into the data more to manually investigate these diamonds. But we don't do it here for the sake of time. Other than that observation, the residuals look fairly randomly scattered about 0.

Models Part II: Shrinkage Methods

we will explore how to use three shrinkage methods:

- 1) Ridge Regression
- 2) The Lasso
- 3) Elastic Net

We will be utilizing an important R package *glmnet*, which was constructed by Hastie and Tibshirani to implement these three methods in a convenient way. Much of this lab is a replication of Section 6.6 in the “Introduction to Statistical Learning” textbook by James, Witten, Hastie, and Tibshirani (with my own commentary throughout).

Install Needed Packages

```
install.packages("glmnet", repos='http://cran.us.r-project.org')
```

```
## Installing package into '/Users/jdwilson4/Library/R/3.3/library'
## (as 'lib' is unspecified)
```

```
##
```

```
## The downloaded binary packages are in
```

```
## /var/folders/hm/8gnvskgx0rb1c11fzmz7sgf82j1yqg/T//RtmpXUU4z3/downloaded_packages
```

```
install.packages("ISLR", repos='http://cran.us.r-project.org')

## Installing package into '/Users/jdwilson4/Library/R/3.3/library'
## (as 'lib' is unspecified)

##
## The downloaded binary packages are in
## /var/folders/hm/8gnvskgx0rb1c11fzmz7sgf82j1yqg/T//RtmpXUU4z3/downloaded_packages
library(glmnet, quietly = TRUE)

##
## Attaching package: 'Matrix'

## The following object is masked from 'package:tidyr':
##
##      expand

##
## Attaching package: 'foreach'

## The following objects are masked from 'package:purrr':
##
##      accumulate, when

## Loaded glmnet 2.0-5
library(ISLR, quietly = TRUE)
```

The *glmnet()* function

Let's first look at the help file for the *glmnet()* function. This function is a workhorse for all three methods that we are interested in implementing. We'll come back to this over and over again for each method.

```
?glmnet
```

Important Arguments

- *x*: input design matrix
- *y*: response variable
- *family*: the family of regressions we want to use. For linear regression, we use “gaussian”; however, later we will look at other families such as “binomial” for logistic regression.
- *lambda*: the grid of possible λ values for regularization
- *alpha*: this function is created to naturally handle elastic net. The penalties are given by:

$$\frac{1-\alpha}{2}\lambda\ell_2^2(\beta) + \alpha\lambda\ell_1(\beta)$$

So, $\alpha = 1$ gives Lasso, and $\alpha = 0$ gives Ridge regression.

- *standardize*: logical indicating whether or not the covariates should be standardized to be on the same scale. This is important for interpretation of coefficients. The default is TRUE.

The Data

The data set that we will investigate is the *Hitters* data in the *ISLR* package. This data contains Major League Baseball statistics for players during the 1986 and 1987 seasons. Let's look a bit at the data

```
attach(Hitters)
?Hitters
names(Hitters)
```

```
## [1] "AtBat"      "Hits"       "HmRun"      "Runs"       "RBI"
## [6] "Walks"      "Years"      "CAtBat"     "CHits"      "CHmRun"
## [11] "CRuns"      "CRBI"       "CWalks"     "League"     "Division"
## [16] "PutOuts"    "Assists"    "Errors"     "Salary"     "NewLeague"
```

Now we'll check if there is any missing data.

```
sum(is.na(Hitters))
```

```
## [1] 59
```

Indeed, there are 59 missing entries in the *Hitters* data set. We could try to impute these data, but for simplicity, we choose to just ignore them and remove all entries from the data set.

```
Hitters <- na.omit(Hitters)
```

Our goal is to understand the relationship between a player's **Salary** and a player's other MLB characteristics throughout the 1986-1987 seasons. Let's prepare the response and design matrices for later analysis.

```
x <- model.matrix(Salary ~., data = Hitters)[, -1]
y <- Hitters$Salary
```

The *model.matrix()* function above automatically sets the classes of each covariate appropriately changing categorical data to *factors*. This will make the use of *glmnet()* much easier.

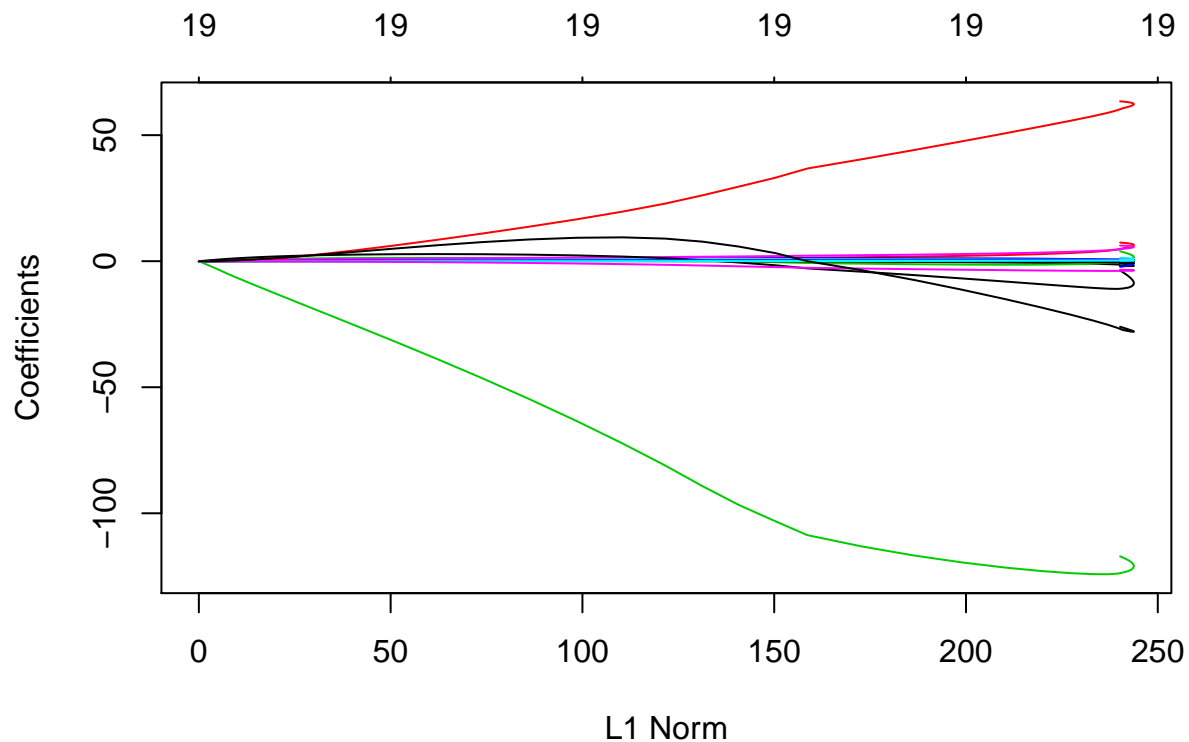
Ridge Regression

Now, we will run ridge regression of the *Salary* data against the remaining predictors in the *Hitters* data set. Note that first we specify a grid of λ values to search over. *glmnet()* will fit a model for each value of λ .

```
#First, set a grid of lambda to search over. We want to include lambda = 0 for standard linear regression
grid.lambda <- 10^seq(10, -2, length = 100)

#Fit the model across the grid of lambda values
ridge.model <- glmnet(x, y, alpha = 0, lambda = grid.lambda)

#Plot the L1 norm of the coefficients
plot(ridge.model)
```



Associated with each value of λ , we have coefficient estimates that can be viewed using the `coef()` function. As there are 100 values of λ and 20 coefficients, the coefficients are given in a 20×100 table. We look at an example of the coefficients and then calculate their ℓ_2 norm. We then do this across all lambda and plot the ℓ_2 norm at each value.

```
#Look at the 50th value of lambda
ridge.model$lambda[50]
```

```
## [1] 11497.57
```

```
#Coefficients for this value
coef(ridge.model)[, 50]
```

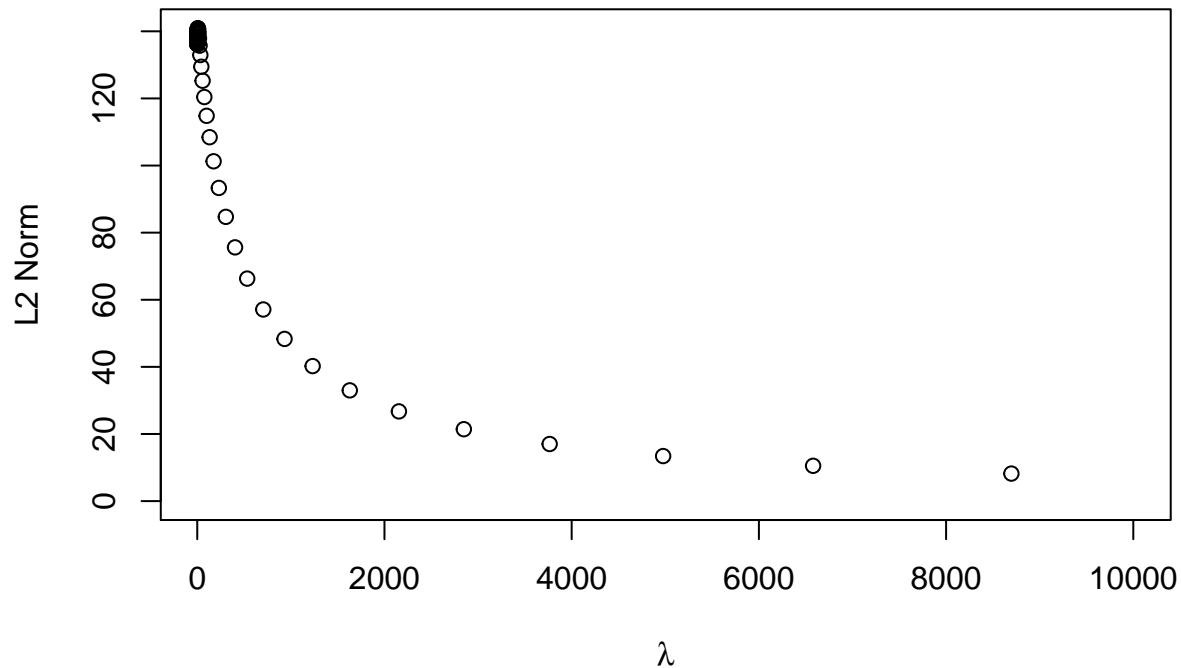
```
## (Intercept)      AtBat      Hits      HmRun      Runs
## 407.356050200  0.036957182  0.138180344  0.524629976  0.230701523
##      RBI      Walks      Years      CAtBat      CHits
##  0.239841459  0.289618741  1.107702929  0.003131815  0.011653637
##    CHmRun    CRuns    CRBI    CWalks    LeagueN
##  0.087545670  0.023379882  0.024138320  0.025015421  0.085028114
## DivisionW   PutOuts   Assists   Errors   NewLeagueN
## -6.215440973  0.016482577  0.002612988 -0.020502690  0.301433531
```

```
#Norm of these variables
sqrt(sum(coef(ridge.model)[-1, 50]^2))
```

```
## [1] 6.360612
```

```
#Let's repeat this for all lambda values and plot the results
ell2.norm <- numeric()
for(i in 1:length(grid.lambda)){
  ell2.norm[i] <- sqrt(sum(coef(ridge.model)[-1, i]^2))
}
```

```
plot(x = grid.lambda, y = ell2.norm, xlab = expression(lambda), ylab = "L2 Norm", xlim = c(10,10000))
```



Cross-Validation and Choosing the *best* tuning parameter

Using cross validation, we will now choose an optimal tuning parameter. To do so, we use the `cv.glmnet()`. By default, the function performs ten-fold cross-validation; however, we can change this by changing the argument `nfolds`. Let's see how this works.

```
set.seed(1) #for reproducibility
#Randomly select a training and test set.
#Here, we leave half of the data out for later model assessment
train <- sample(1:nrow(x), nrow(x) / 2)
test <- (-train)
y.train <- y[train]
y.test <- y[test]

#Now, fit a Ridge regression model to the training data
ridge.model.train <- glmnet(x[train, ], y.train, alpha = 0, lambda = grid.lambda)

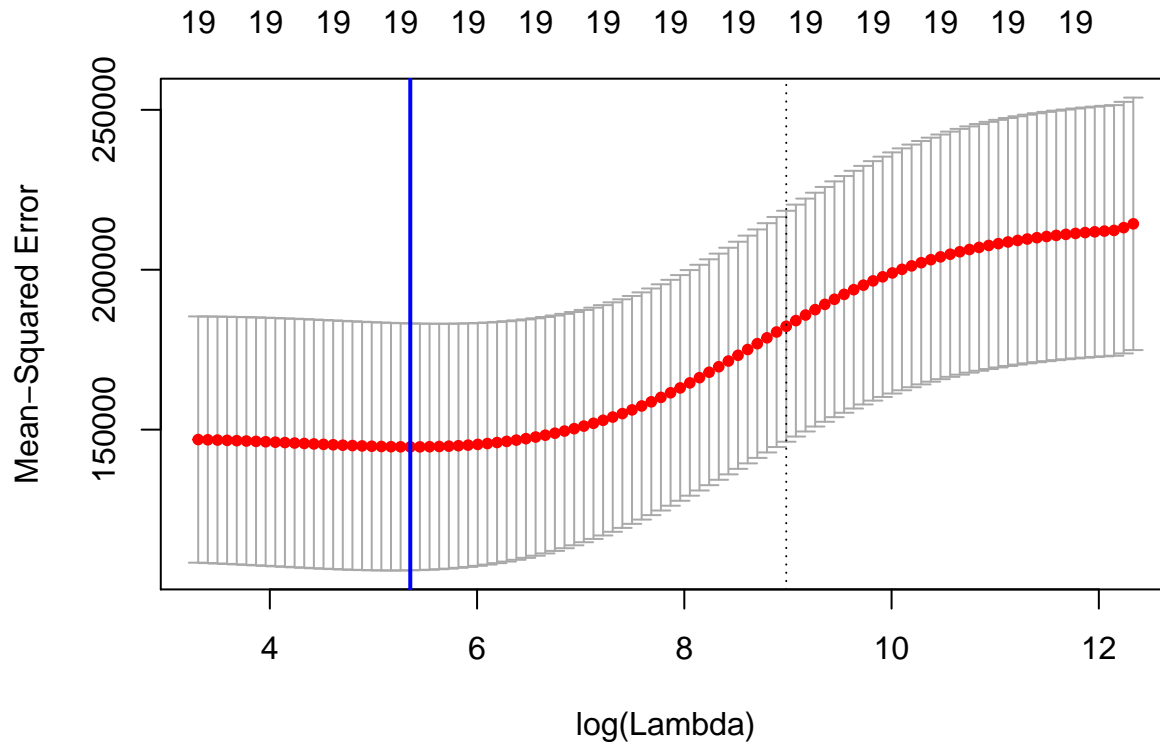
#Let's perform cross validation to choose the best model
?cv.glmnet

#Perform cross validation on the training set to select the best lambda
set.seed(1) #for reproducibility
cv.out <- cv.glmnet(x[train, ], y.train, alpha = 0)
plot(cv.out)

#Find the best lambda value
best.lambda <- cv.out$lambda.min
best.lambda
```

```
## [1] 211.7416
```

```
plot(cv.out)
abline(v = log(best.lambda), col = "blue", lwd = 2)
```



```
#Calculate the MSPE of the model on the test set
ridge.pred <- predict(ridge.model.train, s = best.lambda, newx = x[test, ])
mspe.ridge <- mean((ridge.pred - y.test)^2)
mspe.ridge
```

```
## [1] 96012.47
```

```
#Fit the final model to the entire data set using the chosen lambda
final.model <- glmnet(x, y, alpha = 0, lambda = best.lambda)
Coef.Ridge <- coef(final.model)[1:20, ]
Coef.Ridge
```

```
## (Intercept)      AtBat      Hits      HmRun      Runs
##  9.81383684  0.03174457  1.00837355  0.14062186  1.11296019
##           RBI      Walks      Years      CAtBat      CHits
##  0.87318566  1.80365302  0.13743548  0.01114900  0.06489415
##      CHmRun      CRuns      CRBI      CWalks      LeagueN
##  0.45152976  0.12888607  0.13726927  0.02919186  27.17300585
##  DivisionW      PutOuts      Assists      Errors      NewLeagueN
## -91.62094993  0.19145581  0.04247849 -1.81160164  7.22425927
```

Important Notes:

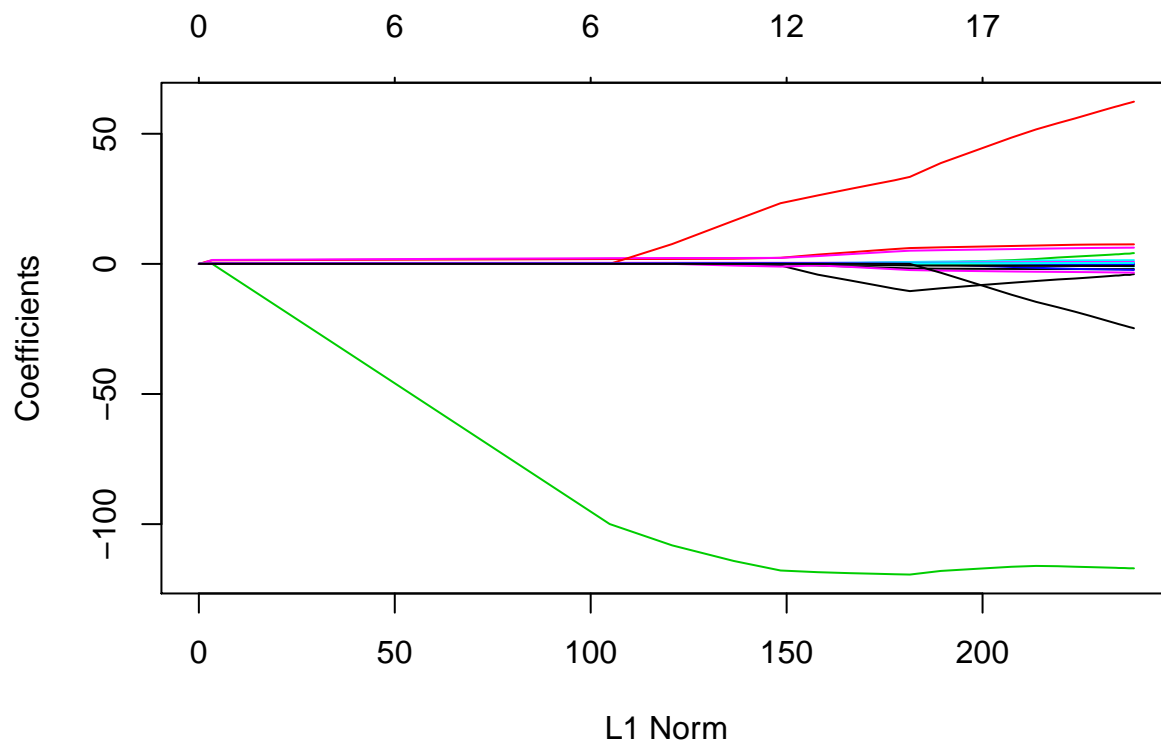
- 1) *glmnet()* does *not* give standard errors for estimates! To do this, one would need to re-fit using a linear regression or directly calculate these values using formula given in class.
- 2) As expected, Ridge regression does *not* shrink any variables exactly to 0! For this, we need to use something like the Lasso.

Lasso

Thankfully, `glmnet()` is readily available for Lasso and only requires the change of a single argument: *alpha*. For Lasso, we need to set *alpha* to 1 instead of 0 as done in the case for Ridge regression. Below, I put all of the code for running Lasso for demonstration. In reality, this is simply a copy-paste of the above code now changing *alpha* as needed. At the end of this section, we will discuss the similarities and differences between Lasso and Ridge regression.

Shrinkage Effects on the Entire Model

```
#First, let's look at the shrinkage effects of Lasso on the entire data set
lasso.model <- glmnet(x, y, alpha = 1, lambda = grid.lambda)
plot(lasso.model)
```



We already see that, unlike Ridge regression, there is a shrinkage effect over an interval of lambda values. This agrees with our theoretical discussion in class.

Now, let's fit the best model to the *Hitters* data and compare the results with Ridge regression. **Note**, we keep the training data and the test data the same so that we can have a fair comparison of methods.

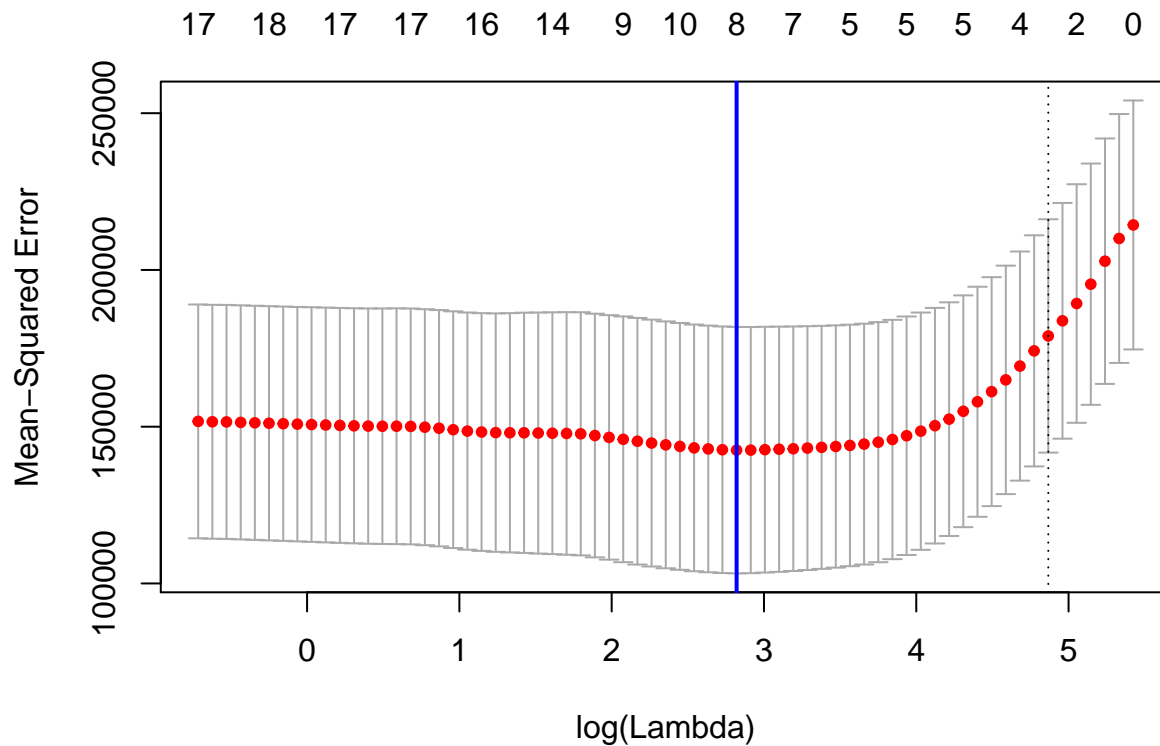
```
#Now, fit a Lasso regression model to the training data
lasso.model.train <- glmnet(x[train, ], y.train, alpha = 1, lambda = grid.lambda)

#Perform cross validation on the training set to select the best lambda
set.seed(1) #for reproducibility
cv.out <- cv.glmnet(x[train, ], y.train, alpha = 1)
plot(cv.out)

#Find the best lambda value
best.lambda <- cv.out$lambda.min
best.lambda
```

```
## [1] 16.78016
```

```
plot(cv.out)
abline(v = log(best.lambda), col = "blue", lwd = 2)
```



```
#Calculate the MSPE of the model on the test set
lasso.pred <- predict(lasso.model.train, s = best.lambda, newx = x[test,])
mspe.lasso <- mean((lasso.pred - y.test)^2)
mspe.lasso
```

```
## [1] 100743.4
```

```
#Fit the final model to the entire data set using the chosen lambda
final.model <- glmnet(x, y, alpha = 1, lambda = best.lambda)
Coef.Lasso <- coef(final.model)[1:20,]
Coef.Lasso
```

```
## (Intercept)      AtBat      Hits      HmRun      Runs
## 19.5052237    0.0000000    1.8702513    0.0000000    0.0000000
##      RBI      Walks      Years      CAtBat      CHits
## 0.0000000    2.2185101    0.0000000    0.0000000    0.0000000
##    CHmRun    CRuns    CRBI    CWalks    LeagueN
## 0.0000000    0.2075887    0.4125063    0.0000000    1.7600993
## DivisionW    PutOuts    Assists    Errors    NewLeagueN
## -103.4996975    0.2207019    0.0000000    0.0000000    0.0000000
```

Note that the final model here only includes 8 of the original 20 coefficients. Lasso did what we hoped! However, we see that the MSPE on the held-out test set (100743.4) was higher than that achieved by Ridge regression (96012.47). We see there is a tradeoff between prediction and interpretation of the model.

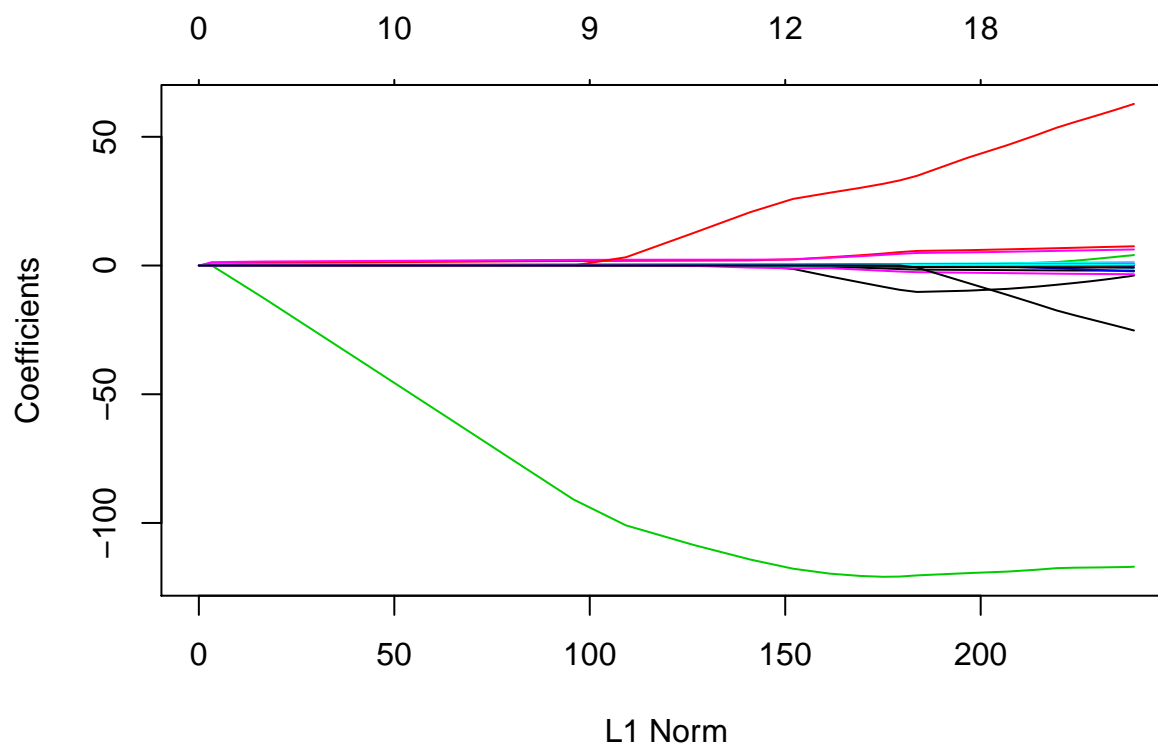
Elastic Net

Now, we repeat the same exercise above using a mixture of Lasso and Ridge regression methods. Here, we choose $\alpha = 0.50$ (equal parts L1 and L2 penalties). Note again that the only thing changing from the above is the setting of α .

Shrinkage Effects on the Entire Model

#First, let's look at the shrinkage effects of Lasso on the entire data set

```
EN.model <- glmnet(x, y, alpha = 0.5, lambda = grid.lambda)
plot(EN.model)
```



There is some shrinkage that occurs with Elastic Net. In fact, it appears that the amount of shrinkage is between that of Ridge (no shrinkage) and the Lasso.

Now, let's fit the best model to the *Hitters* data and compare the results with Ridge regression. **Note**, we keep the training data and the test data the same so that we can have a fair comparison of methods.

#Now, fit a Lasso regression model to the training data

```
EN.model.train <- glmnet(x[train, ], y.train, alpha = 0.5, lambda = grid.lambda)
```

#Perform cross validation on the training set to select the best lambda

```
set.seed(1) #for reproducibility
```

```
cv.out <- cv.glmnet(x[train, ], y.train, alpha = 0.5)
```

```
plot(cv.out)
```

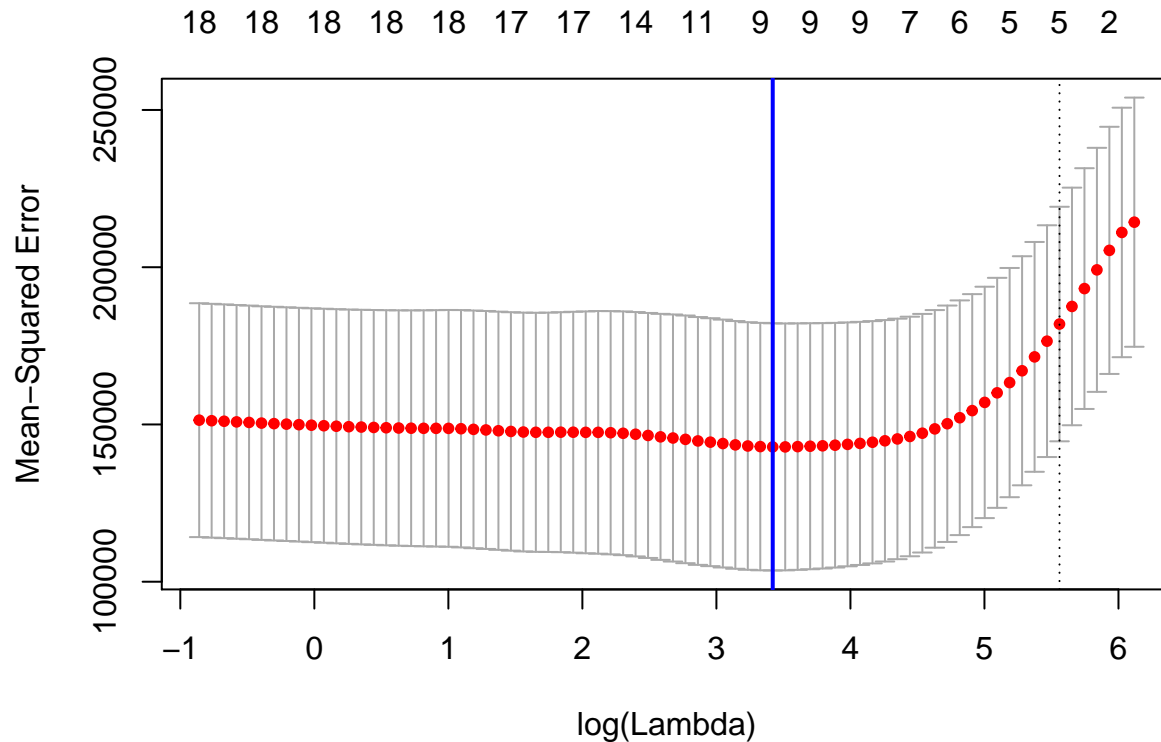
#Find the best lambda value

```
best.lambda <- cv.out$lambda.min
```

```
best.lambda
```

```
## [1] 30.57891
```

```
plot(cv.out)
abline(v = log(best.lambda), col = "blue", lwd = 2)
```



```
#Calculate the MSPE of the model on the test set
EN.pred <- predict(EN.model.train, s = best.lambda, newx = x[test,])
mspe.EN <- mean((EN.pred - y.test)^2)
mspe.EN
```

```
## [1] 100313.6
```

```
#Fit the final model to the entire data set using the chosen lambda
final.model <- glmnet(x, y, alpha = 0.5, lambda = best.lambda)
Coef.EN <- coef(final.model)[1:20,]
Coef.EN
```

```
##      (Intercept)      AtBat      Hits      HmRun      Runs
## 19.83656095      0.00000000      1.84038330      0.00000000      0.00000000
##           RBI           Walks           Years           CAtBat           CHits
## 0.00000000      2.24415298      0.00000000      0.00000000      0.04918383
##           CHmRun          CRuns          CRBI          CWalks          LeagueN
## 0.33652045      0.18517361      0.25805452      0.00000000      5.66345256
##      DivisionW      PutOuts      Assists      Errors      NewLeagueN
## -102.96577405      0.22081427      0.00000000      0.00000000      0.00000000
```

In the final model, we have 10 of the original 20 predictors.

Comparison of the three models Let's compare the coefficients of each of these three models and the MSPE of each.

```
Coefficients <- data.frame(Ridge = Coef.Ridge, Lasso = Coef.Lasso, Elastic.Net = Coef.EN)
MSPE <- data.frame(Ridge = mspe.ridge, Lasso = mspe.lasso, Elastic.Net = mspe.EN)
```

```
Coefficients
```

##	Ridge	Lasso	Elastic.Net
## (Intercept)	9.81383684	19.5052237	19.83656095
## AtBat	0.03174457	0.0000000	0.00000000
## Hits	1.00837355	1.8702513	1.84038330
## HmRun	0.14062186	0.0000000	0.00000000
## Runs	1.11296019	0.0000000	0.00000000
## RBI	0.87318566	0.0000000	0.00000000
## Walks	1.80365302	2.2185101	2.24415298
## Years	0.13743548	0.0000000	0.00000000
## CAtBat	0.01114900	0.0000000	0.00000000
## CHits	0.06489415	0.0000000	0.04918383
## CHmRun	0.45152976	0.0000000	0.33652045
## CRuns	0.12888607	0.2075887	0.18517361
## CRBI	0.13726927	0.4125063	0.25805452
## CWalks	0.02919186	0.0000000	0.00000000
## LeagueN	27.17300585	1.7600993	5.66345256
## DivisionW	-91.62094993	-103.4996975	-102.96577405
## PutOuts	0.19145581	0.2207019	0.22081427
## Assists	0.04247849	0.0000000	0.00000000
## Errors	-1.81160164	0.0000000	0.00000000
## NewLeagueN	7.22425927	0.0000000	0.00000000

MSPE

##	Ridge	Lasso	Elastic.Net
## 1	96012.47	100743.4	100313.6

- 1) The coefficients kept by Lasso are a subset of those kept by Elastic Net.
- 2) Ridge regression keeps all coefficients (no shrinkage)
- 3) The MSPE is smaller for models with higher p . This shows a tradeoff between complexity and performance of the models.

Key Point: Linear regression and other statistical models will be covered much more thoroughly in the followup classes **MATH 372: Linear Regression** and **MATH 373: Statistical Learning** taught by Prof. Stevens and myself next Fall and Spring. So take these classes!