Lab Assignment 3: Quant III
Kyle Davis

Question 1:
A new DGP:

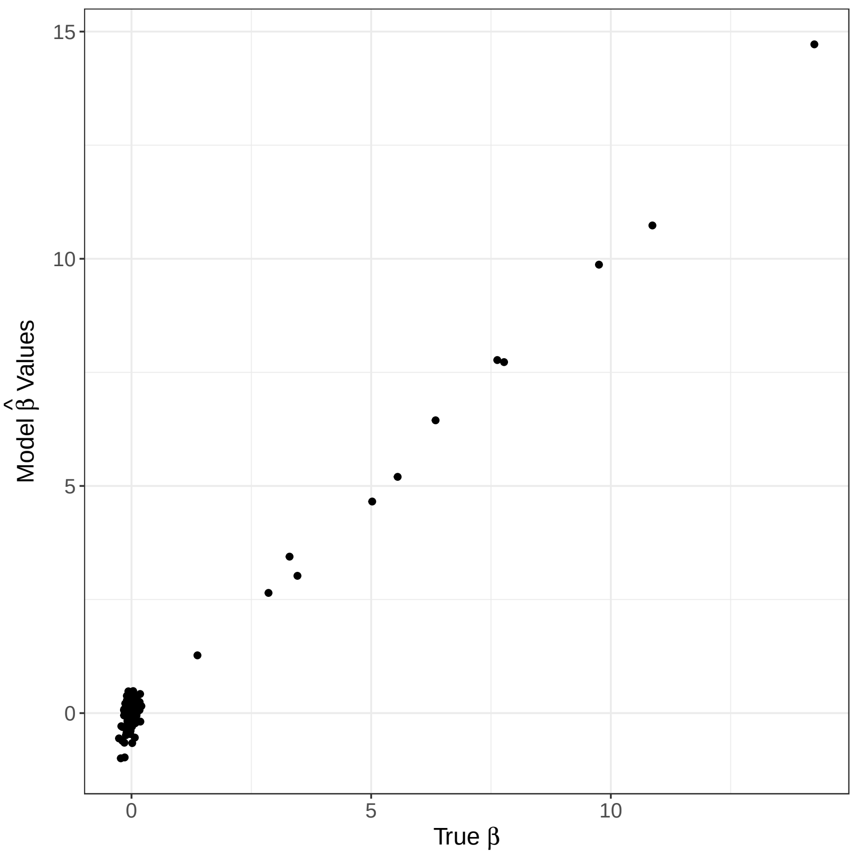
```
set.seed(12345)
library(caret)  # for train()
library(MASS)   # mvrnorm()
library(Rlab)   # for rbern()

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

And let's use the new R function `createDataPartion()` to make a test set and training set:

```
##   Test Set (20%); Training Set (80%)
# Get 800 random index from y
trainidx <- createDataPartition(y, times = 1, p=.80, list=FALSE)
trainy   <- y[trainidx]   # Get traniing set index numbers; sample y's values
testy    <- y[-trainidx]  # Other 20% for test set (y)

trainx   <- X[trainidx,]  # same for setting up X variable:
testx    <- X[-trainidx,]

# Create Data Frames for these:
train_data = data.frame(trainx, trainy)
test_data  = data.frame(testx, testy)
```

Quesiton 2:
Let's use a quick plot to compare our linear model $\hat{\beta}$ to what β actually is in the DGP:

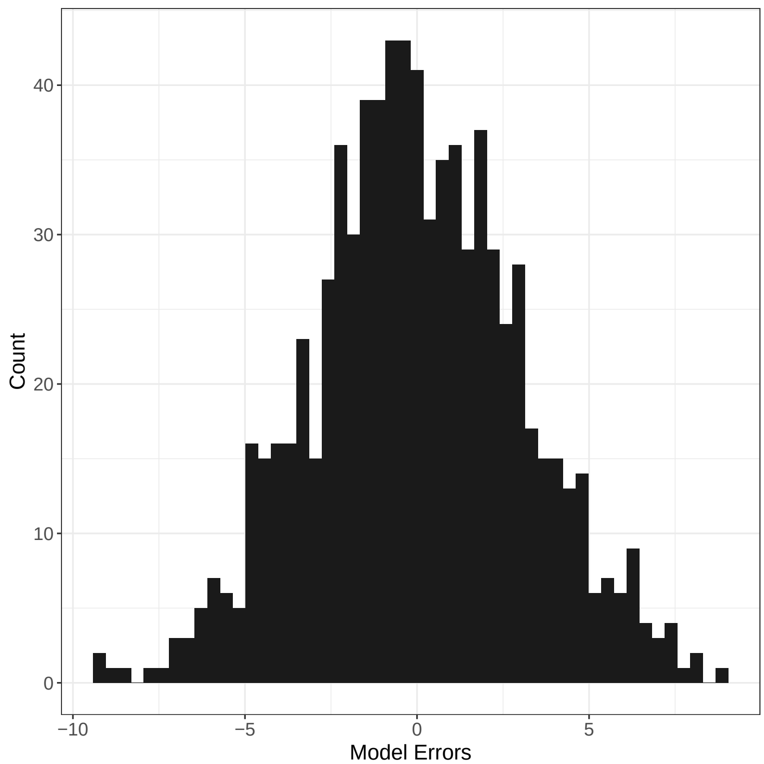
```
library(ggplot2) # Graphing

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



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

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

error    <- train_data$trainy - predict(trainmod)
```

```r
# head(error)  # Check this makes sense

#  Ideally we want normality here, mean zero
qplot(error, bins=50, fill=I("grey10"))+
  xlab( "Model Errors")+
  ylab( "Count")+
  theme_bw()+
  theme(axis.text=element_text(size=12),
        axis.title=element_text(size=14))
```




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

## [1] 3.039912

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

## [1] 2.415841
```

Using just our training data with the linear model we see our root mean squared error and mean absolute error are about around 3, pretty low. More intutitvely, our errors when plotted are normally distributed with a mean of zero.

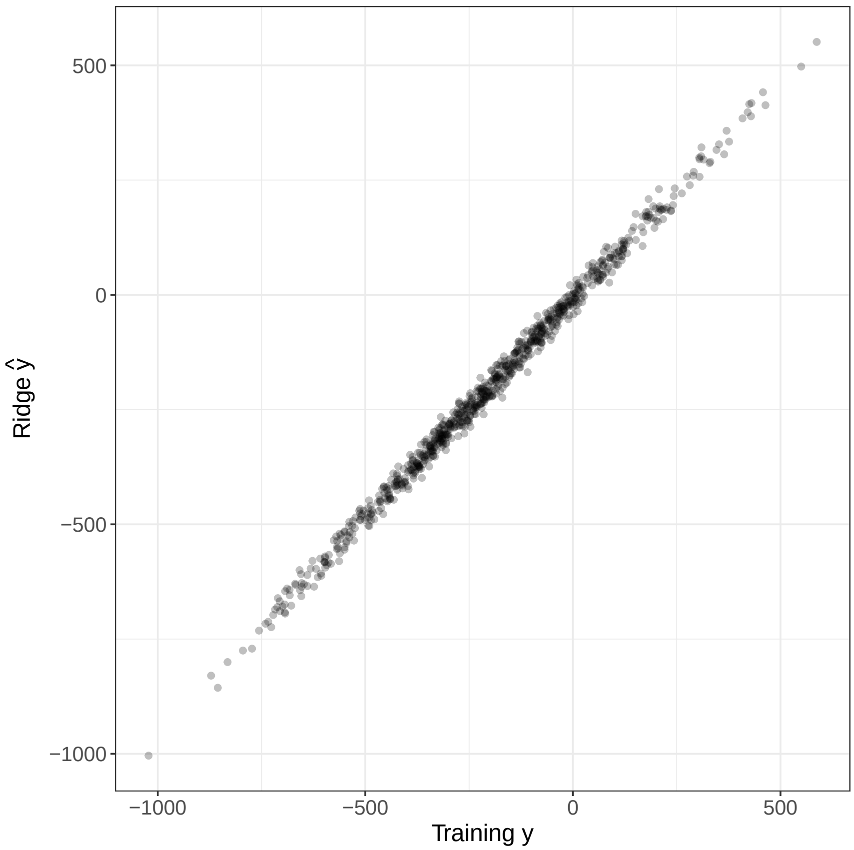
Question 3:
Using penalized regression is especially helpful with models that have a large number of predictors relative to our sample size. The “elastic net” is particularly useful because it “switches” between two other penalties (using α), the “LASSO” (Least Aboslute Shrinkage and Selection Operator uses a squared transformation) and “Ridge” (using the absolute value of our coefficents), applying both of these in an ever-checking gradient descent down our distribution to the point of least error. Ideally we would like a smooth distribution in this process instead of quick-changing erratic distribution and the elastic net provides the previously mentioned functions to accomplish this via the transformation of our coefficents as they travel farther away from zero. In my research this could be useful when using machine learning for political media bias, as I could have over a thousand paramenters using texual data.[1]

Question 4:
Running a LASSO:

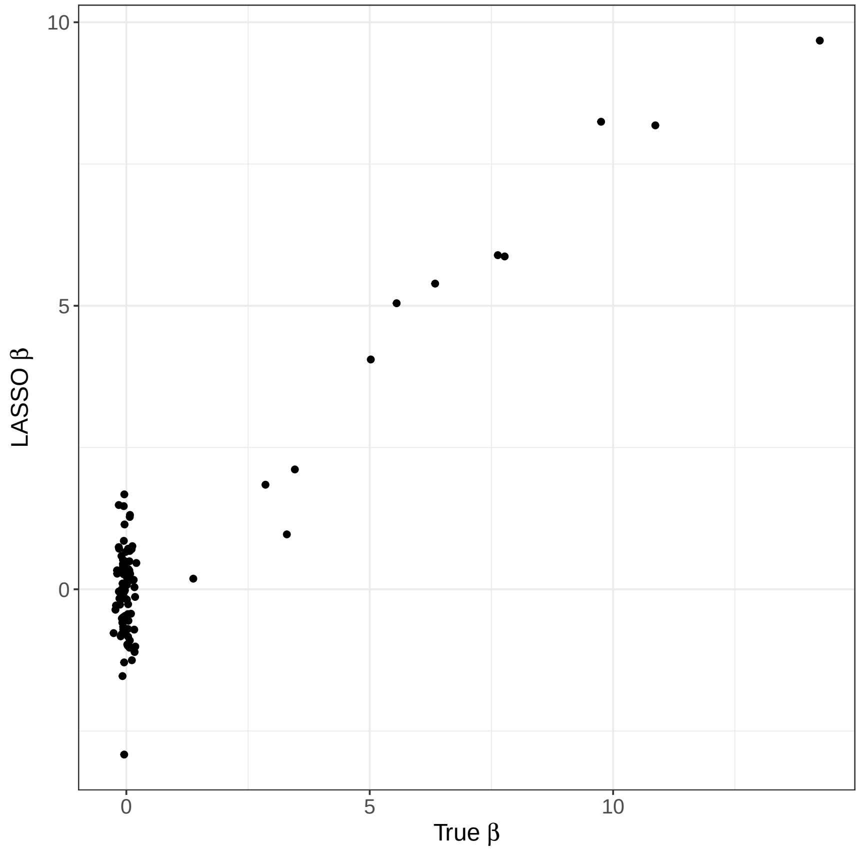
```
# Remember our training data frame:
train_data = data.frame(trainx, trainy)

mod_lasso = train(trainy~., method="glmnet",# formula and method
                  tuneGrid=expand.grid(alpha=0, # alpha=0 is the LASSO
                          lambda=seq(0,100,1)), #gradient descent to zero
                  data=train_data,
                  preProcess=c("center"),# centering standardization of points
                  trControl=trainControl(method="cv",number=2, search="grid"))
# "trainControl" sets a resampling method per a "search" parameter grid

# mod_lasso  ## final values were around lambda = 14 using RMSE


## Narrowing lambda sequence closer to what the lambda reported earlier:
mod_lasso = train(trainy~., method="glmnet",
                  tuneGrid=expand.grid(alpha=0,
                  # *setting bounds closer around 14; smaller "steps" (0.01)
                           lambda=seq(0,40,0.01)),
                  data=train_data,
                  preProcess=c("center"),
                  trControl=trainControl(method="cv",number=2, search="grid"))

# mod_lasso  ## Narrowing in, we found a new best lambda of around 14.5!


## Predictions:
```

1. A more technical explaination will come in Question 5 when explaining model fit. I’m not sure if the explaination in Question 5 is something I would ever use to explain to my grandma, so Question 3 stands as is.

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



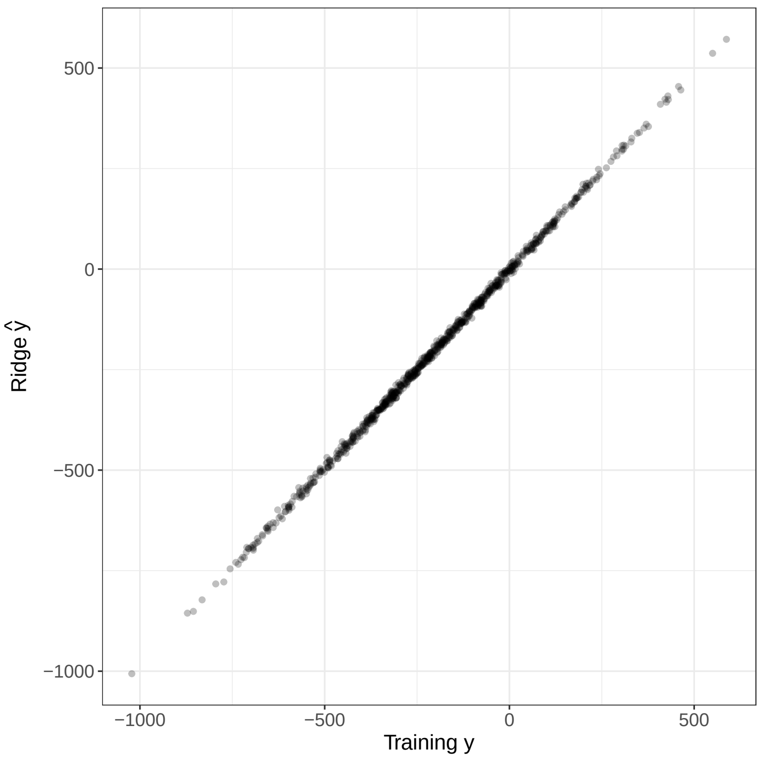

```
# best values using best lambda:
lasso_beta = coef(mod_lasso$finalModel, mod_lasso$bestTune$lambda)
qplot(b, lasso_beta[-1])+ #removing intercept
  xlab( expression(paste("True ", beta)))+
  ylab( expression(paste("LASSO ", beta)))+
  theme_bw()+
  theme(axis.text=element_text(size=12),
        axis.title=element_text(size=14))
```



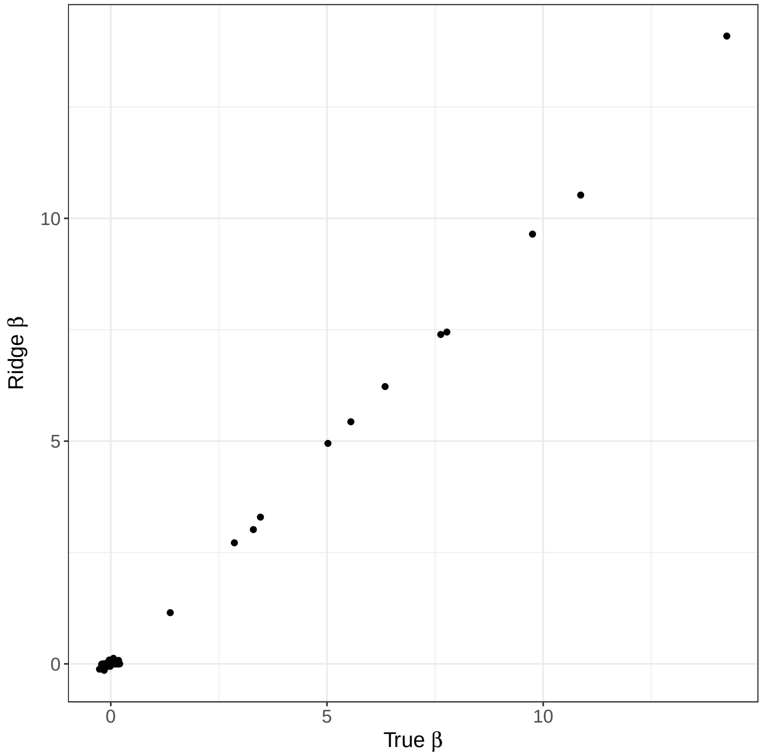
What this tells us is that our predicted values from the LASSO model does well with our training y data. Specifically, in the first figure we see that our training dependent variable and our penalized LASSO predictions match with each other positively. In the second figure, we see our actual beta's more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

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

# Best lambda (RMSE) is 1.01

mod_ridge = train(trainy~., method="glmnet",
                  tuneGrid=expand.grid(alpha=1,
                          lambda=seq(0,3,0.01)),
                  data=train_data,
                  preProcess=c("center"),
                  trControl=trainControl(method="cv",number=2, search="grid"))
```

```r
# New best lambda of .94

yhat = predict(mod_ridge)
qplot(trainy, yhat, alpha=I(0.25))+
  xlab( expression(paste("Training ", y)))+
  ylab( expression(paste("Ridge ", hat(y))))+
  theme_bw()+
  theme(axis.text=element_text(size=12),
        axis.title=element_text(size=14))
```




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



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

Ridge: ($\alpha = 0$)

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

LASSO: ($\alpha = 1$)

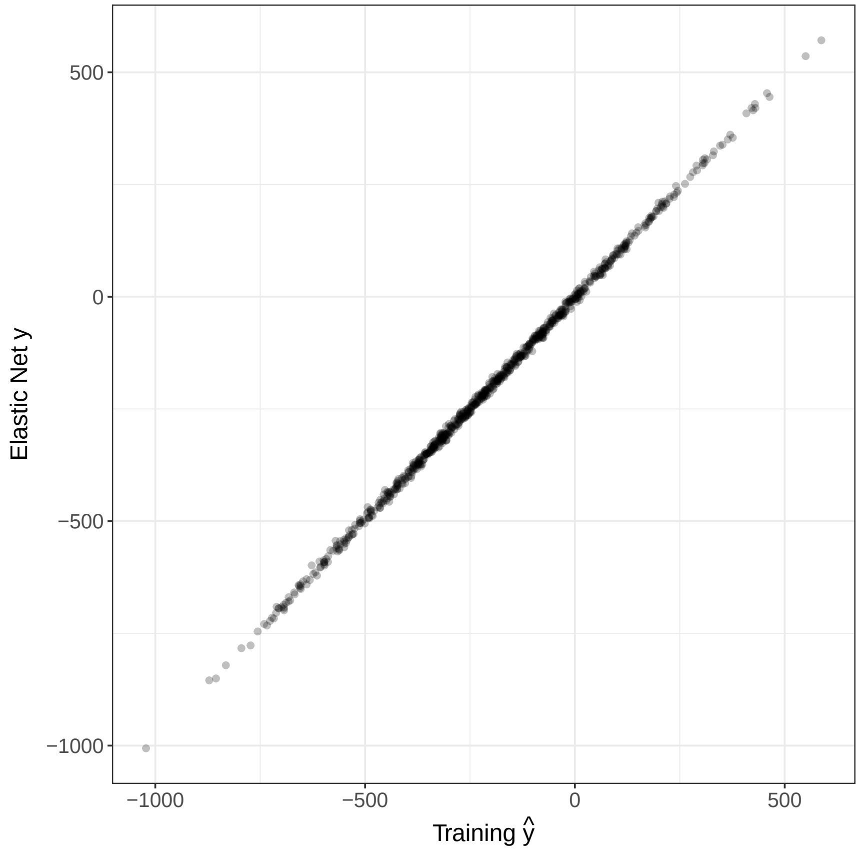
$$\lambda \sum_{j=1}^{p} (\alpha)|\beta_j| \tag{2}$$

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

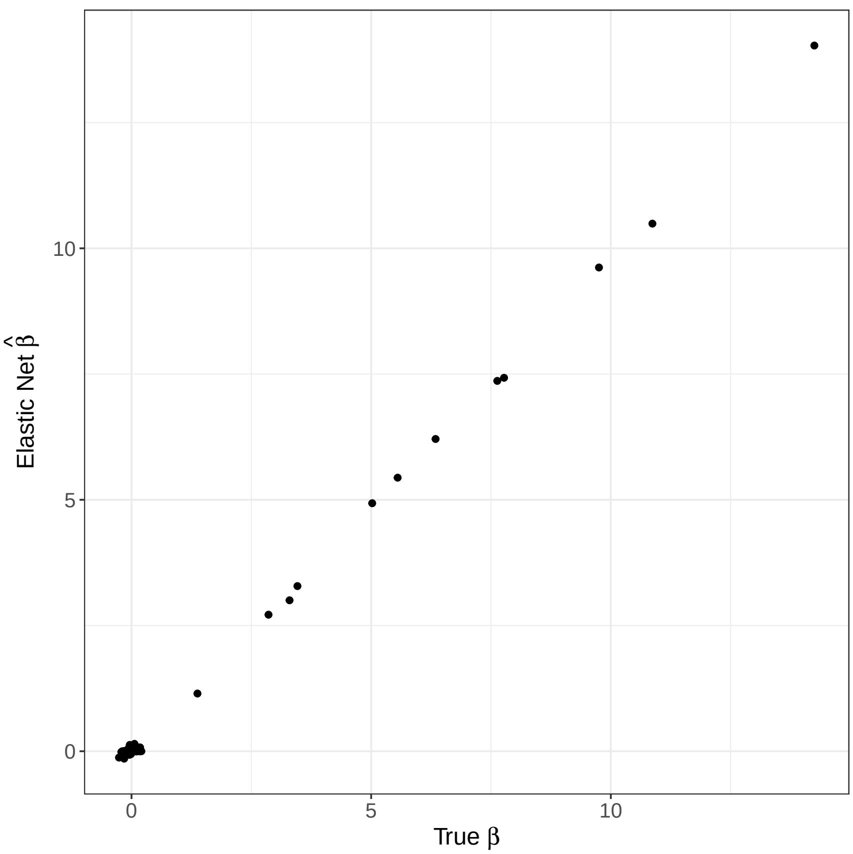
Elastic Net: ($0 < \alpha < 1$)

$$\lambda \sum_{j=1}^{p} \left[\frac{1}{2}(1-\alpha)\beta^2 \right] + \lambda \sum_{j=1}^{p} (\alpha)|\beta_j| \tag{3}$$

```r
## Elastic Net:
mod_enet = train(trainy~., method="glmnet",
                 tuneGrid=expand.grid(alpha=seq(0,1,0.1),
                                      lambda=seq(0,100,1)),
                 data=train_data,
                 preProcess=c("center"),
                 trControl=trainControl(method="cv",number=2, search="grid"))

# alpha used was .9 and lambda ended up being 1.


mod_enet = train(trainy~., method="glmnet",
                 tuneGrid=expand.grid(alpha=seq(0.6,1,0.5),
                                        lambda=seq(0,10,0.5)),
                 data=train_data,
                 preProcess=c("center"),
                 trControl=trainControl(method="cv",number=2, search="grid"))

# alpha ended up at .6 and lambda at 1.

yhat = predict(mod_enet)
qplot(trainy, yhat, alpha=I(0.25))+
  xlab( expression(paste("Training ", hat(y))))+
  ylab( expression(paste("Elastic Net ", y)))+
  theme_bw()+
  theme(axis.text=element_text(size=12),
        axis.title=element_text(size=14))
```




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

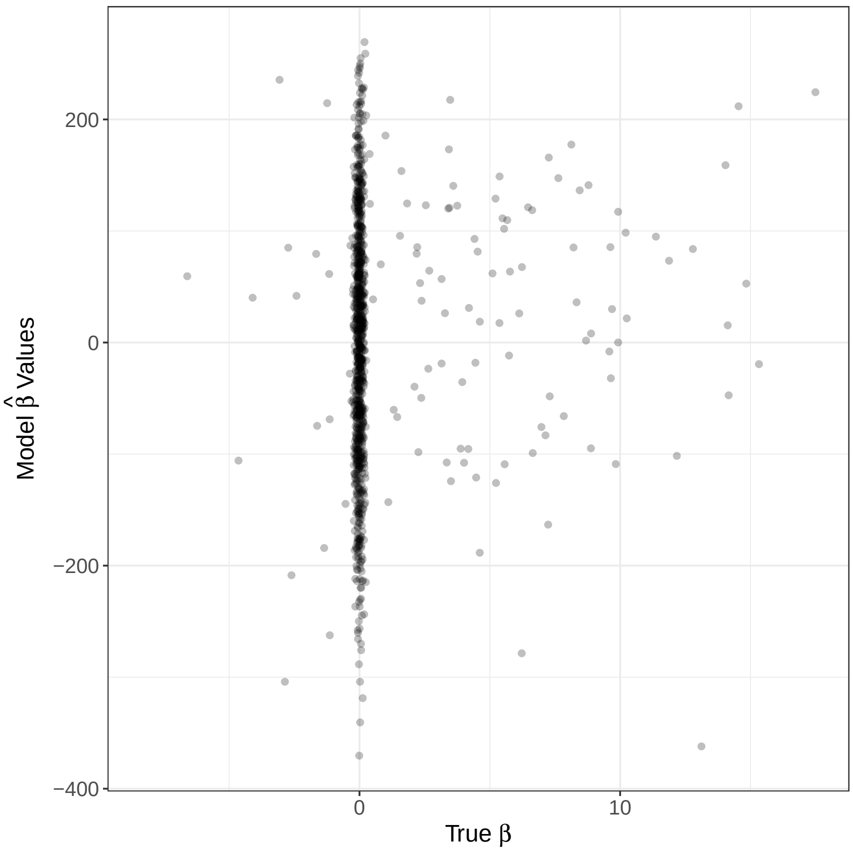


In the elastic net model we see slightly more variance in our compared beta's but our results are still very robust with little variance overall. Our α parameter ended up narrowing in around .6; which sugguests to me that using the elastic net was probably the best choice instead of fixing the model to either LASSO (at 1) or Ridge (at 0). The following question asks us to increase our number of parameters and this will likely change these results.

Question 5:
Consider having 1500 parameters with only an N of 1000:

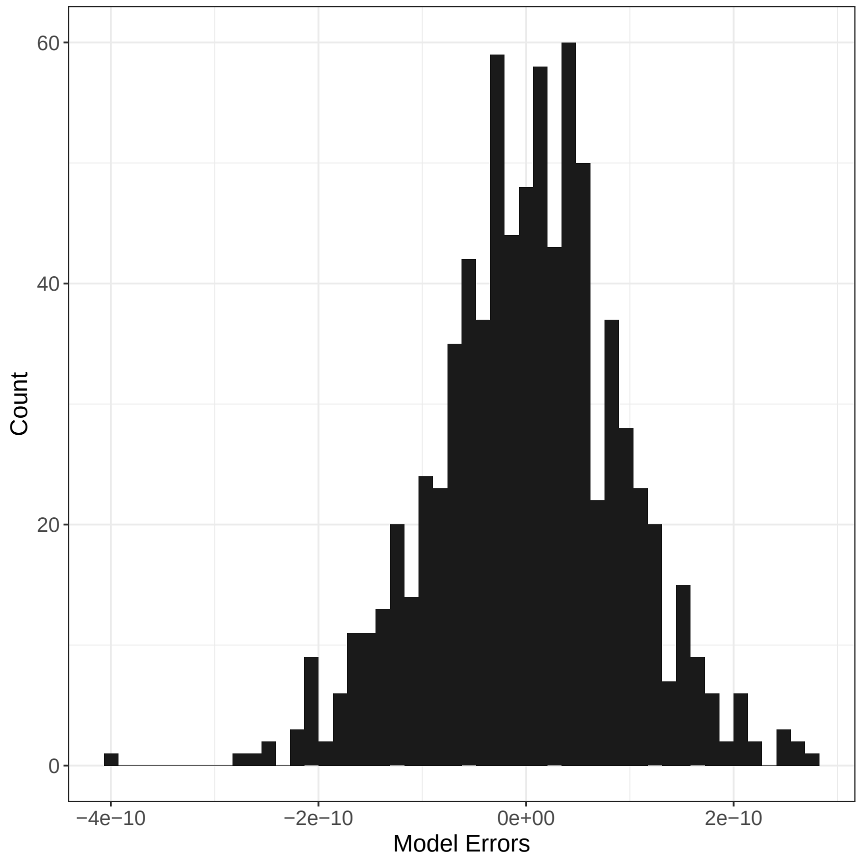
```
N     <- 1000
P     <- 1500 # Note this change

Sigma <- rWishart(n=1, df=P, Sigma=diag(P))[,,1]
X     <- mvrnorm(N, runif(P, -10, 10), Sigma)
p     <- rbern(P, 0.1)
b     <- p * rnorm(P, 5, 5) + (1-p) * rnorm(P, 0, 0.1)
e     <- rnorm(N, 0, sd=sqrt(10))
y = X%*%b + e
m1    <- lm(y~X)

my_data = data.frame(X,y)
```

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



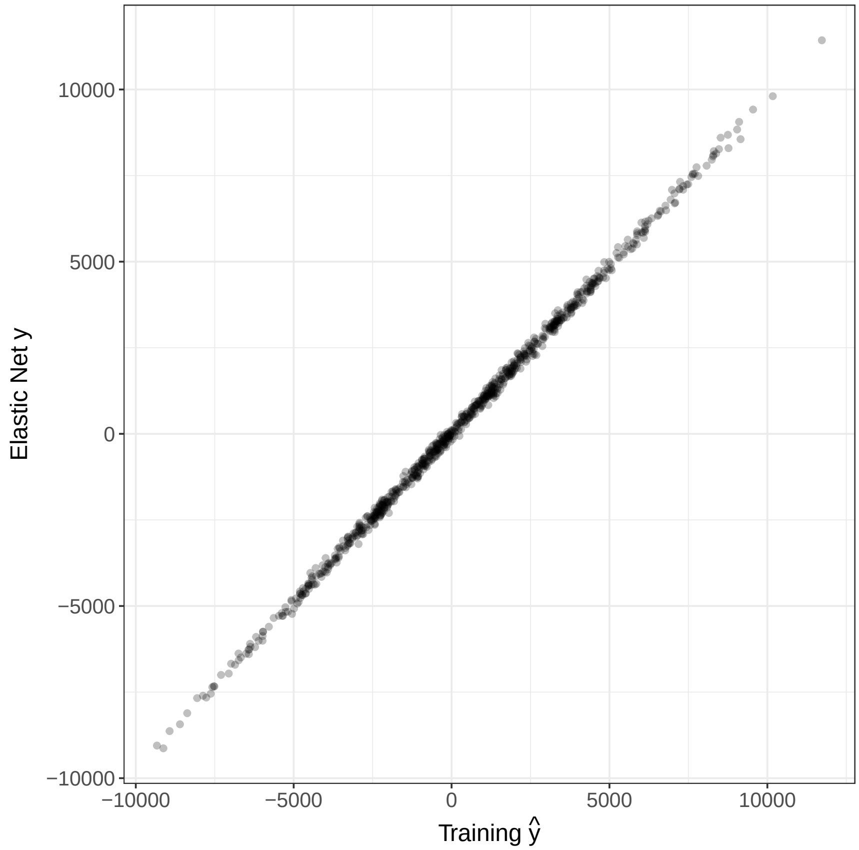

```r
#  Test Set; Training Set (20% and 80% again)
trainidx <- createDataPartition(y, times = 1, p=.80, list=FALSE)
trainy   <- y[trainidx]
testy    <- y[-trainidx]

trainx   <- X[trainidx,]
testx    <- X[-trainidx,]

train_data = data.frame(trainx, trainy)
test_data  = data.frame(testx, testy)

# Training linear Model:
trainmod <- lm(train_data$trainy ~ trainx, data = train_data)
```

```r
error     <- train_data$trainy - predict(trainmod)

## Ideally we want normality here, mean zero
qplot(error, bins=50, fill=I("grey10"))+
  xlab( "Model Errors")+
  ylab( "Count")+
  theme_bw()+
  theme(axis.text=element_text(size=12),
        axis.title=element_text(size=14))
```

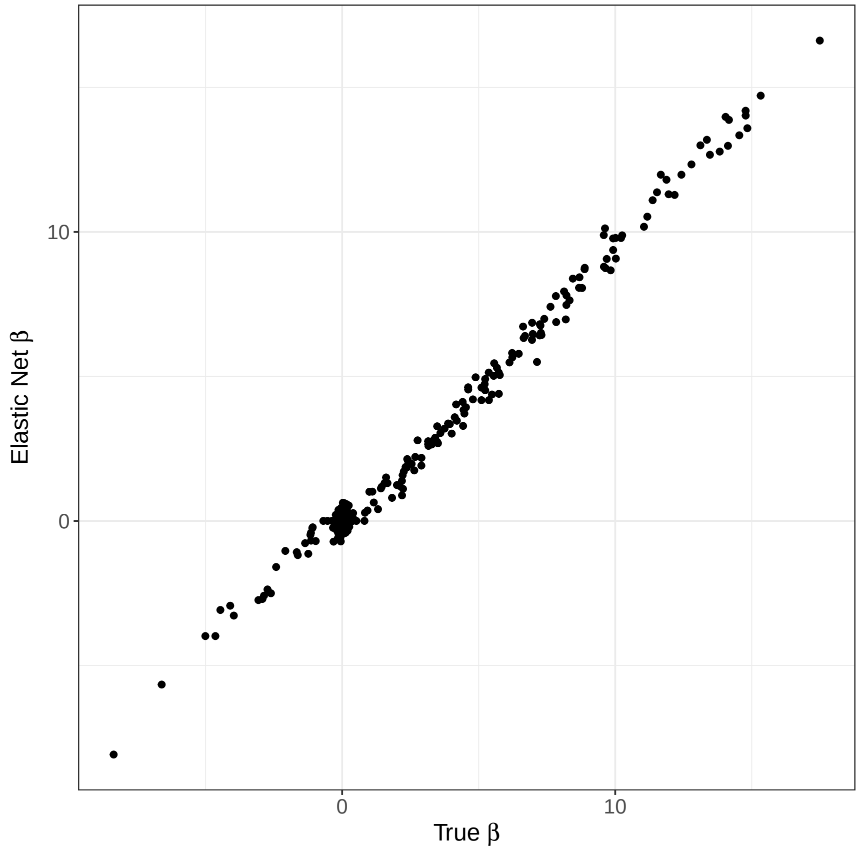



```r
# Root Mean Squred Error
sqrt(mean(error^2))

## [1] 9.043963e-11

# Mean Aboslute Error
mean(abs(error))

## [1] 7.054893e-11

### Very significant!
```

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

mod_enet = train(trainy~., method="glmnet",
                 tuneGrid=expand.grid(alpha=seq(0.6,1,0.5),
                                       lambda=seq(0,10,0.5)),
                 data=train_data,
                 preProcess=c("center"),
                 trControl=trainControl(method="cv",number=2, search="grid"))
# alpha ended up at .6 and lambda at 10.


yhat = predict(mod_enet)
qplot(trainy, yhat, alpha=I(0.25))+
  xlab( expression(paste("Training ", hat(y))))+
  ylab( expression(paste("Elastic Net ", y)))+
  theme_bw()+
  theme(axis.text=element_text(size=12),
        axis.title=element_text(size=14))
```




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



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