# Math for Data Science: Problem Set 4

Name: Ray Hossain, Group: Monserrat Lopez Perez, Sai Prusni Bandela

2023-27-11

**Due Date:** Friday, December 8 by the end of the day. (The Moodle submission link will become inactive at midnight of December 9.)

**Instructions:** Please submit one solution set per person and include your name and your group members' names at the top. This time, please write your solutions within this Rmd file, under the relevant question. Please submit the knitted output as a pdf. Make sure to show all code you used to arrive at the answer. However, please provide a brief, clear answer to every question rather than making us infer it from your output, and please avoid printing unnecessary output.

## 1. More Cat Compression

(25 points.) I have another cat named Lev. He was upset that Laszlo got to be in my course materials and he didn't. In this exercise we will explain to Lev why my initial choice of Laszlo was nothing personal.

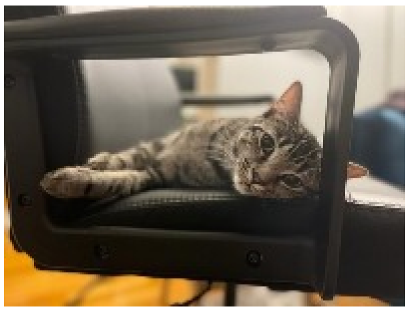
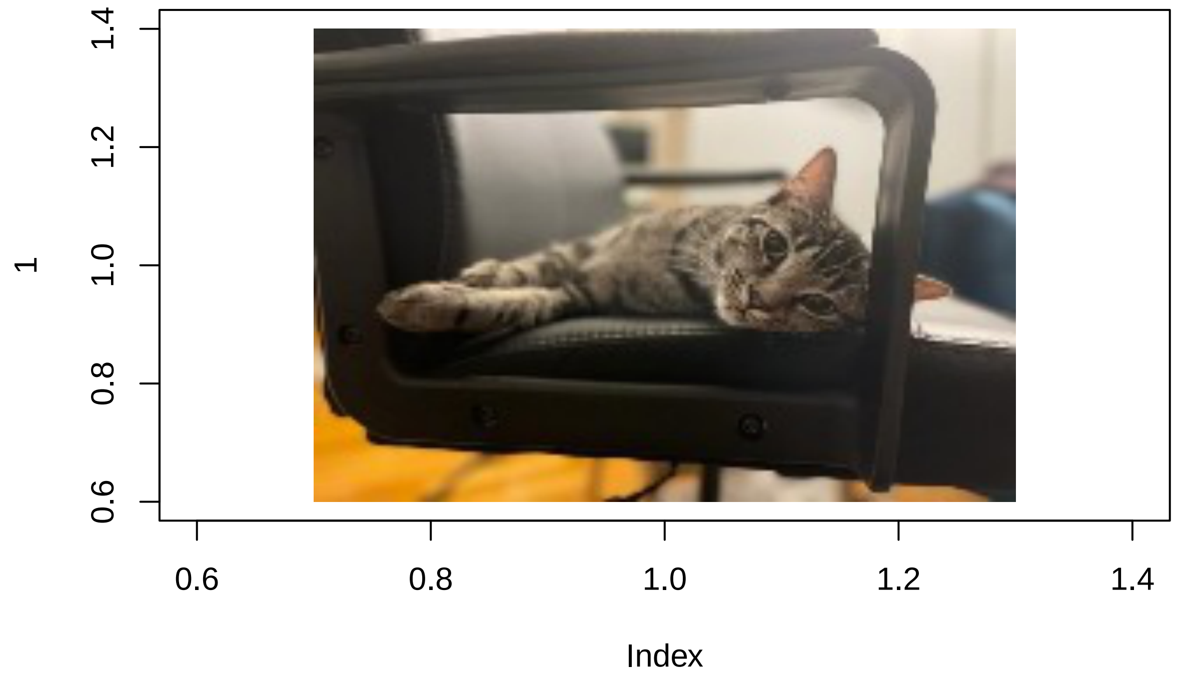


Figure 1: My other cat, Lev.

a. (3 points) Load the image of Lev. Extract its red, green, and blue matrices and store them as separate objects called `r`, `g`, and `b`. Center the blue matrix and compute its variance-covariance matrix. What is the dimensionality of this variance-covariance matrix and why?

```r
# load laszlo and plot
lev <- readJPEG("lev.jpg")
plot(1, type = "n")
rasterImage(lev, 0.7, 0.6, 1.3, 1.4)
```



```r
# store rgb components in separate matrices
r <- lev[,,1]
g <- lev[,,2]
b <- lev[,,3]
```

```r
# center the data
b.centered <- scale(b, center = TRUE, scale = FALSE)

# make variance-covariance matrix
vc <- cov(b.centered)

# check dimensions
dim(vc)
```

```
## [1] 200 200
```

The dimensions of this variance-covariance matrix is 200 by 200. This is because of the shape of the image itself.

b. (3 points) Find the total variance of this image. Compare this to the total variance of the centered blue matrix for Laszlo.

```r
var.total <- sum(apply(b.centered, 2, function(x) sum(x^2))) / nrow(b.centered)
```

```
# load laszlo
laszlo <- readJPEG("laszlo.jpg")

# store blue components in separate matrix
lazlo_b <- laszlo[,,3]
lazlo_b.centered <- scale(lazlo_b, center = TRUE, scale = FALSE)
lazlo_var.total <- sum(apply(lazlo_b.centered, 2, function(x) sum(x^2))) / nrow(lazlo_b.centered)

print(var.total)
```

```
## [1] 9.441113
```

```
print(lazlo_var.total)
```

```
## [1] 12.88591
```

There is a higher total variance of the blue matrix in the Laszlo image than in the Lev image. This likely has to do with the contrast between the two.

c. (3 points) Use the `eigen` command to get the eigendecomposition of the blue variance-covariance matrix for Lev. Multiply the centered blue data matrix by the eigenvector corresponding to the largest eigenvalue to get the first principal component. Compute its variance.

```
eigs <- eigen(vc)

# multiply centered data by first eigenvector
pcb.1 <- b.centered %*% eigs$vectors[,1] #solving for z


var.pcb.1 <- sum(pcb.1^2) / length(pcb.1)
```

d. (3 points) Use the answers to the previous two questions to compute the proportion of variance explained for Lev's first principal component. Check your answer against a scree plot produced by the `fviz_eig` command.[1] Compare to what we saw for Laszlo in the lab.

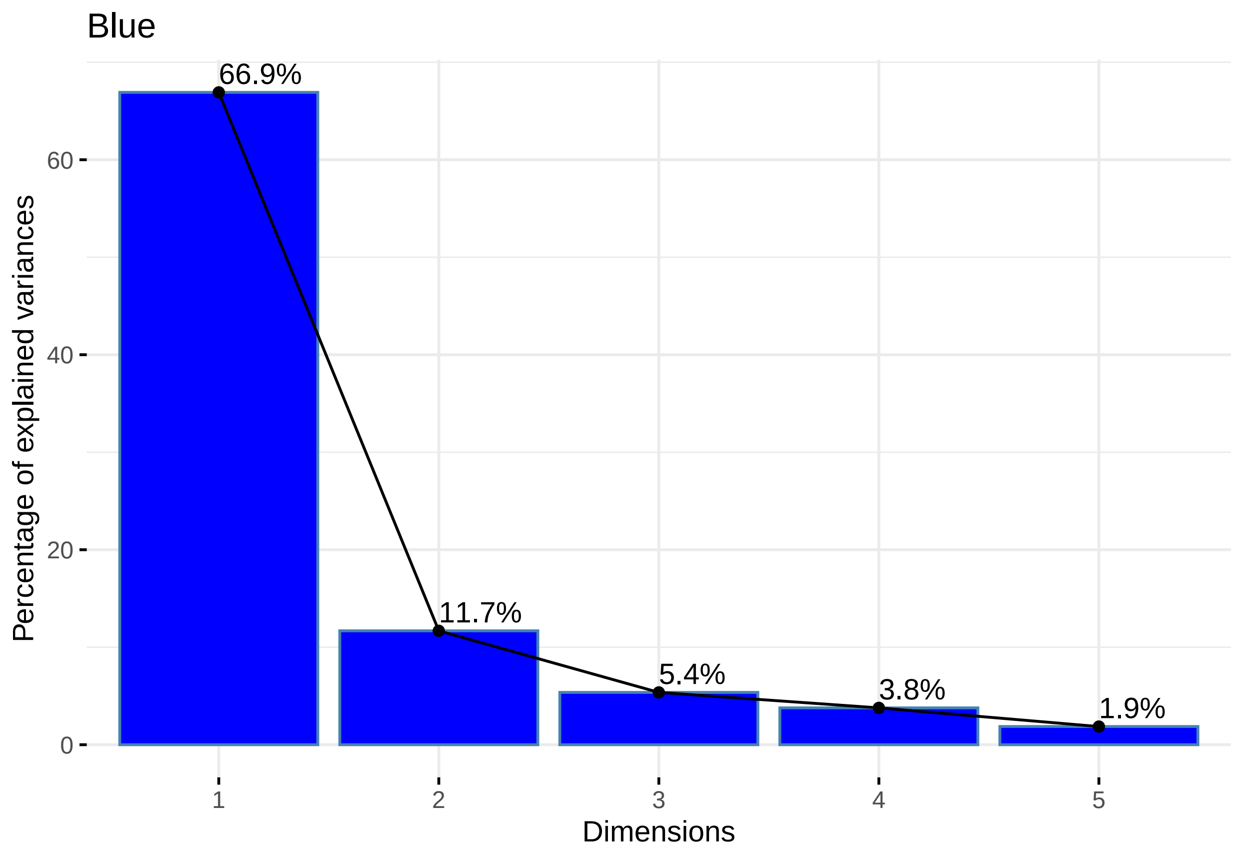
```
var.pcb.1/var.total # data is 66%
```

```
## [1] 0.6690778
```

```
# run pca
b.pca <- prcomp(b, center = TRUE, scale. = FALSE)


# automatically compute the scree plot
fviz_eig(b.pca, main = "Blue", barfill = "blue", ncp = 5, addlabels = TRUE)
```

[1] If they are similar up to the second decimal place, that's good enough.



e. (3 points) Now compute Lev's second principal component. Compute the covariance of the first principal component with the second.[2] Is this what you expected? Comment briefly.

```r
#Take the first PCA and second PCA, then cov() them make sure that it is zero

pcb.2 <- b.centered %*% eigs$vectors[,2]


round(cov(pcb.1, pcb.2), 12)
```

```
##      [,1]
## [1,]    0
```

This output is zero. Which signifies that there is no covariance between the two principal components. This is what we expected because the first principal component is 66.9% of the data. Meanwhile the second principal component does not intersect with the first 66.9%, it is its own seperate 11.7% of the data.

f. (5 points) Now let's run PCA on Lev's `r`, `g`, and `b` matrices using the `prcomp` function with the options `center=FALSE` and `scale.=FALSE`.[3] Combine these objects into a list. Now, looping over a handful of numbers of principal components, reconstitute images of Lev as we did in Lab 9. How many principal components does it take to start to recognize Lev as a cat?

[2]You can round to 10 decimal places.
[3]Scaling and centering is desirable when your variables are on different scales, but in this case it messes up the colors.

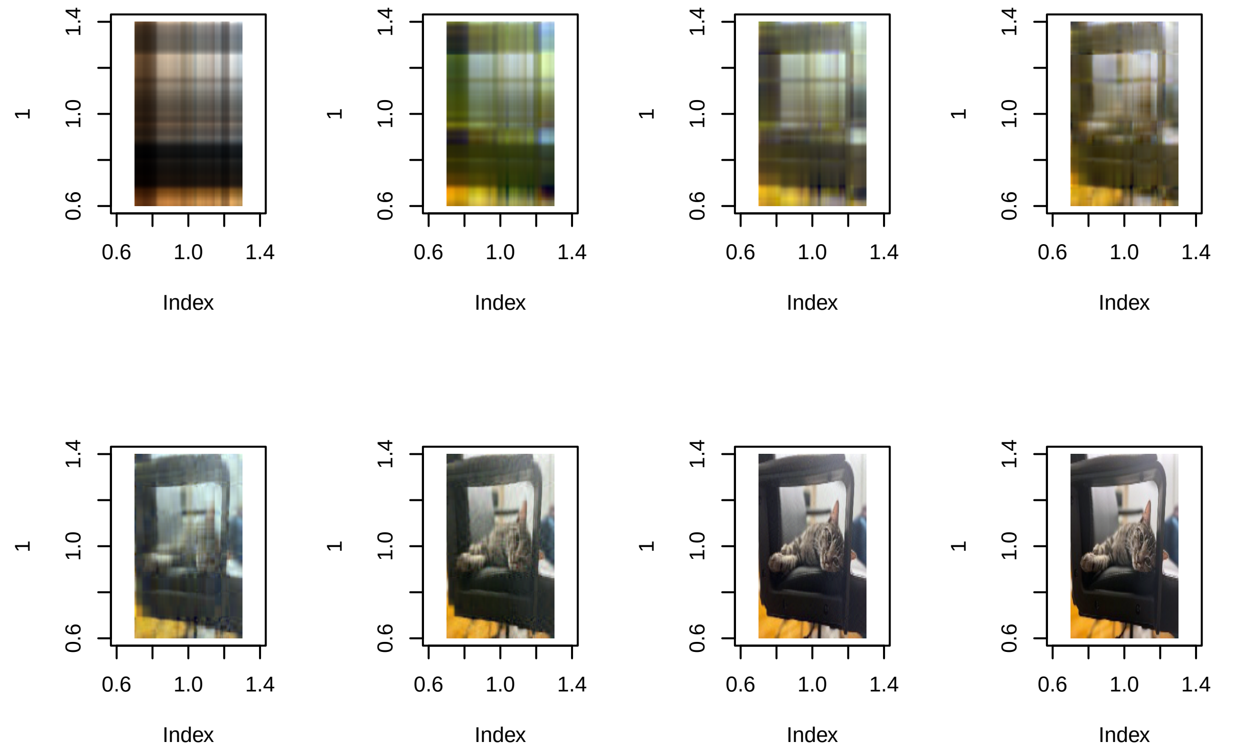
```r
# run pca
r.pca <- prcomp(r, center = FALSE, scale. = FALSE)
g.pca <- prcomp(g, center = FALSE, scale. = FALSE)
b.pca <- prcomp(b, center = FALSE, scale. = FALSE)

# put them together
rgb.pca <- list(r.pca, g.pca, b.pca)


vec <- c(1, 2, 3, 4, 5, 10, 20, 50, 100)

for(i in vec) {
  photo.pca <- sapply(rgb.pca, function(j) {
    # recompose the compressed image
    new.RGB <- j$x[,1:i] %*% t(j$rotation[,1:i])
    # rescale to between 0 and 1
    new.RGB <- (new.RGB - min(new.RGB)) / (max(new.RGB) - min(new.RGB))
  }, simplify = "array")
  assign(paste("photo_", round(i, 0), sep = ""), photo.pca)
}

par(mfrow=c(2,4))
plot(1, type = "n")
rasterImage(photo_1, 0.7, 0.6, 1.3, 1.4)
plot(1, type = "n")
rasterImage(photo_2, 0.7, 0.6, 1.3, 1.4)
plot(1, type = "n")
rasterImage(photo_3, 0.7, 0.6, 1.3, 1.4)
plot(1, type = "n")
rasterImage(photo_5, 0.7, 0.6, 1.3, 1.4)
plot(1, type = "n")
rasterImage(photo_10, 0.7, 0.6, 1.3, 1.4)
plot(1, type = "n")
rasterImage(photo_20, 0.7, 0.6, 1.3, 1.4)
plot(1, type = "n")
rasterImage(photo_50, 0.7, 0.6, 1.3, 1.4)
plot(1, type = "n")
rasterImage(photo_100, 0.7, 0.6, 1.3, 1.4)
```



g. (5 points) Using what you learned from parts (b), (d), and (f) above, explain why Lev requires more principal components to be minimally represented than Laszlo, despite being equally beautiful.

This is because there is more variance in the Lazlo image so its easier to distinguish him from the background. The Lev image has less variance so it requires more principal components to minimally represent him. This has to due with the fact there is more contrast in the Lazlo image, resulting in a higher variance, which results in less principal components needed to represent him.

## 2. Penalized Regression

(25 points.) We will derive the estimator for ridge regression, which is one of several *penalized regression* methods.[4] The process we will follow is very similar to the standard regression estimator we derived in Lab 8, but with a small change to the loss function. Rather than minimizing the sum of squared errors, we will minimize the sum of squared errors subject to a constraint:

$$||\beta||_2^2 \leq s$$

where $s$ is just some constant chosen by the analyst. Recall that $||\beta||_2^2$ is the squared $L_2$ norm of the $\beta$ vector.[5] This question will build on the concepts and data in Lab 8, so please revisit that lab if anything here is unclear.

[4]See p. 237-244 of ISL.
[5]Also known as the Euclidian norm.

a. (2 points) Write down the Lagrangian for this constrained minimization problem. Please use matrix notation (including for the $L_2$ norm) as we did in Lab 8.

$$||\beta||_2^2 \leq s \rightarrow \beta^T\beta - s$$

$$\sum_{i=1}^{n} \varepsilon_i^2 \rightarrow \sum (Y - X\beta)^2 \rightarrow (Y - X\beta)^T(Y - X\beta)$$

$$\mathcal{L} = (Y - X\beta)^T(Y - X\beta) + \lambda(\beta^T\beta - s)$$

b. (7 points) Take the first derivative of the Lagrangian with respect to $\beta$ and set it equal to $\mathbf{0}$ to get the first order condition. Use the Matrix Cookbook to help you. Solve for $\hat{\beta}$.

$$\downarrow expand$$

$$\frac{\partial \mathcal{L}}{\partial \beta} = Y^TY - 2\beta^TX^TY + \beta^TX^TX\beta + \lambda\beta^T\beta - \lambda s = 0$$

$$\downarrow derive$$

$$0 = -2X^TY + 2X^TX\beta + 2\lambda\beta$$

Dividing both sides by 2 and factor out $\beta$, then multiply both side by the inverse to isolate $\hat{\beta}$:

$$X^TY = (X^TX + \lambda)\beta$$

$$\hat{\beta} = (X^TX + \lambda I)^{-1}X^TY$$

c. (3 points) Compare your answer to the standard linear regression estimator. Under what condition are they the same?

In standard linear regression, the estimator is given by:

$$\hat{\beta}_{\text{OLS}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

Comparing this with the ridge regression estimator from the previous response:

$$\hat{\beta}_{\text{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y}$$

They are the same when $\lambda$ is equal to zero.

d. (5 points) Using the same `BostonHousing` dataset and the same set of variables from Lab 8, compute the ridge regression $\hat{\beta}$ with the equation you found in part (b) above. First standardize your $\mathbf{X}$ matrix using the `scale()` function in `R`. Use the `glmnet()` function in the `glmnet` package to check your answer.[6]

```r
data(BostonHousing)


# design matrix
X <- as.matrix(BostonHousing[,c("crim", "chas", "nox", "dis", "ptratio", "rad")],
                              ncol = 6,
                              byrow = TRUE)
X <- apply(X, 2, as.numeric)
X <- cbind(1, X)


# response vector
Y <- as.matrix(BostonHousing$medv)

#The other stuff
I <- diag(ncol(X))
lambda <- 200

beta <- solve(t(X) %*% X + lambda * I) %*% (t(X) %*% Y)

#Doing the same with glmnet

fit <- glmnet(X, Y, alpha = 0, lambda = 200)
beta_glmnet <- coef(fit)[-1]


print(beta)
```

```
##                [,1]
##           1.5337010
## crim     -0.2592343
## chas      1.1700344
## nox       0.5802448
## dis       1.0688842
## ptratio   0.9954687
## rad      -0.1572359
```

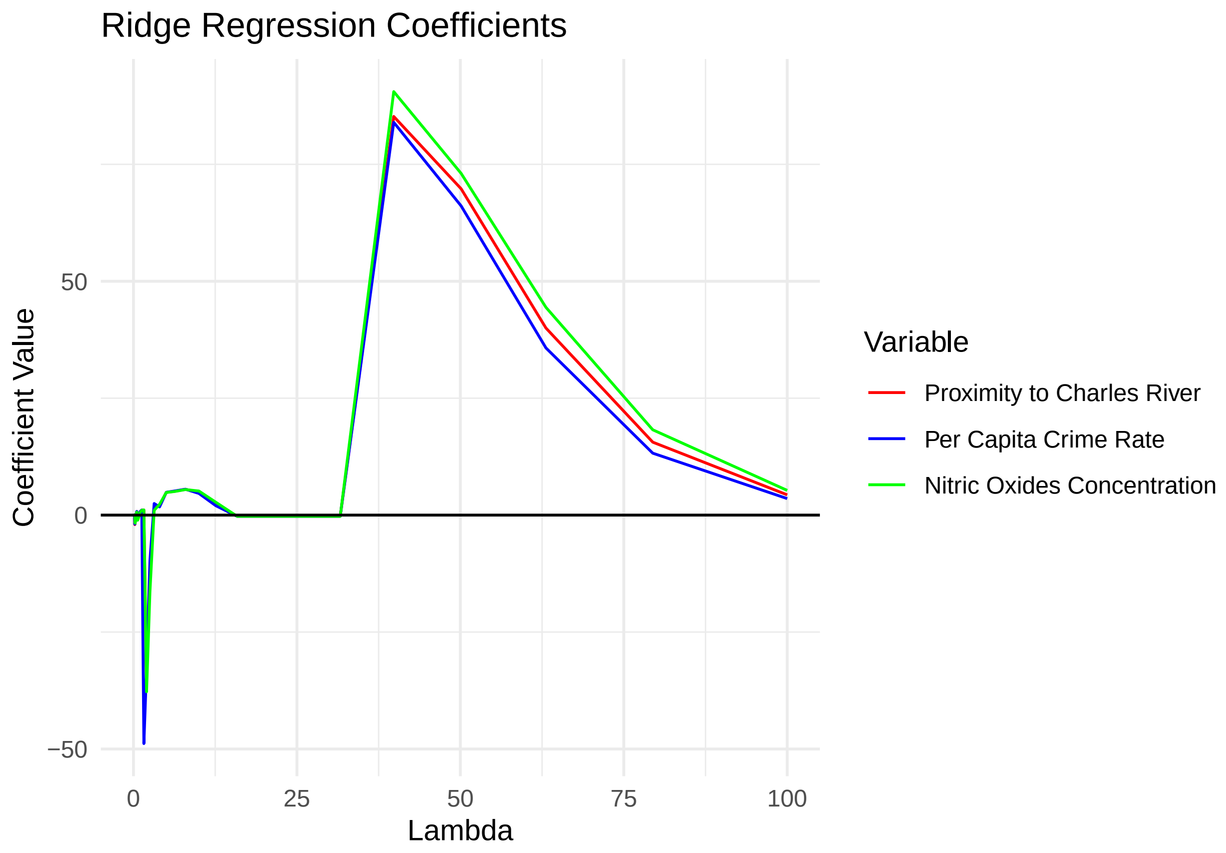
```r
print(beta_glmnet)
```

```
## [1]  0.00000000 -0.01695749  0.27706014 -1.39650877  0.04193844 -0.09136135
## [7] -0.01605890
```

e. (3 points) For the sequence of lambdas below, make a plot with `lambda_seq` on the $x$-axis (increasing from 0 to 100) and the estimated $\beta$ coefficients for per capita crime rate in the town (in blue), proximity

[6] Note that `glmnet`'s `lambda` parameter corresponds to your $\frac{\lambda}{N}$, where $N$ is the number of rows of your data. Also, your estimates may differ slightly from `glmnet`'s, at around the second decimal place. That's alright; this is likely due to `glmnet`'s optimization algorithm. It's computationally expensive to invert large matrices so `glmnet` is probably taking some more efficient but (slightly) less precise approach.

to the Charles River (in red), and nitric oxides concentration (in green) on the $y$-axis. Use `geom_line` to plot the coefficients and add a black horizontal line at $y = 0$. Label your axes and include a legend.



f. (5 points) Discussion:

- Based on your plot above, explain why ridge is one of a number of so-called "shrinkage" estimators.
- Tie this back to the constrained optimization problem you solved to obtain the ridge regression $\hat{\beta}$. Can you see how a larger value of $\lambda$ (or equivalently a small value of $s$) corresponds to greater shrinkage?
- You will learn more about this class of estimators and their virtues next semester, but do you have any intuitions as to when and why they might be desirable?

Shrinkage estimators can help to reduce the variance of the estimates and improve predictive performance. Ridge is a "shrinkage estimator" because it penalizes the s and lambda. The graph above shows that as lambda increases, so does the magnitude of the coefficients from zero but it comes back down towards zero.