Bhuyan_Prashant_is605_FinalExam

Prashant B. Bhuyan

May 25, 2015

Part 1

1. What is the rank of the following matrix?

```
# Rank counts the maximum number of linearly independent rows and
# cols in a matrix (scalar multiples)
# for m rows and n columns the max rank is min(m, n).
# m is a 3x4 matrix.
m <- matrix(c(1,2,6,1,-1,-1,3,5,-2,5,-9,4), nrow = 3, ncol = 4)
z <- qr(m)
rank <- z$rank
# output results - rank is 3
rank
```

```
## [1] 3
```

2. What is the transpose of the above matrix?

```
# the transpose of matrix m above- just exchange rows for columns
transpose_m <- t(m)

# print solution
transpose_m
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    6
## [2,]    1   -1   -1
## [3,]    3    5   -2
## [4,]    5   -9    4
```

3. Define orthonormal basis vectors. Please write down at least one orthonormal basis for the 3-dimensional vector space R3.

Solution:

Vectors are orthonormal if in a subset of vectors in some vector space the inner product of unit vectors $< v_i, v_j >= 0$ where $i \neq j$ such that the unit vectors are perpendicular. Further, such orthonormal vectors are linearly independent and the space they span is called the orthonromal basis.

An example of an orthonormal basis for the 3 dimensional vector space R^3 is:

(0,1,0), (0,0,1), (1,0,1)

4. Given the following matrix, what is its characteristic polynomial?

```
library(matrixcalc)
# square matrix
matrixA = matrix(c(4,1,-1,0,-2,0,5,3,6),3,3)

# print matrix A
matrixA
```

```
##      [,1] [,2] [,3]
## [1,]    4    0    5
## [2,]    1   -2    3
## [3,]   -1    0    6
```

Given the matrix A above, the characteristic polynomial is:

$x^3 - 8x^2 + 9x + 58$

5. What are its (matrix A above) eigenvectors and eigenvalues?

```
# find eigenvalues and eigenvectors
eigen_matrixA <- eigen(matrixA)
matrixA_eigenval <- eigen_matrixA$values
matrixA_eigenvector <- eigen_matrixA$vectors

# print eigenvalues
matrixA_eigenval
```

```
## [1]  5+2i  5-2i -2+0i
```

```
# print eigenvectors
matrixA_eigenvector
```

```
##                 [,1]           [,2] [,3]
## [1,] 0.8855+0.0000i 0.8855+0.0000i 0+0i
## [2,] 0.2272+0.0869i 0.2272-0.0869i 1+0i
## [3,] 0.1771+0.3542i 0.1771-0.3542i 0+0i
```

6. When would a model be said to have a high bias and when would it be said to have a high variance?

Solution:

A model with high bias is a model whose average predictions over many iterations are very far off (large error) from the correct value.

A model with high variance is a model whose predictions for a particular value over many iterations vary drastically from iteration to iteration.

7. Assuming that we are repeatedly sampling sets of numbers (each set is of size n) from an unknown probability density function. What can we say about the average value of each set?

The mean of each sampling set, regardless of the underlying distribution, will exhibit normality- (be distributed normally).

8. What is the derivative of $e^x sin^2(x)$?

Solution: $[d/dx]e^x * sin^2(x) + e^x * [d/dx]sin^2(x)$

$e^x * sin^2(x) + 2 * sin(x) * [d/dx]sin(x) * e^x$

$e^x * sin^2(x) + 2 * cos(x) * e^x * sin(x)$

$e^x * sin^2(x) + 2e^x * cos(x) * sin(x)$

9. What is the derivative of $ecos(2x)$

Solution:

$[d/dx]e * cos(2x)$

$e * [d/dx]cos(2x)$

$e * (-sin(2x)) * [d/dx]2x$

$-e * 2 * [d/dx]x * sin(2x)$

$-2e * 1 * sin(2x)$

$-2e * sin(2x)$

10. What is the integral of $x^2 sin(x)dx$?

Solution:

$2xsin(x) + (2 - x^2) * cos(x) + C$

*** Mini-Coding ***

2.1 Bayes Part A

You are working for a credit card company and you know that about 80% of your customers have a good credit score . People with good credit have a loan default rate of 2%. Whereas people with bad credit default on their loans at the rate of 10%. Given that you are looking at a defaulted loan, what is the probability that it belongs to someone with a bad credit score?

```r
# Solution

# given a defaulted loan, what is the probability that the loan was made to
#someone with bad credit?

# prob of good cs
cust_with_good_cs <- .8

# prob of bad cs
cust_with_bad_cs <- .2

# prob of default given good credit
pr_default_given_good_cs <- .02

# prob of default given bad credit
pr_default_given_bad_cs <- .10

# prob of default
# this says that of all the loans made, 3.6% default.
pr_default <- (cust_with_good_cs*pr_default_given_good_cs)+
  (cust_with_bad_cs*pr_default_given_bad_cs)

# prob of bad credit given prob of default
# this says that 55.55% of all defaulted loans belonged
#to people with bad credit - apply Bayesian Logic
pr_bad_credit_given_default <-
  (pr_default_given_bad_cs*cust_with_bad_cs)/pr_default

# prob of good credit given prob of default
# this says that 44.44% of all defaulted loans belonged
# to people with good credit.
pr_good_credit_given_default <-
  1-pr_bad_credit_given_default

# output results
# results are surprising because it raises the
# question of why there is just a marginal difference
# between the credit profiles of the people that default.
# Are credit scores really the best predictors of loan default???
pr_bad_credit_given_default
```

```
## [1] 0.5556
```

2.1 Bayes Part B

Suppose there are two full bowls of cookies. Bowl 1 has 10 chocolate chip and 30 plain cookies, while bowl 2 has 20 of each. Our friend Fred picks a bowl at random, and then picks a cookie at random. We may assume there is no reason to believe Fred treats one bowl differently from another, likewise for the cookies. The cookie turns out to be a plain one. How probable is it that Fred picked it out of Bowl 1?

```
# Solution:

# probability of picking bowl # 1
pr_bowl_1 <- .5

# number of plain cookies in bowl # 1
num_plain_bowl_1 <- 30

# number of chocolate chip cookies in bowl # 1
num_choc_bowl_1 <- 10

# probability of picking bowl # 2
pr_bowl_2 <- 1 - pr_bowl_1

# number of plain cookies in bowl # 2
num_plain_bowl_2 <- 20

# number of chocolate chip cookies in bowl # 2
num_choc_bowl_2 <- 20

# probability of picking a plain cookie from bowl # 1
pr_plain_cookie_given_bowl_1 <- num_plain_bowl_1/(num_plain_bowl_1+num_choc_bowl_1)

# probability of picking a plain cookie from bowl # 2
pr_plain_cookie_given_bowl_2 <- num_plain_bowl_2/(num_plain_bowl_2+num_choc_bowl_2)

# apply Bayesian logic to figure out pr that a plain cookie came from bowl 1
pr_bowl_1_given_plain_cookie <-
  ((pr_plain_cookie_given_bowl_1)*
     (pr_bowl_1))/(((pr_plain_cookie_given_bowl_1)*
                      (pr_bowl_1))+((pr_plain_cookie_given_bowl_2)*(pr_bowl_2)))

# display results- the results show there is a 60% chance that the plain cookie came
# from bowl 1.
pr_bowl_1_given_plain_cookie
```

```
## [1] 0.6
```

2.2 Central Limit Theorem Part A

Assume a sample of size 100 is selected from a log-normally distribution population with mean 50 and standard deviation 10. What will be the mean of this sample? What will be its standard error?

```
# Solution

# Compute Mean Standard Error of Lognormal Distribution
# The main concept here is that regardless of the
# underlying distribution from which we are sampling
# the distribution of the  mean and stdev tend toward a
# normal distribution over a large number of iterations.
# This is important for statistical inference since non
```

```r
# normal distributions over a large iterations yield
# approximately normal sampling means from which probabilities
# can be inferred.
lapply(1, function(k){
  dist <- rlnorm(100, meanlog = log10(50), sdlog = log10(10))
  c(mean = mean(dist), stderr = sd(dist)/sqrt(length(dist)))

})
```

```
## [[1]]
##   mean stderr
## 12.006  1.935
```

```r
# for large n the sample mean of the lognormal distribution
# should be normally distributed. As such probabilities
# about the first lognormal distribution can be statistically
# inferred.
```

2.2 Central Limit Theorem Part B

A random sample of 100 observations is to be drawn from a population with a mean of 40 and a standard deviation of 25. The standard deviation of the mean's distribution will be?

```r
# Solution
# ***Note***: We don't know what kind of distribution
# this is-it's not specified- I'm assuming it's a normal
# distribution. the standard deviation of the mean is
# just the standard error of the mean.
dis_n = 100
dis <- rnorm(dis_n, mean = 40, sd = 25)

# compute stdev of the mean's distribution - over
# large number of trials, the mean should be approximately
# distributed.
stdev_of_mean <- sd(dis)/sqrt(length(dis_n))

# output results- the standard deviation of the mean or
# std error shows how much the mean varies over different
# trials. The standard error of the mean decreases as number
# of trials increases.
stdev_of_mean
```

```
## [1] 26.46
```

2.2 Central Limit Theorem Part C

A random sample of 100 observations is to be drawn from a population with a mean of 40 and a standard deviation of 25. The probability that the mean of the sample will exceed 45 is approximately? (please be precise to within 2 decimal places).

```r
# Solution.
# assuming normal distribution
q_size <- 100
q_dist <- rnorm(q_size, mean = 40, sd = 25)
qthresh <- 45
q_pop_mean <- mean(q_dist)
q_pop_sd <- sd(q_dist)

# compute z score- this standardizes measurements in terms of
# distance of a value from the mean in standard deviations.
z_score = (qthresh-q_pop_mean)/(q_pop_sd*sqrt(length(q_size)))
z_score
```

```
## [1] -0.0173
```

```r
# output result using pnorm - probability normal
# distribution function
pr_mean_exceeds_qthresh <- pnorm(z_score)

# this gives the probability that mean of the
# sample is 45.
pr_mean_is_qthresh <- pnorm(z_score)
pr_mean_is_qthresh
```

```
## [1] 0.4931
```

```r
# this gives the probability that the mean of the
# sample exceeds 45.
pr_mean_exceeds_qthresh <- 1-pr_mean_is_qthresh
pr_mean_exceeds_qthresh
```

```
## [1] 0.5069
```

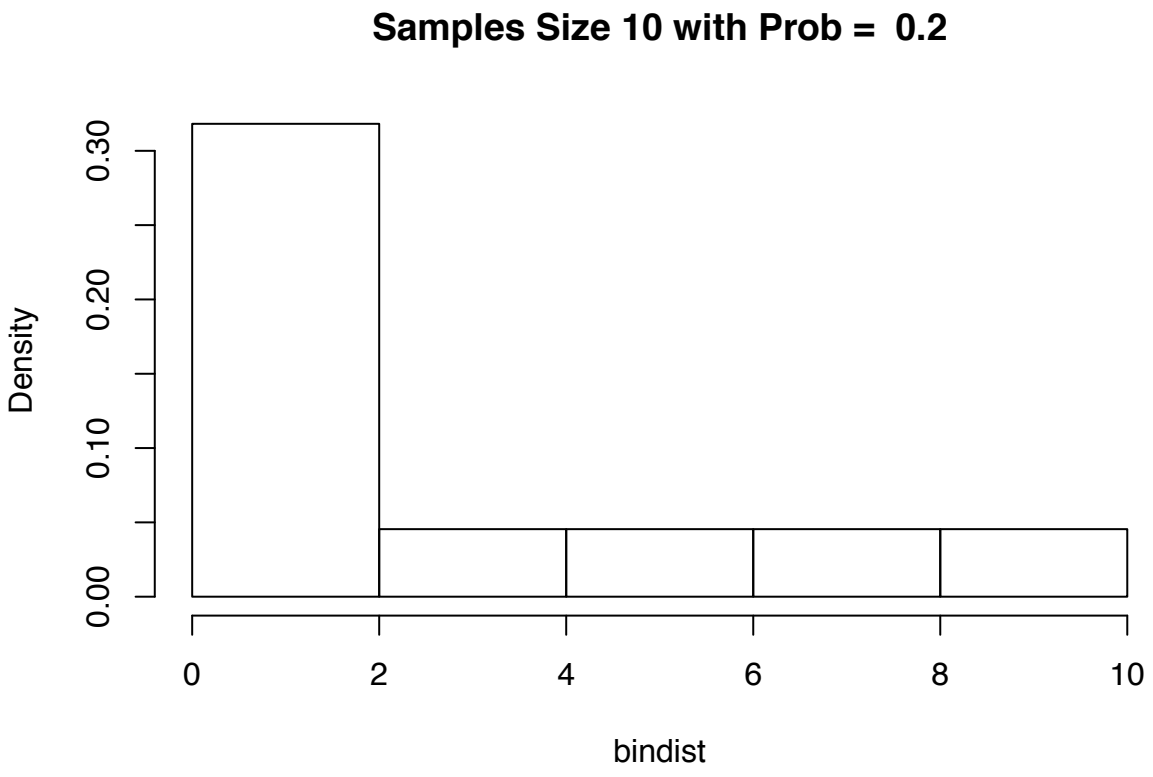
2.3 Sampling from Function.

Assume that you have a function that generates integers between 0 and 10 with a binomial probability distribution where p = .2 and . Write a function to sample from this distribution. After that, generate 1000 samples from this distribution and plot the histogram of the sample. Please note that the Binomial distribution is a discrete distribution and takes values only at integer values of x between x = [0, 10]. Sampling from a discrete distribution with finite values is very simple but it is not the same as sampling from a continuous distribution.

```r
# Solution

# function generates 1000 samples from a binomial
# distribution of values between 0 and
# 10 and plots the histogram of the sample
# P(x<=10) - P(X<=0)
f <- function(n, p){
```

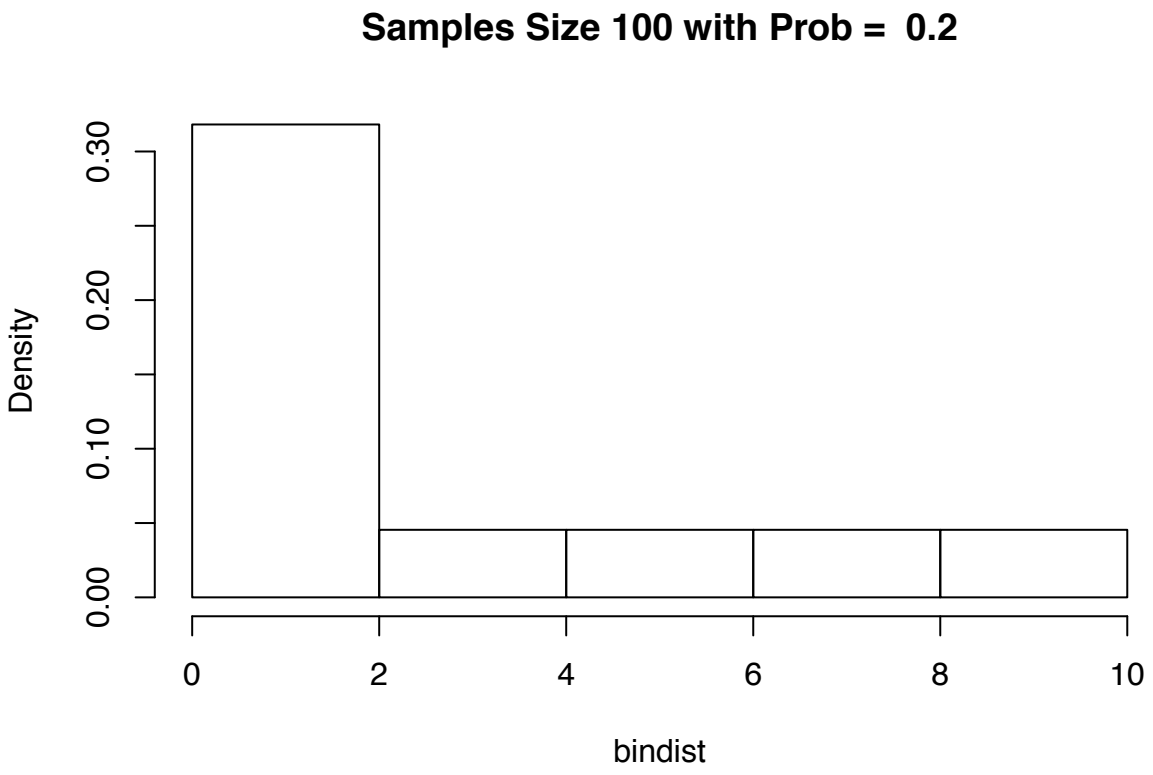
```r
  bindist <- cbind(0:10, round(dbinom(0:10, n, p), 5))

  # note values are rounded out to 5 decimals
  print(bindist)
  hist(bindist, xlim=range(0,10), prob = T,
       main = paste("Samples Size", n, "with Prob = ", p))

  }

# 10 trials
f(10, .2)
```

```
##       [,1]    [,2]
##  [1,]    0 0.10737
##  [2,]    1 0.26844
##  [3,]    2 0.30199
##  [4,]    3 0.20133
##  [5,]    4 0.08808
##  [6,]    5 0.02642
##  [7,]    6 0.00551
##  [8,]    7 0.00079
##  [9,]    8 0.00007
## [10,]    9 0.00000
## [11,]   10 0.00000
```



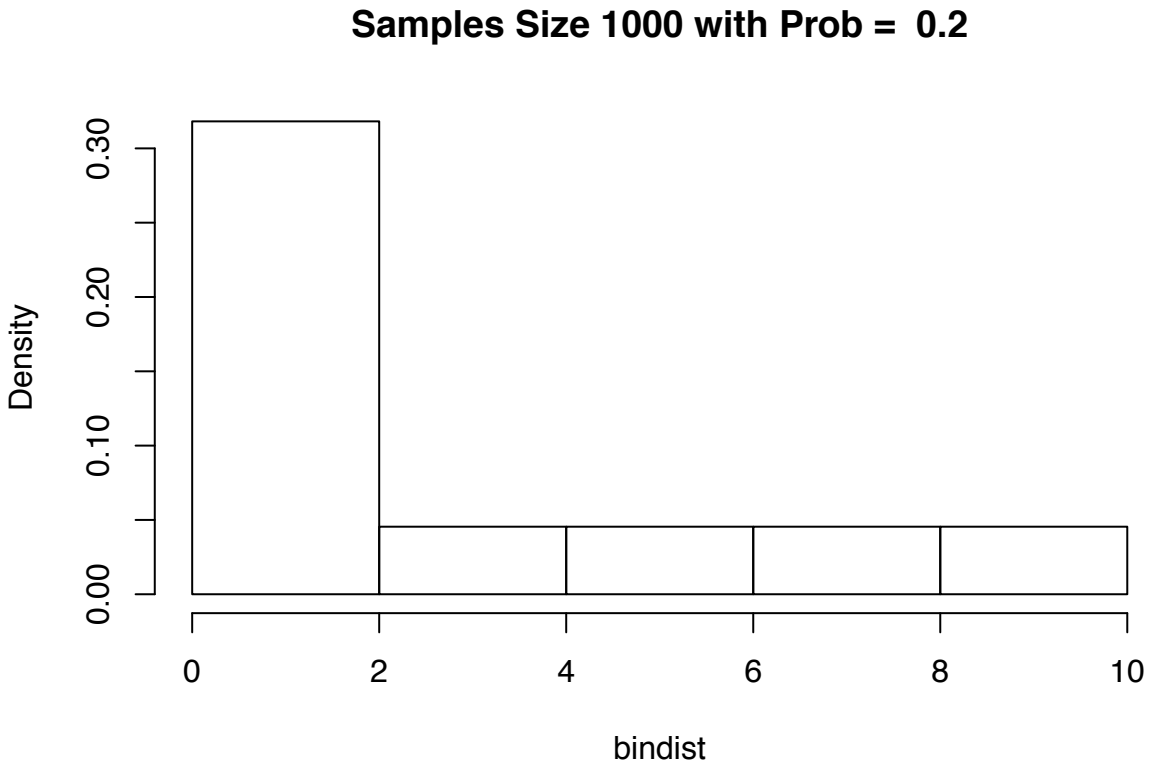

```r
# 100 trials
f(100, .2)
```

```
##       [,1]    [,2]
##  [1,]    0 0.00000
##  [2,]    1 0.00000
##  [3,]    2 0.00000
##  [4,]    3 0.00000
##  [5,]    4 0.00000
##  [6,]    5 0.00001
##  [7,]    6 0.00006
##  [8,]    7 0.00020
##  [9,]    8 0.00058
## [10,]    9 0.00148
## [11,]   10 0.00336
```




```
# 1000 trials
f(1000, .2)
```

```
##       [,1] [,2]
##  [1,]    0    0
##  [2,]    1    0
##  [3,]    2    0
##  [4,]    3    0
##  [5,]    4    0
##  [6,]    5    0
##  [7,]    6    0
##  [8,]    7    0
##  [9,]    8    0
## [10,]    9    0
## [11,]   10    0
```




```
# The intuition here is that of n trials the
# integers between 0 and 10 that are distributed
# with a probability rate of 20% fall overwhemingly
# between 0 and 2.
```

2.4. Principal Components Analysis.

For the auto data set attached with the final exam, please perform a Principal Components Analysis by performing an SVD on the 4 independent variables (with mpg as the dependent variable) and select the top 2 directions. Please scatter plot the data set after it has been projected to these two dimensions. Your code should print out the two orthogonal vectors and also perform the scatter plot of the data after it has been projected to these two dimensions.

```
# read in auto data into a data frame
auto_data <- read.csv("~/Desktop/auto.csv")

# name columns
colnames(auto_data) <- cbind("Displacement", "Horsepower", "Weight", "Acceleration", "MPG")

# check headers
head(auto_data)
```

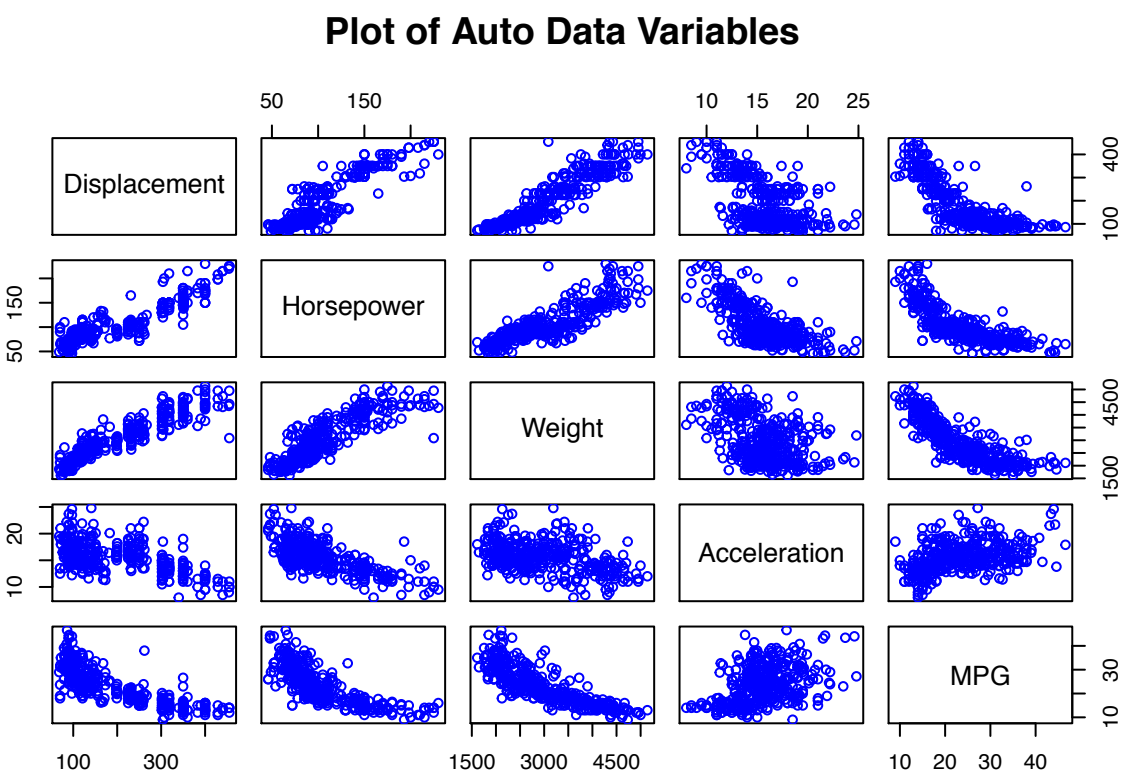
```
##   Displacement Horsepower Weight Acceleration MPG
## 1          350        165   3693         11.5  15
## 2          318        150   3436         11.0  18
## 3          304        150   3433         12.0  16
## 4          302        140   3449         10.5  17
```

```
## 5          429        198   4341         10.0  15
## 6          454        220   4354          9.0  14
```

```
# create df of 4 independent vars
displacement <- auto_data["Displacement"]
horsepower <- auto_data["Horsepower"]
weight <- auto_data["Weight"]
acceleration <- auto_data["Acceleration"]

indep_df <- data.frame(displacement, horsepower, weight, acceleration)

# dependent variable
mpg <- auto_data["MPG"]
head(mpg)
```

```
##   MPG
## 1  15
## 2  18
## 3  16
## 4  17
## 5  15
## 6  14
```

```
# plot the data
plot(auto_data, cex = 0.9, col = "blue", main = "Plot of Auto Data Variables")
```



```r
# in the plots above we see some relationships between variables that seem positively
#linearly dependent and others that seem negatively linearly dependent and still others
#where there is no discernable pattern.
# however we want to figure out which variables have the most variance to get into the
#structure of the data.  We can do this with PCA.

# scale and standardize the data
standardize <- function(x) {
  (x - mean(x))

}

scaled_auto_data <- apply(auto_data, 2, function(x) (x-mean(x)))

# standardize the indep vars
scaled_indep_df <- apply(indep_df, 2, function(x) (x-mean(x)))

# standardize the dependent mpg var
scaled_mpg <- apply(mpg, 2, function(x) (x-mean(x)))

# find eigen values of covariance matrix
cov = cov(scaled_auto_data)

# find eigen values of covariance matrix
eigen = eigen(cov)

# name the rows
rownames(eigen$vectors) = c("Displacement", "Horsepower", "Weight", "Acceleration", "MPG")

# name the columns
colnames(eigen$vectors) = c("PC1", "PC2", "PC3", "PC4", "PC5")

# output results - each PC is a linear combination of variables and the coefficients
#are loadings. Each PC is just an eigen vector.
eigen
```

```
## $values
## [1] 7.333e+05 1.511e+03 2.615e+02 1.786e+01 2.901e+00
##
## $vectors
##                     PC1      PC2       PC3       PC4       PC5
## Displacement -0.114242  0.94566 -0.304224 -0.004927 -0.009604
## Horsepower   -0.038957  0.29937  0.948590 -0.043621 -0.084452
## Weight       -0.992659 -0.12076 -0.002631 -0.005350  0.003025
## Acceleration  0.001348 -0.03476 -0.077255  0.009240 -0.996361
## MPG           0.007593 -0.01741 -0.040621 -0.998979 -0.005497
```

```r
sum(eigen$values)
```

```
## [1] 735120
```

```r
var(scaled_auto_data[,1]) + var(scaled_auto_data[,2])
```

```
## [1] 12430
```

```r
# loadings
loadings = eigen$vectors
loadings
```

```
##                     PC1      PC2       PC3       PC4       PC5
## Displacement -0.114242  0.94566 -0.304224 -0.004927 -0.009604
## Horsepower   -0.038957  0.29937  0.948590 -0.043621 -0.084452
## Weight       -0.992659 -0.12076 -0.002631 -0.005350  0.003025
## Acceleration  0.001348 -0.03476 -0.077255  0.009240 -0.996361
## MPG           0.007593 -0.01741 -0.040621 -0.998979 -0.005497
```

```r
# examine relationship between scaled mpg data and eigen vectors

# PC1 - MPG vs Independent Vars
slope_pc1 = eigen$vectors[1,1]/eigen$vectors[2,1]
slope_pc1
```

```
## [1] 2.932
```

```r
slope_pc2 = eigen$vectors[1,2]/eigen$vectors[2, 2]
slope_pc2
```

```
## [1] 3.159
```

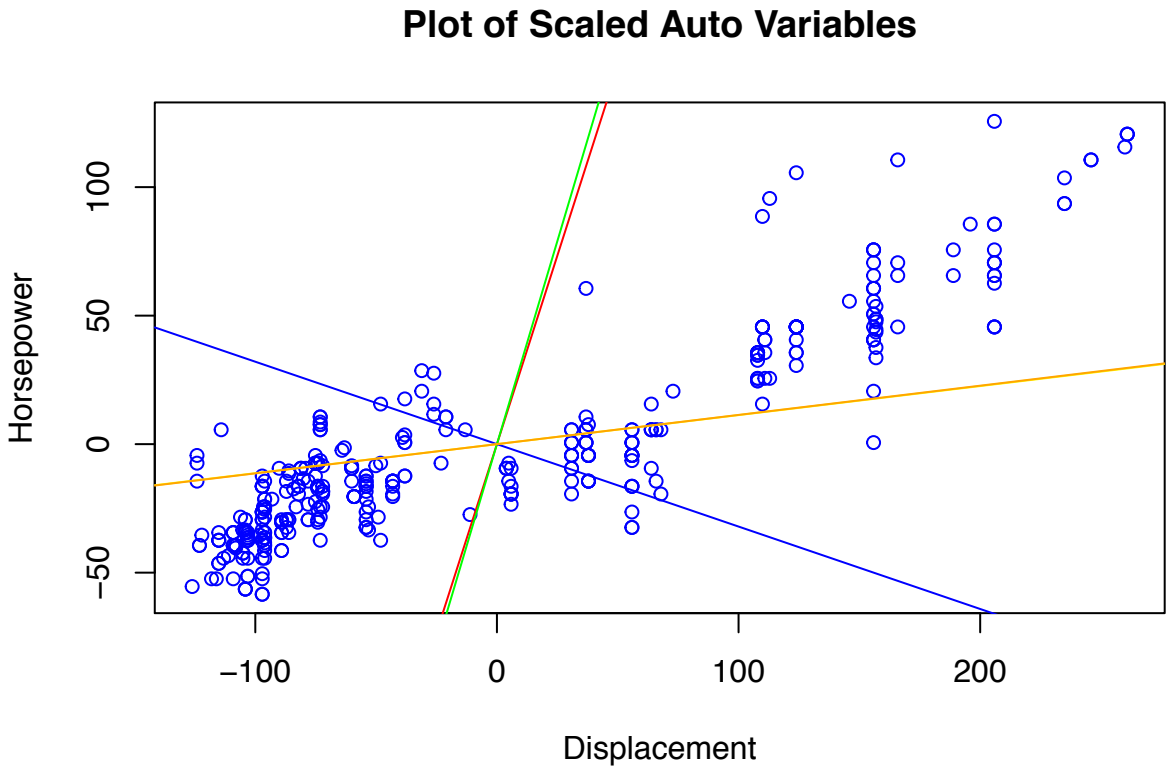
```r
slope_pc3 = eigen$vectors[1,3]/eigen$vectors[2, 3]
slope_pc3
```

```
## [1] -0.3207
```

```r
slope_pc4 = eigen$vectors[1,4]/eigen$vectors[2, 4]
slope_pc4
```

```
## [1] 0.1129
```

```r
slope_pc5 = eigen$vectors[1,5]/eigen$vectors[2,5]

# plot scaled auto data
plot(scaled_auto_data, cex = 0.9, col = "blue", main = "Plot of Scaled Auto Variables")


# plot slopes of eigenvector/principal components and compare to the scaled data
# here we say that pc5 captures more variation than the otehr ocmon
abline(0, slope_pc1, col = "red")
abline(0, slope_pc2, col = "green")
abline(0, slope_pc3, col = "blue")
abline(0, slope_pc4, col = "yellow")
abline(0, slope_pc5, col = "orange")
```




```
# compute % variation for each principal component
var_pc1 = 100*round(eigen$values[1]/sum(eigen$values), digits = 5)

var_pc2 = 100*round(eigen$values[2]/sum(eigen$values), digits = 5)

var_pc3 = 100*round(eigen$values[3]/sum(eigen$values), digits = 5)

var_pc4 = 100*round(eigen$values[4]/sum(eigen$values),digits = 5)

var_pc5 = 100*round(eigen$values[5]/sum(eigen$values), digits = 5)

# output variation by principal component
# here we see that principal component 1 accounts for most #variation amongst the
#scaled variables
xlab = paste("PC1 - ", var_pc1," % of variation ", sep = "")
xlab
```

```
## [1] "PC1 - 99.756 % of variation "
```

```
xlab2 = paste("PC2 - ", var_pc2," % of variation ", sep = "")
xlab2
```

```
## [1] "PC2 - 0.205 % of variation "
```

```
xlab3 = paste("PC3 - ", var_pc3," % of variation ", sep = "")
xlab3
```

```
## [1] "PC3 - 0.036 % of variation "
```

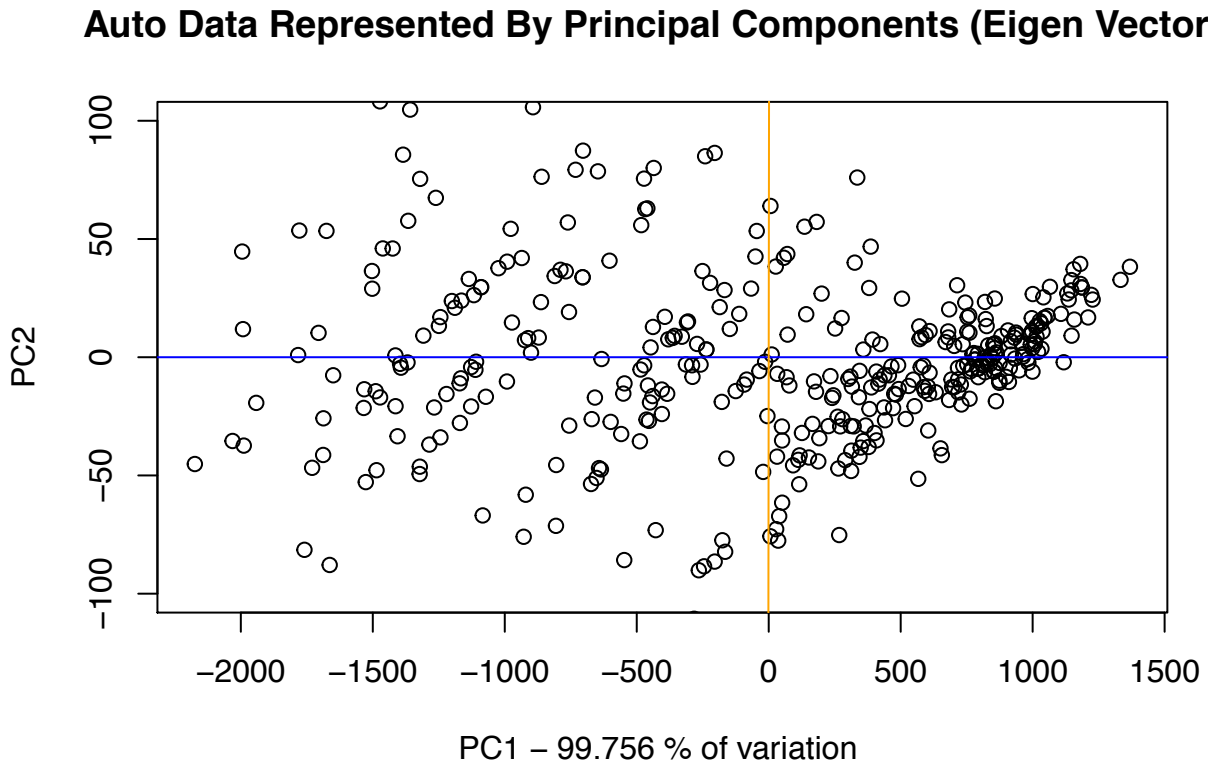
```
xlab4 = paste("PC4 - ", var_pc4," % of variation ", sep = "")
xlab4
```

```
## [1] "PC4 - 0.002 % of variation "
```

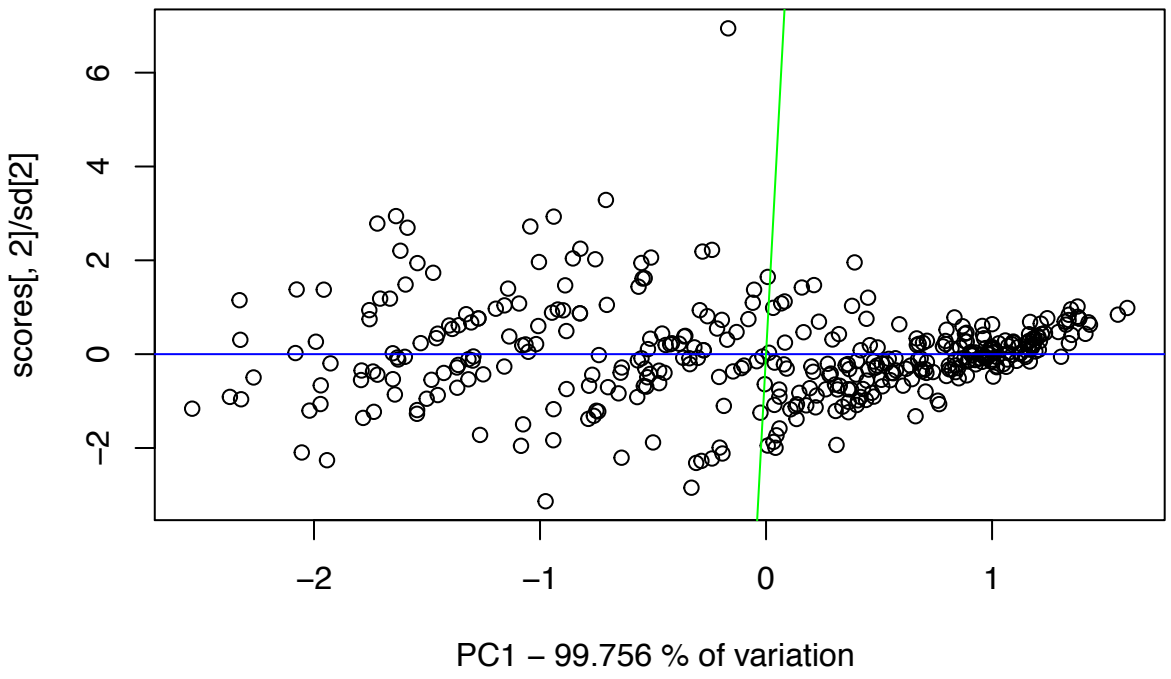
```
xlab5 = paste("PC5 - ", var_pc5," % of variation ", sep = "")
xlab5
```

```
## [1] "PC5 - 0 % of variation "
```

```
# score the dat- multiply scaled data by eigen vectors (principal components)
sd = sqrt(eigen$values)
rownames(loadings) = colnames(auto_data)
scores = scaled_auto_data %*% loadings
plot(scores,
     ylim = c(-100,100),
     main = "Auto Data Represented By Principal Components (Eigen Vectors)", xlab = xlab)
abline(0,0,col = "blue")
abline(0,90,col = "orange")
```




```
# Correlation BiPlot
scores.min = min(scores[,1:2])
scores.max = max(scores[,1:2])
plot(scores[,1]/sd[1],scores[,2]/sd[2], main = "Auto Data Bi Plot", xlab = xlab)
rownames(scores)=seq(1:nrow(scores))
abline(0,0,col = "blue") # PC1
abline(0,90,col = "green") # PC2
```




```
# here we can see that correlation confirms our intution based on the first principal
#component of the relationship between the variables.

# ANALYSIS
# According to the PC1, the eigen vector that accounts for 99.756% of the variance in
#the data, better MPG and Acceleration is associated with lower Weight and Displacement
#primarily.  Interestingly, Weight is a bigger factor on MPG than Horsepower (bigger
#engines) according to PCs.

# The correlation matrix of the auto data below confirms these readings.  MPG is
#negatively correlated to Weight, Displacement and Horespower and positively correlated
#to Acceleration.  Lighter cars accelerate faster and have better MPG.
cor(scaled_auto_data)
```

```
##              Displacement Horsepower  Weight Acceleration     MPG
## Displacement       1.0000     0.8973  0.9331      -0.5422 -0.8049
## Horsepower         0.8973     1.0000  0.8644      -0.6889 -0.7782
## Weight             0.9331     0.8644  1.0000      -0.4159 -0.8321
## Acceleration      -0.5422    -0.6889 -0.4159       1.0000  0.4222
## MPG               -0.8049    -0.7782 -0.8321       0.4222  1.0000
```

```
# taking some random samples of the original data set we can see this behavior above.
auto_data[50,] # MPG of 30, Weight of 2074
```

```
##    Displacement Horsepower Weight Acceleration MPG
## 50           79         70   2074         19.5  30
```

```r
auto_data[331,] # MPG of 23.7, Weight of 2420
```

```
##     Displacement Horsepower Weight Acceleration  MPG
## 331           70        100   2420         12.5 23.7
```

```r
auto_data[75,] # MPG of 18, Weight of 2933
```

```
##    Displacement Horsepower Weight Acceleration MPG
## 75          121        112   2933         14.5  18
```

2.5. Sampling in Bootstrapping.

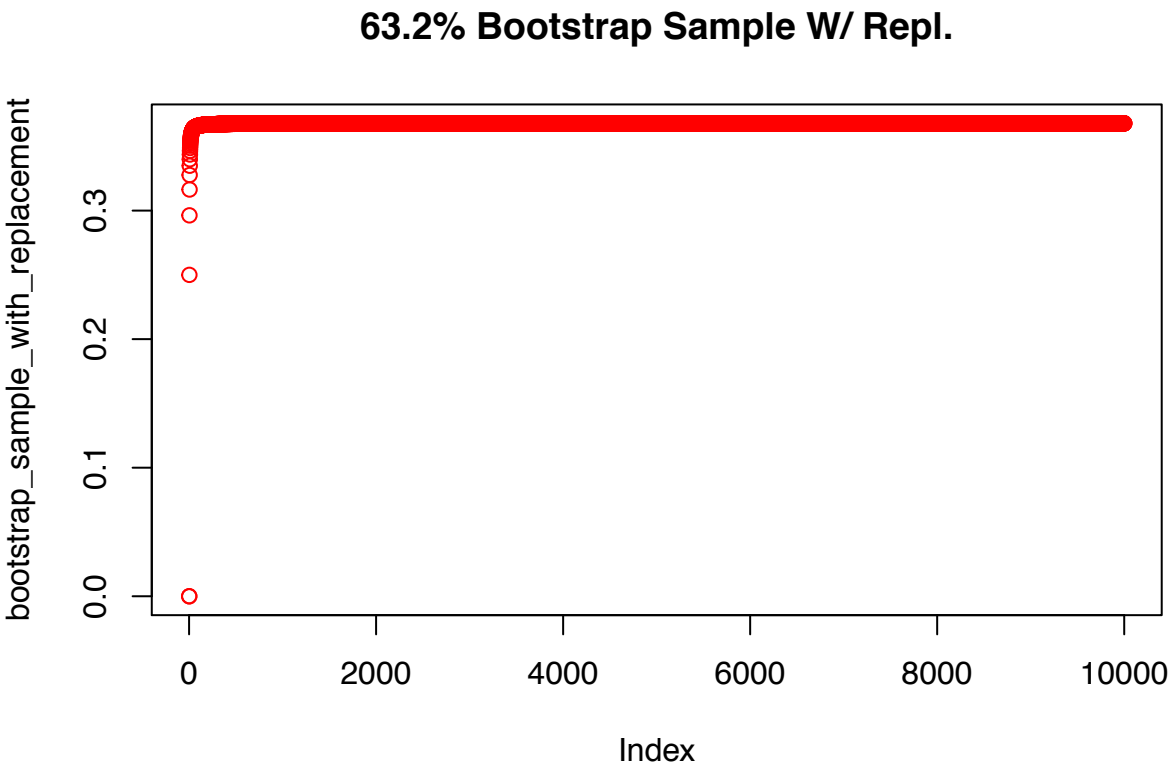
As we discussed in class, in bootstrapping we start with n data points and repeatedly sample many times with replacement. Each time, we generate a candidate data set of size n from the original data set. All parameter estimations are performed on these candidate data sets. It can be easily shown that any particular data set generated by sampling n points from an original set of size n covers roughly .632 of the original data set. Using probability theory and limits, please show that this is true.

Solution:

Bootstrap Sampling is an inferior method to K-Fold Cross Validation for estimating test prediction error because of the .632 bias intrinsic to the bootstrap sampling method.

Prove: $1 - 1/n^n = e^{-1}$ as n approaches infinity.

```r
# plot bootstrap with replacement distribution

boot <- function(n){
  bootstrap_sample_with_replacement <- 0
  for(i in 1:n){
    d <- (1-1/i)^i
    # print(d)

    bootstrap_sample_with_replacement <- c(bootstrap_sample_with_replacement, d)


  }
  plot(bootstrap_sample_with_replacement, main = "63.2% Bootstrap Sample W/ Repl.", col = "red")
}

boot(10000)
```



We can see as the sample grows large the distribution converges with a limit of .367861 which is approximately e^{-1}. We know that $e^{x/n} = 1 + x/n$. We also know that $e^{x/n^n} = e^x$. Now, by substitution we can derive $1 + x/n^n = e^x$ which yields $1 - x/n^n = e^{-x}$ which is $1 - 1/n^n = e^{-1}$ for $x = -1$ such that $e^{-1} = 1/e$ which apprximates .367861.

Mini Project - Multi Linear Regression Using Gradient Descent with K-Fold Cross Validation

In this mini project, you'll perform a Multivariate Linear Regression analysis using Gra- dient Descent. The data set consists of two predictor variables and one response variable. The predictor variables are living area in square feet and number of bedrooms. The response variable is the price of the house. You have 47 data points in total.

Since both the number of rooms and the living area are in different units, it makes it hard to compare them in relative terms. One way to compensate for this is to standardize the variables. In order to standardize, you estimate the mean and standard deviation of each variable and then compute new versions of these variables. For instance, if you have a variable x, then then standardized version of x is $x_{std} = x - \mu\sigma$ where μ and σ are the mean and standard deviation of x.

As we saw in the gradient descent equations, we introduce a dummy variable $x_0 = 1$ in order to calculate the intercept term of the linear regression. Please standardize the 2 variables, introduce the dummy variable and then write a function to perform gradient descent on this data set. You'll repeat gradient descent for a range of α values. Please use $\alpha = (.001, .01, 0.1, 1.0)$ as your choices. Plot $J(\theta)$ versus number of iterations for each of the 4 α choices.

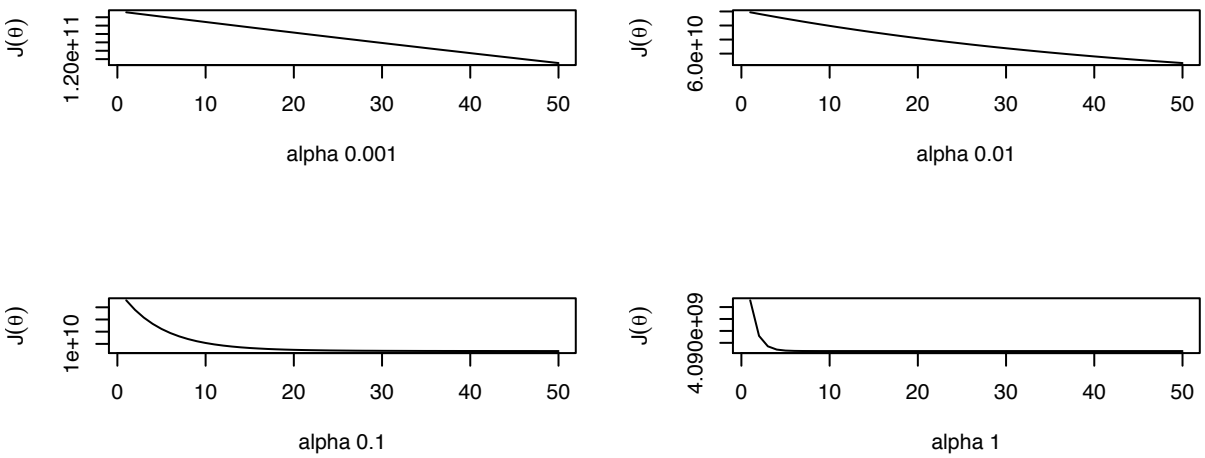
Once you have your final gradient descent solution, compare this with regular linear regression (using the built-in function in R or scikit-learn). Also solve using Ordinary Least Squares approach. Please document all 3 solutions in your submission. Finally, using the built-in cross-validation function in R or scikit-learn, perform a 5-fold cross-validation and estimate the test prediction error. In order to perform the cross- validation, you can also write your own procedure if you prefer to use your own instead of the built-in functions.

Let's read into a data frame the scaled and raw data that we create in python (see python notebook output for data acquisition code).

```
# Solution

# set working directory
setwd("~/Desktop/is605Final")
# read data into a data frame
data <-
  read.csv("~/Desktop/is605Final/MLR_GradientDescentData.csv", header = TRUE)

# set column variables
raw_area = data['RawArea']
raw_bedrooms = data['RawBedrooms']
home_px = data['HomePx']
scaled_sqft = data['ScaledSqFt']
scaled_bedrooms = data['ScaledBedrooms']


# crop data to get the columns we are interested
# in into a single df.
cropped_data =
  cbind(raw_area, raw_bedrooms, home_px, scaled_sqft, scaled_bedrooms)
cropped_data2 =
  cbind(raw_area, raw_bedrooms, home_px, scaled_sqft, scaled_bedrooms)

# cropped vars
cropped_scaledsqf = (cropped_data2[1:47,4])
cropped_scaledbeds = (cropped_data2[1:47,5])
cropped_homepx = (cropped_data2[1:47, 3])

# new data
new_data =
  cbind(cropped_scaledsqf, cropped_scaledbeds, cropped_homepx)

# set initial alpha values for the MLR using gradient descent
alpha = as.vector(c(.001, .01, .1, 1.0))
J = array(0, c(50, length(alpha)))
m = length(cropped_homepx)
theta = matrix(c(0,0,0), nrow = 1)
x =
  matrix(c(rep(1,m), cropped_scaledsqf, cropped_scaledbeds), ncol = 3)
y =
  matrix(cropped_homepx, ncol = 1)

# create a function to update deltas
delta = function(x, y, th){
  delta = (t(x) %*% ((x %*% t(th)) - y))
  return(t(delta))
}

# construct a function to compute the cost
cost = function(x, y, th, m){
  prt = ((x%*% t(th)) - y)
```

```
  return(1/m * (t(prt) %*% prt))
  }

# for each alpha value run 50 iterations and collect the errors
for(j in 1:length(alpha)){
  for(i in 1:50){
    J[i,j] = cost(x, y, theta, m)
    theta = theta - alpha[j] * 1/m * delta(x, y, theta )
    }
  }

# set graphical params
par(mfrow=c(3,2))

# plot J(theta) for each alpha
for(j in 1:length(alpha)){
  plot(J[,j], type = "l",
       xlab = paste("alpha", alpha[j]), ylab = expression(J(theta)))

}

for(i in 1:100000){
  theta = theta - 1 * 1/m * delta(x,y,theta)
  if(abs(delta(x, y, theta)[2]) < 0.0000001){
    break
  }
}

# Output Theta Values
theta
```

```
##         [,1]   [,2]  [,3]
## [1,] 340413 109448 -6578
```



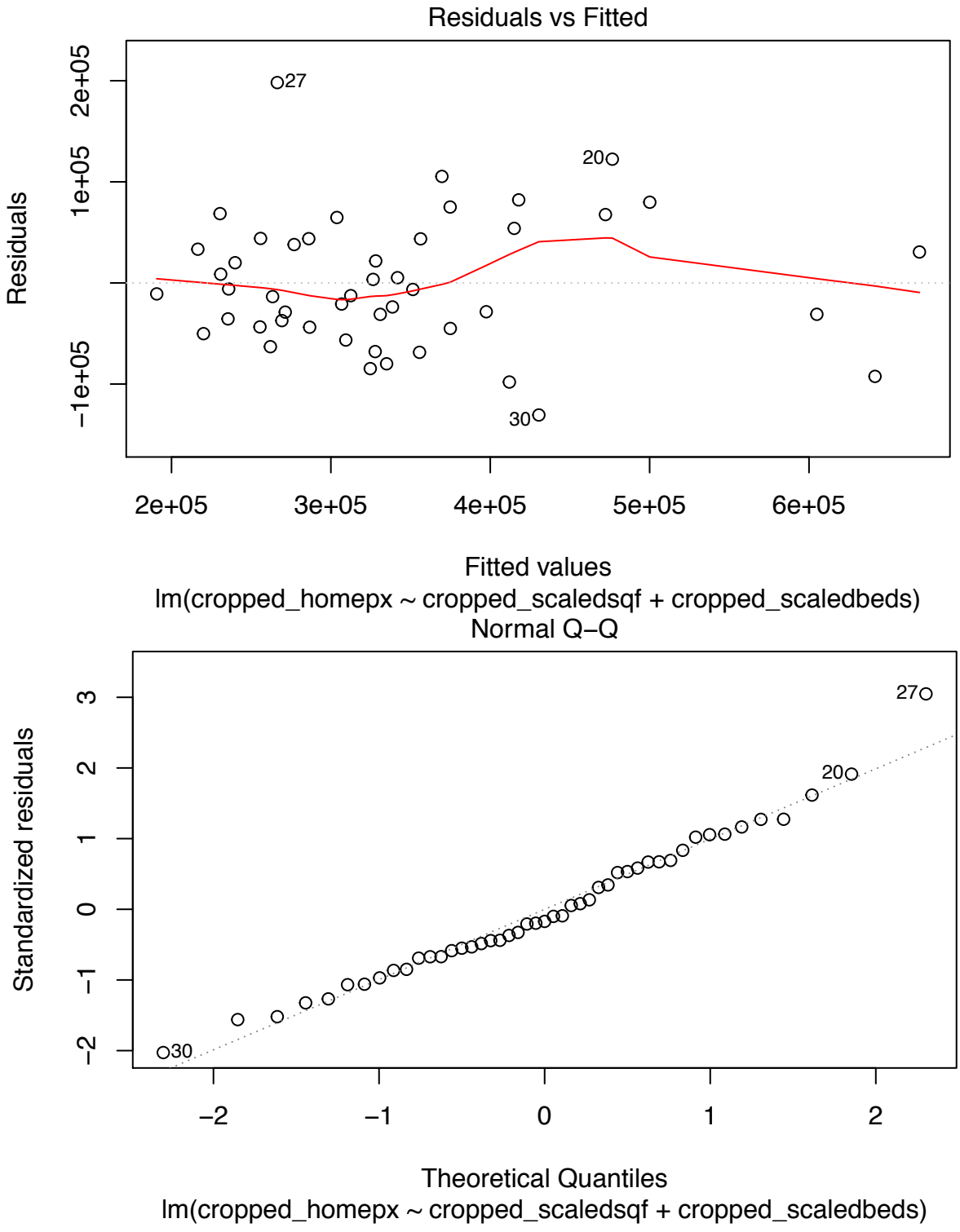
Using built in function to create 2 predictor model . . . we see that the 2 predictor model using the built in function yields the same solution as the Gradient Descent above. The multiple R-Squared of .7329 means that a significant portion of the home price is estimated by these variables of living area and bedrooms; however, the rest of the home price may be predicted by other factors such as location presumably. A p-value

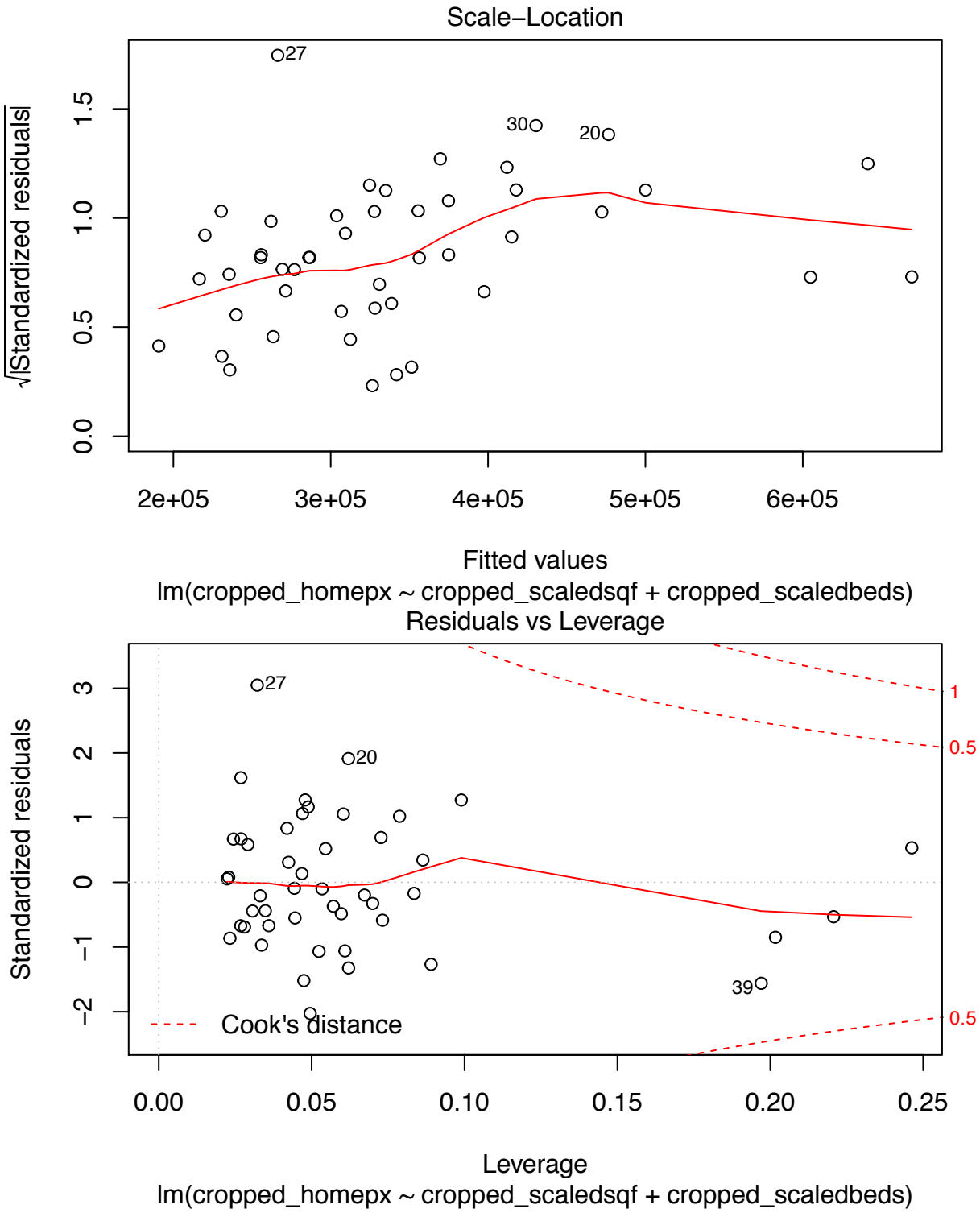
of 2.428e-13 signifies that there is enough evidence that these two predictors, living area and number of bedrooms, are predictive of home prices that the null hypothesis can confidently be rejected for the entire population.

```
# compute the 2 predictor solution using built lm function
mlr_model_2preds =
  lm(formula = cropped_homepx ~ cropped_scaledsqf + cropped_scaledbeds)
# summarize the multivariate solution
summary(mlr_model_2preds)
```

```
##
## Call:
## lm(formula = cropped_homepx ~ cropped_scaledsqf + cropped_scaledbeds)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -130582  -43636  -10829   43698  198147
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)          340413       9637   35.32  < 2e-16 ***
## cropped_scaledsqf    109448      11632    9.41  4.2e-12 ***
## cropped_scaledbeds    -6578      11632   -0.57     0.57
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 66100 on 44 degrees of freedom
## Multiple R-squared:  0.733,  Adjusted R-squared:  0.721
## F-statistic: 60.4 on 2 and 44 DF,  p-value: 2.43e-13
```

```
# plot the results
plot(mlr_model_2preds)
```


Residuals vs Fitted
Residuals
2e+05
1e+05
−1e+05
27
20
30
2e+05
3e+05
4e+05
5e+05
6e+05
Fitted values
lm(cropped_homepx ~ cropped_scaledsqf + cropped_scaledbeds)
Normal Q−Q
Standardized residuals
3
2
1
0
−1
−2
27
20
30
−2
−1
0
1
2
Theoretical Quantiles
lm(cropped_homepx ~ cropped_scaledsqf + cropped_scaledbeds)



Now we'll use manual OLS to find the solution to the 2 predictor model.

```
# create x matrix with living area, bedrooms and a dummy variable
ols_x = matrix(c(rep(1,m), data[1:47,2], data[1:47,3]), ncol = 3)

# create a y matrix that contains the response variable of home price
```

```r
ols_y = matrix(data[1:47,4], ncol = 1)

# compute theta by solving the matrix equation
ols.theta = solve(t(x) %*% x) %*% (t(x) %*% y)

# output theta
print("Theta: ")
```

```
## [1] "Theta: "
```

```r
print(ols.theta)
```

```
##        [,1]
## [1,] 340413
## [2,] 109448
## [3,]  -6578
```

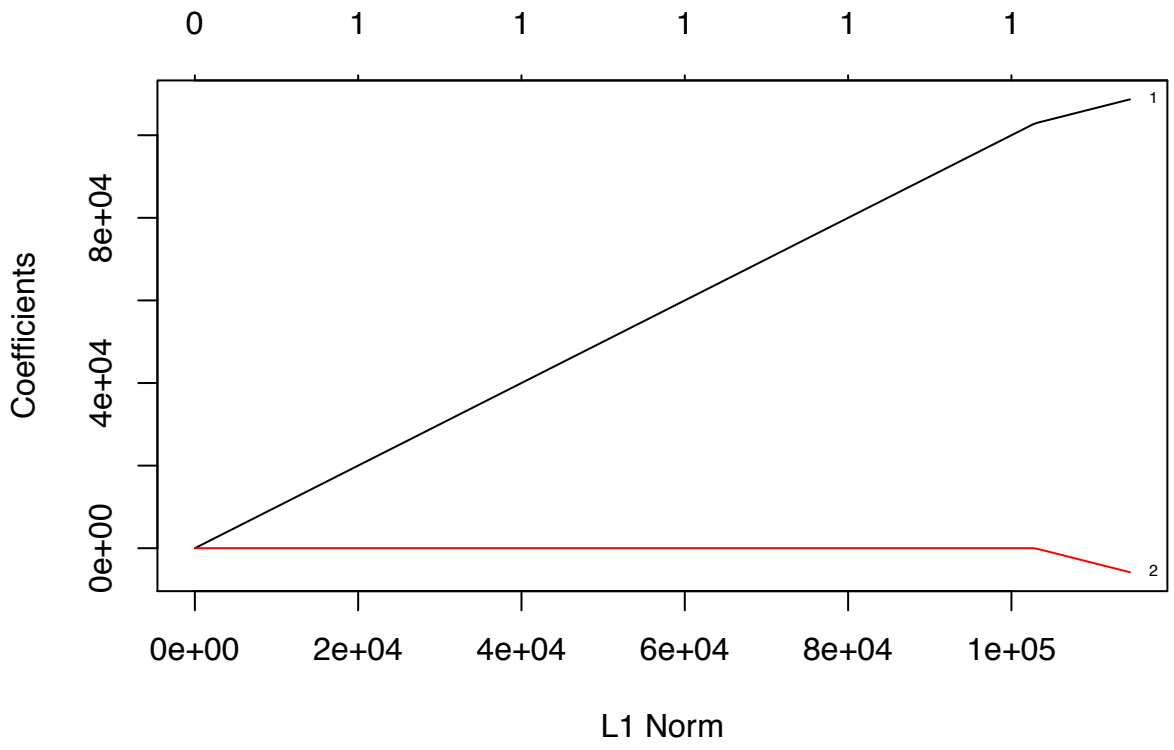
We can see above all three solutions are the same. The benefit of gradient descent is that it would be faster to run using large amounts of data.

Now we'll apply k-fold cross validation

```r
# import requisite cross
# validation library for generalized linear models
library(glmnet)
```

```
## Loading required package: Matrix
## Loaded glmnet 1.9-8
```

```r
# create area vector
area <- cbind(cropped_scaledsqf)

# create bedrooms vector
beds <- cbind(cropped_scaledbeds)

# create home price vector
homepx <- cbind(cropped_homepx)

# fit x = areas, beds and y = homepx
fit = glmnet(cbind(area, beds), homepx)

# plot the coefficients
plot(fit, label = TRUE)
```




```
# output the fit - a summary of the path at each step
# this shows the number of non-zero coefficients,
# the $ of null deviance that is explained and
#the lambda value
print(fit)
```

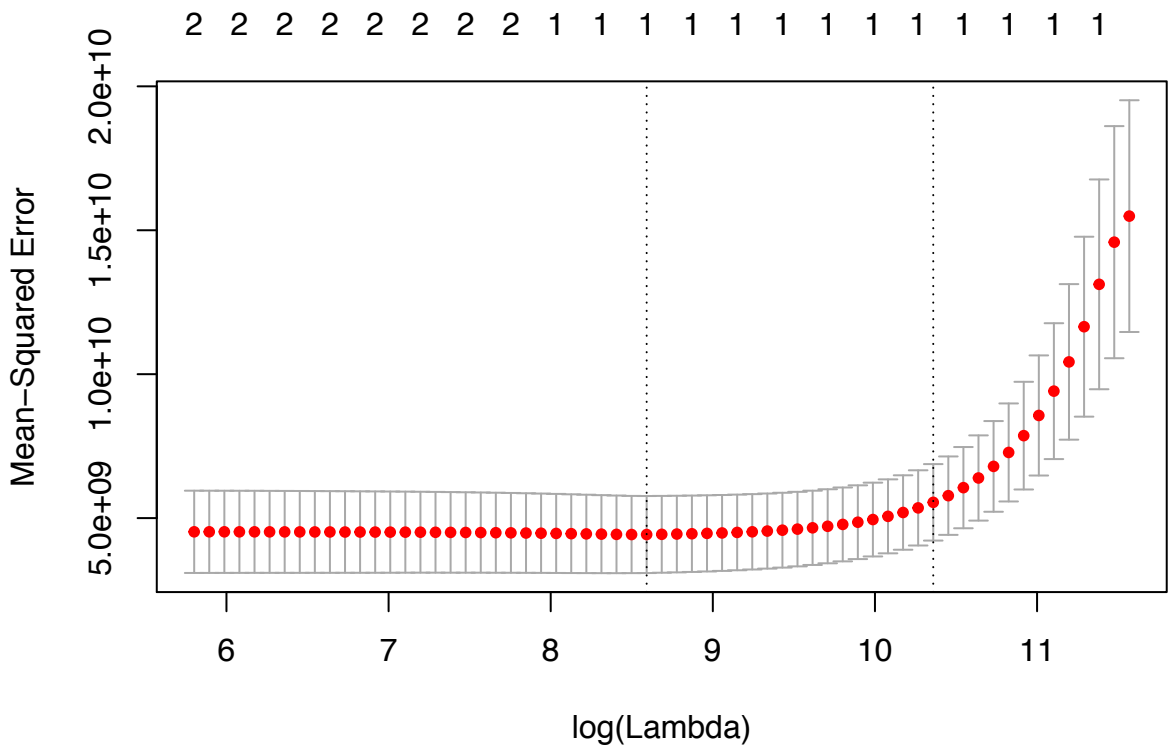
```
##
## Call:  glmnet(x = cbind(area, beds), y = homepx)
##
##       Df  %Dev Lambda
##  [1,]  0 0.000 106000
##  [2,]  1 0.124  96400
##  [3,]  1 0.227  87800
##  [4,]  1 0.313  80000
##  [5,]  1 0.384  72900
##  [6,]  1 0.443  66400
##  [7,]  1 0.492  60500
##  [8,]  1 0.532  55100
##  [9,]  1 0.566  50200
## [10,]  1 0.594  45800
## [11,]  1 0.617  41700
## [12,]  1 0.637  38000
## [13,]  1 0.653  34600
## [14,]  1 0.666  31600
## [15,]  1 0.677  28800
## [16,]  1 0.686  26200
## [17,]  1 0.694  23900
## [18,]  1 0.700  21800
## [19,]  1 0.705  19800
## [20,]  1 0.710  18100
```

```
## [21,]  1 0.713  16500
## [22,]  1 0.716  15000
## [23,]  1 0.719  13700
## [24,]  1 0.721  12400
## [25,]  1 0.723  11300
## [26,]  1 0.724  10300
## [27,]  1 0.725   9420
## [28,]  1 0.726   8580
## [29,]  1 0.727   7820
## [30,]  1 0.728   7120
## [31,]  1 0.728   6490
## [32,]  1 0.729   5910
## [33,]  1 0.729   5390
## [34,]  1 0.729   4910
## [35,]  1 0.730   4470
## [36,]  1 0.730   4080
## [37,]  1 0.730   3710
## [38,]  1 0.730   3380
## [39,]  1 0.730   3080
## [40,]  2 0.731   2810
## [41,]  2 0.731   2560
## [42,]  2 0.731   2330
## [43,]  2 0.732   2130
## [44,]  2 0.732   1940
## [45,]  2 0.732   1760
## [46,]  2 0.732   1610
## [47,]  2 0.732   1460
## [48,]  2 0.732   1330
## [49,]  2 0.733   1220
## [50,]  2 0.733   1110
## [51,]  2 0.733   1010
## [52,]  2 0.733    920
## [53,]  2 0.733    838
## [54,]  2 0.733    764
## [55,]  2 0.733    696
## [56,]  2 0.733    634
## [57,]  2 0.733    578
## [58,]  2 0.733    526
## [59,]  2 0.733    480
## [60,]  2 0.733    437
## [61,]  2 0.733    398
## [62,]  2 0.733    363
## [63,]  2 0.733    331
```

```
# this gives us the coefficients at one or
# more lambdas within the specified range.
coef(fit, s = 0.05)
```

```
## 3 x 1 sparse Matrix of class "dgCMatrix"
##                            1
## (Intercept)           340413
## cropped_scaledsqf     108692
## cropped_scaledbeds    -5825
```

```
# now we perform cross validation
crossvalid_fit = cv.glmnet(cbind(area, beds), homepx)

# plot the cross validation fit
# The plot contains a cross validation curve
# represented by a dotted red line as well as boundaries
# for standard deviation along bars represented the error.
# Vertical dotted lines represented selected errors or lambdas.
plot(crossvalid_fit)
```




```
# output the selected lambdas represented as dotted
# vertical lines in the plot above. lambda.min
# is the value of lambda that yields the minimum mean error.
crossvalid_fit$lambda.min
```

```
## [1] 5388
```

```
# coefficients that yield min error
as.matrix(coef(crossvalid_fit, s = "lambda.min"))
```

```
##                          1
## (Intercept)         340413
## cropped_scaledsqf   100376
## cropped_scaledbeds       0
```

```
x_mod = cbind(area, beds)

# make prediction using the crossvalid_fit object
predict(crossvalid_fit, newx = x_mod[1:5,], s = "lambda.min")
```

```
##           1
## [1,] 353604
## [2,] 289257
## [3,] 391395
## [4,] 265765
## [5,] 467998
```