Lasso and Ridge Regression:

1. **In this exercise, using simulated data, I have performed the lasso.**
   a. Using the rnorm function have generated 30 values for x and ep (noise). Have also included the libraries needed and set the seed to 1.

```
library(leaps)
library(ISLR)
library(glmnet)
set.seed (1)

#1.a
x= rnorm(30)
ep=rnorm(30)
```

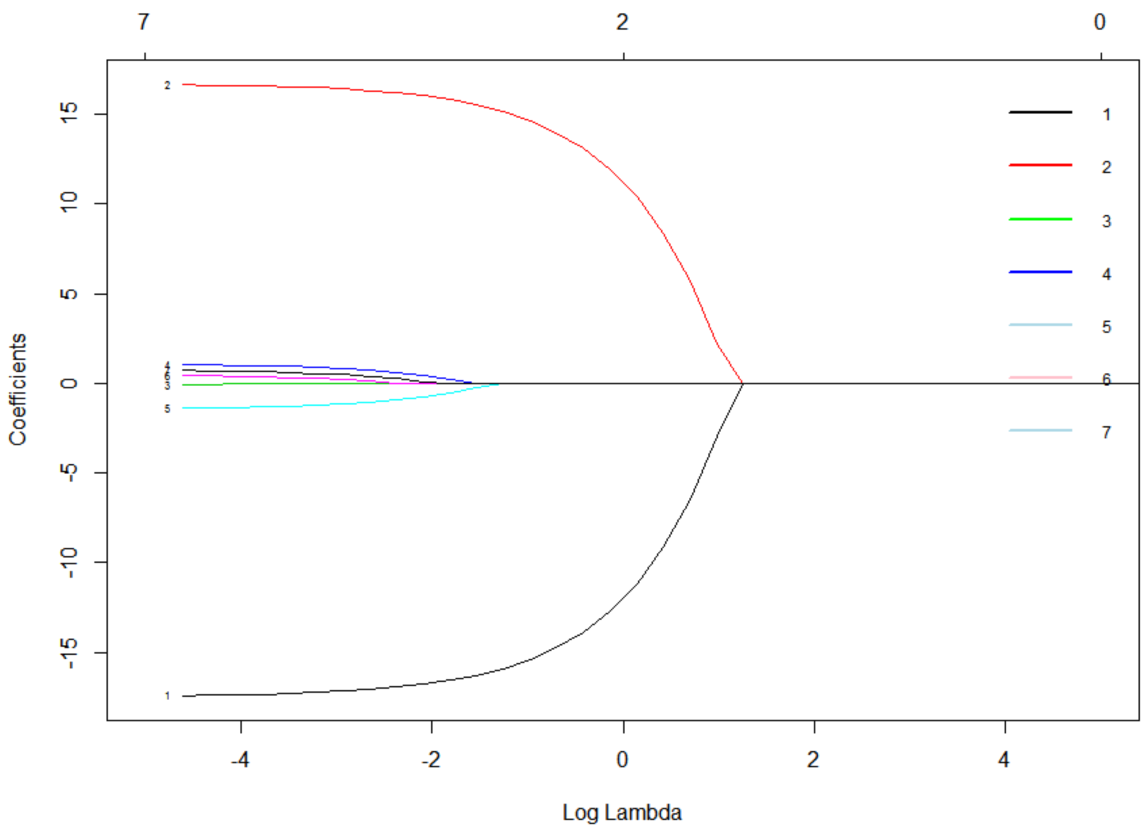
   b. Generated Y as per the equation provided using the 30 values for x and ep.

```
y= 3- 2*x + 3*(x^2) + ep
```

   c. Using the poly function, I have generated xpoly and x1poly later to generate a set of x..x^7 values.

      i. Using the glmnet function and setting alpha=1, we get lasso regression so have used this here. Have stored the value of the lasso regression output in a variable named lasso.mod. Have created a grid of lambda as explained in the textbook. Using grid =10^ seq (10,-2, length =100), I have basically created 100 different values of lambda. Using the plot function, I have plotted the log lambda on the x axis against the coefficients on the y axis. The plot looks like something shown below:

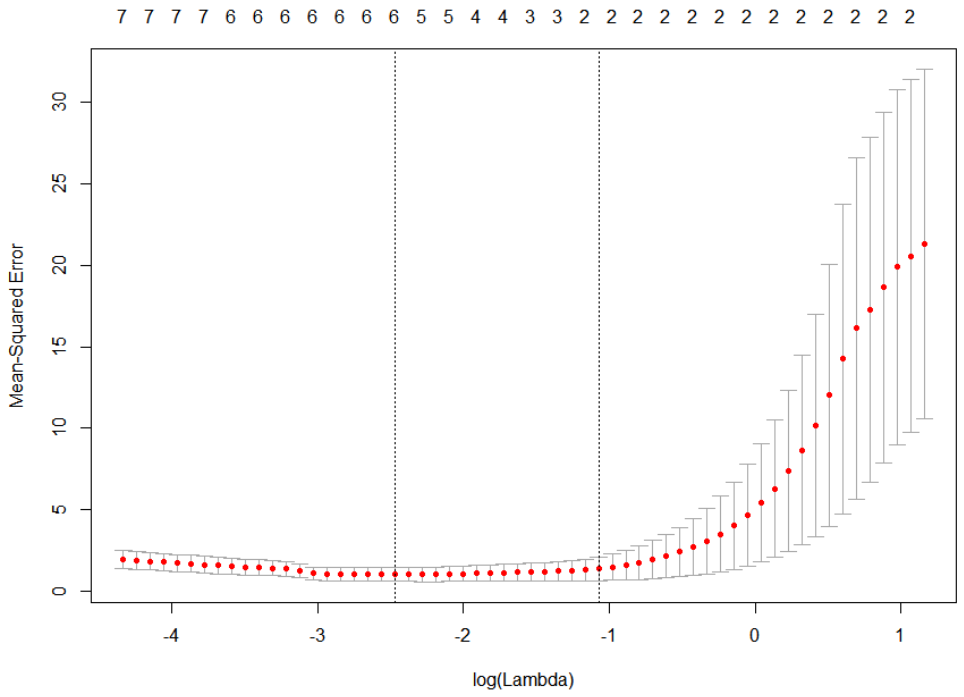
```
grid=10^seq(10,-2,length=100)
xpoly=poly(x,7)
lasso.mod=glmnet(x=xpoly,y=y,alpha=1, lambda=grid)
plot(lasso.mod,xvar="lambda",label=TRUE, xlim=c(-5,5))
legend('topright', c("1","2","3","4","5","6","7") , bty='n', lty=1,col=c('black', 'red', 'green',' blue', 'light blue', 'pink','black'), cex=.75, lwd=c(2.5,2.5))
```

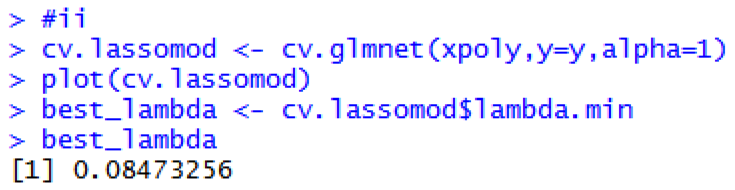


7
2
0
Coefficients
Log Lambda
-4
-2
0
2
4
15
10
5
0
-5
-10
-15
1
2
3
4
5
6
7

ii. Here, I need to use cross validation to select the best tuning parameter. Using the cv.glmnet function, I am able to run cross validation on xpoly and y to get a list of Degrees of freedom, the value of lambda and the % in deviation. Using the .min function, I select the minimum value of lambda which comes out to be 0.08473256 and has been stored in a variable known as best_lambda. The plot on the cross validation produces a log lambda on the x axis and the mean squared error on the y axis and it looks like shown below:

cv.lassomod <- cv.glmnet(xpoly,y=y,alpha=1)
plot(cv.lassomod)
best_lambda <- cv.lassomod$lambda.min
best_lambda

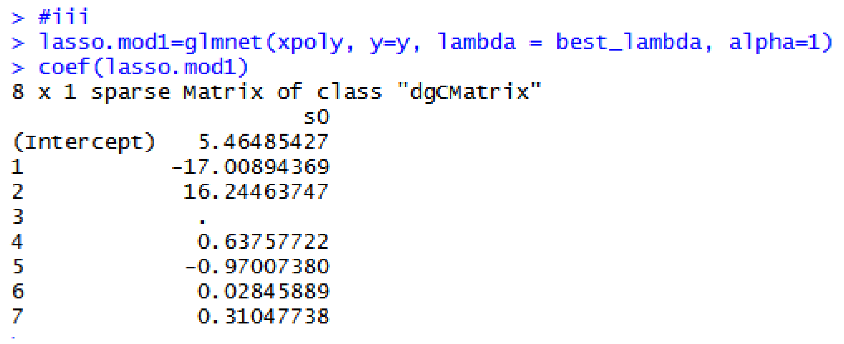




```
> #ii
> cv.lassomod <- cv.glmnet(xpoly,y=y,alpha=1)
> plot(cv.lassomod)
> best_lambda <- cv.lassomod$lambda.min
> best_lambda
[1] 0.08473256
```

iii. Using the value that was derived above for the best_lambda, I have fitted the model. Also, when I generate the output for the coefficients, I can see that the 3rd coefficient has gone down to zero. This happens only in lasso where coefficients actually go down to zero.

lasso.mod1=glmnet(xpoly, y=y, lambda = best_lambda, alpha=1)
coef(lasso.mod1)

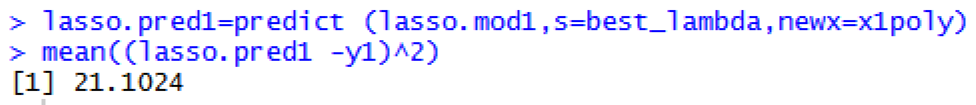


```
> #iii
> lasso.mod1=glmnet(xpoly, y=y, lambda = best_lambda, alpha=1)
> coef(lasso.mod1)
8 x 1 sparse Matrix of class "dgCMatrix"
                     s0
(Intercept)   5.46485427
1           -17.00894369
2            16.24463747
3             .
4             0.63757722
5            -0.97007380
6             0.02845889
7             0.31047738
```

d. Here I have generated 1000 values on the same basis as 1(a) and 1(b). Also have 1000 y values generated. All the variables have been stored as x1, ep1 and y1. Have applied the glmnet function and alpha=1 (for lasso) to predict y1. Here I have used the same tuning parameter which I got above (i.e. best_lambda). I get the mean squared error as below:

x1= rnorm(1000)
ep1=rnorm(1000)
*y1= 3- 2*x1 + 3*(x1^2) + ep1*
x1poly=poly(x1,7)

lasso.pred1=predict (lasso.mod1,s=best_lambda,newx=x1poly)
mean((lasso.pred1 -y1)^2)



```
> lasso.pred1=predict (lasso.mod1,s=best_lambda,newx=x1poly)
> mean((lasso.pred1 -y1)^2)
[1] 21.1024
```

2. **In this exercise, I have applied ridge regression to the data that was generated in 1(a), 1(b), and 1(d). (Make sure you use the same random seed!)**

   a. I have used the same seed value used in the above question. Here the difference would be to use alpha=0 since we are applying ridge regression and the glmnet function needs alpha to be 0 for ridge regression. I have again generated 30 values for x, ep and y as per the required equation. I have also generated 1000 values vector again namely x1, ep1 and generated y1 depending on the same equation. The x1m matrix has also been created from the x1 dataset. I have also used the poly function to create xpoly variable which has values from X..X^7. The grid variable has also been created to store 100 values of lambda to be used ahead.

x= rnorm(30)
ep=rnorm(30)
y= 3- 2*x + 3*(x^2) + ep
xpoly=poly(x,7)

x1= rnorm(1000)
ep1=rnorm(1000)
y1= 3- 2*x1 + 3*(x1^2) + ep1
x1poly=poly(x1,7)

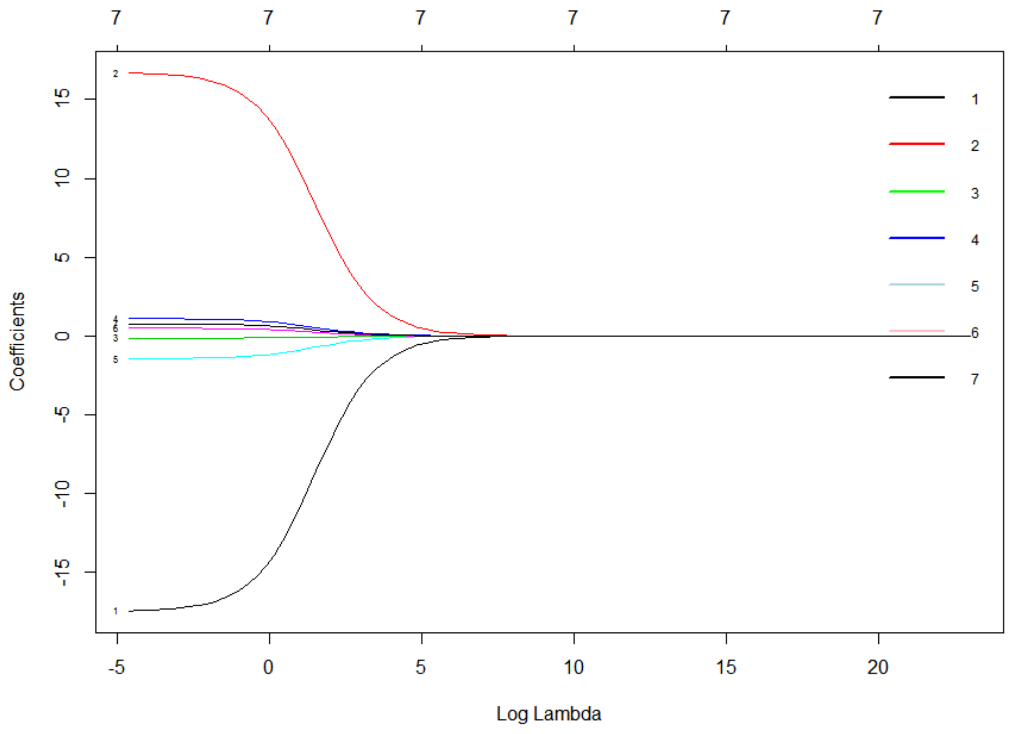
grid =10^ seq (10,-2, length =100)

i. Here I have used the same function glmnet and on the same lines as question one I have performed the regression. The only difference here is that I have used the alpha value as 0 because that applied ridge regression. The coefficients plotted against the log lambda generates the following plot. If we look closely the coefficient tend towards 0 but are not exactly 0 since this is ridge regression wherein the values become 0 only when lambda is infinity.

ridge.mod=glmnet(x=xpoly,y=y,alpha=0, lambda = grid)

plot(ridge.mod,xvar="lambda", label=TRUE)

legend('topright', c("1","2","3","4","5","6","7") , bty='n', lty=1,col=c('black', 'red', 'green',' blue', 'light blue', 'pink','black'), cex=.75, lwd=c(2.5,2.5))



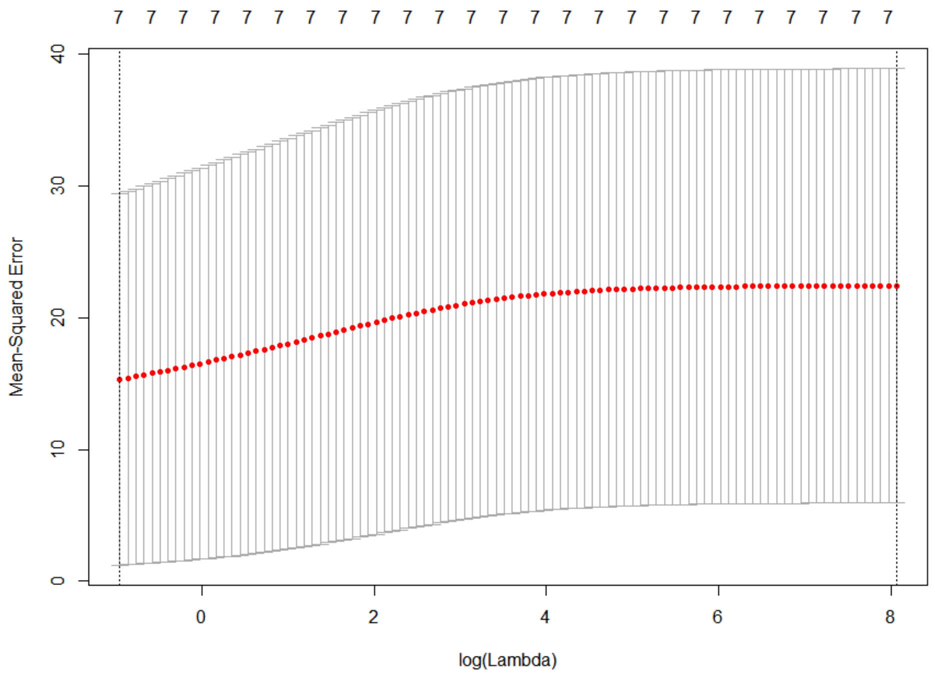
ii. Here again I have used cross validation to check out the best tuning parameter as we did in the above question. The below is the plot of mean squared error plotted against the log lambda and have also generated the best value of lambda by selecting the minimum value.

cv.ridgemod <- cv.glmnet(xpoly,y=y,alpha=0)

plot(cv.ridgemod)

best_lambda <- cv.ridgemod$lambda.min

best_lambda



```
> ridge.pred=predict (ridge.mod ,s=best_lambda,newx=xpoly)
> mean(( ridge.pred -y)^2)
[1] 0.5633316
```

iii. I then fitted this model to the n observations and generated a list of the coefficient values. The mean squared error which is generated by the ridge regression is higher than what was generated by lasso. Also when generated the coefficient values, we can see that the values have the tendency to go close to 0 but are not completely 0 because ridge regression doesn't make the value to be 0 exactly. Here too I have used s=best _lambda which sets my tuning parameter as the one generated in the part ii of this subquestion.

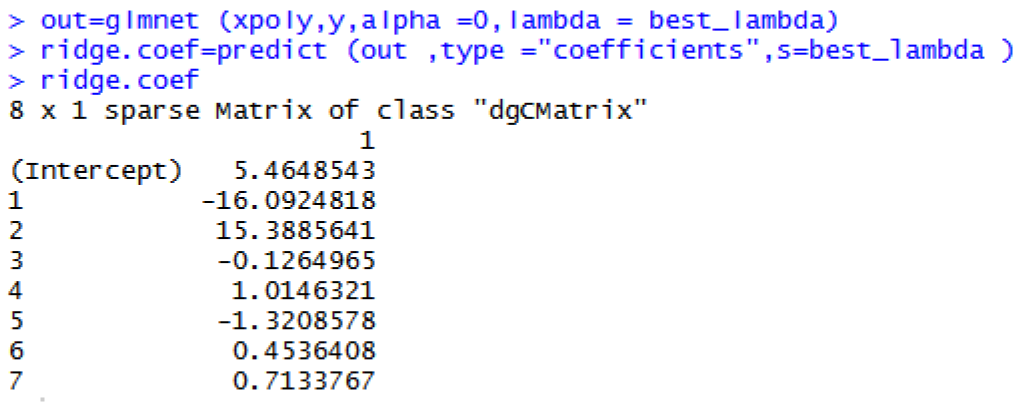
ridge.pred=predict (ridge.mod ,s=best_lambda,newx=xpoly)

mean((ridge.pred -y)^2)

rid.out=glmnet (xpoly,y,alpha =0,lambda = best_lambda)

ridge.coef=predict (rid.out ,type ="coefficients",s=best_lambda)

ridge.coef

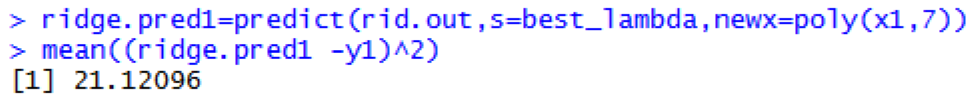


```
> out=glmnet (xpoly,y,alpha =0,lambda = best_lambda)
> ridge.coef=predict (out ,type ="coefficients",s=best_lambda )
> ridge.coef
8 x 1 sparse Matrix of class "dgCMatrix"
                      1
(Intercept)   5.4648543
1           -16.0924818
2            15.3885641
3            -0.1264965
4             1.0146321
5            -1.3208578
6             0.4536408
7             0.7133767
```

b. Here I have fitted the model generated above to the 1000 observation which were generated earlier in this question. The mean squared error is again very high when compared to the a.iii subpart question.

This proves that the tuning parameter generated for one data set cannot be fully applied to another data set and we expect the same result.

```
ridge.pred1=predict(rid.out,s=best_lambda,newx=poly(x1,7))
mean((ridge.pred1 -y1)^2)
```



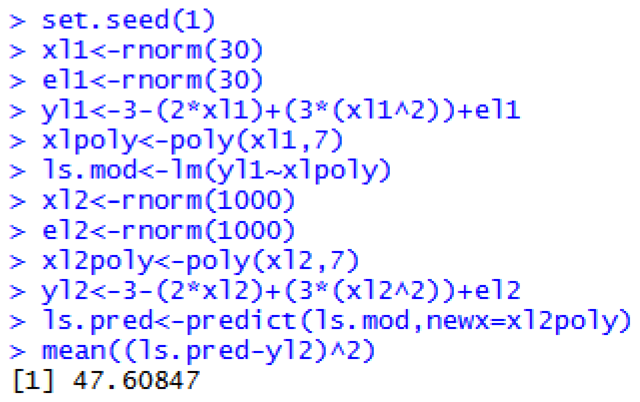
```
> ridge.pred1=predict(rid.out,s=best_lambda,newx=poly(x1,7))
> mean((ridge.pred1 -y1)^2)
[1] 21.12096
```

c. Below is the output of the linear model. When we try to fit a linear model to this data generated above, we get a very high mean squared error. Here I have generated a model using the rnorm for 30 values. And then used this model to predict using 1000 values. The code is as follows:

```
set.seed(1)
xl1<-rnorm(30)
el1<-rnorm(30)
yl1<-3-(2*xl1)+(3*(xl1^2))+el1
xlpoly<-poly(xl1,7)
ls.mod<-lm(yl1~xlpoly)

xl2<-rnorm(1000)
el2<-rnorm(1000)
xl2poly<-poly(xl2,7)
yl2<-3-(2*xl2)+(3*(xl2^2))+el2

ls.pred<-predict(ls.mod,newx=xl2poly)
mean((ls.pred-yl2)^2)
```



```
> set.seed(1)
> xl1<-rnorm(30)
> el1<-rnorm(30)
> yl1<-3-(2*xl1)+(3*(xl1^2))+el1
> xlpoly<-poly(xl1,7)
> ls.mod<-lm(yl1~xlpoly)
> xl2<-rnorm(1000)
> el2<-rnorm(1000)
> xl2poly<-poly(xl2,7)
> yl2<-3-(2*xl2)+(3*(xl2^2))+el2
> ls.pred<-predict(ls.mod,newx=xl2poly)
> mean((ls.pred-yl2)^2)
[1] 47.60847
```

d. Given an option to select between the lasso, ridge and least squares model, I would prefer either the lasso or the ridge over least squares. The mean squared error in my case for the lasso and the ridge is pretty close to each other, however lasso still has a smaller MSE. Lasso is a more interpretable because it makes the coefficient zero. Least squares increases the flexibity so it has low bias and high variance and there is no penalty to compensate for the high variance which ultimately increase the training MSE.