Lab 5 MATH 342W

Loyd Flores

11:59PM March 12

Write a function spec'd as follows:

```r
#' Orthogonal Projection
#'
#' Projects vector a onto v.
#'
#' @param a   the vector to project
#' @param v   the vector projected onto
#'
#' @returns   a list of two vectors, the orthogonal projection parallel to v named a_parallel,
#'            and the orthogonal error orthogonal to v called a_perpendicular
orthogonal_projection = function(a, v){
  a_parallel = (t(t(v)) %*% t(v) / (sum(v^2))) %*% a
  list(a_parallel = a_parallel, a_perpendicular = a - a_parallel)
}
```

Provide predictions for each of these computations and then run them to make sure you're correct.

```r
orthogonal_projection(c(1,2,3,4), c(1,2,3,4))
```

```
## $a_parallel
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
## [4,]    4
##
## $a_perpendicular
##      [,1]
## [1,]    0
## [2,]    0
## [3,]    0
## [4,]    0
```

```r
#prediction:
orthogonal_projection(c(1, 2, 3, 4), c(0, 2, 0, -1))
```

```
## $a_parallel
##      [,1]
## [1,]    0
## [2,]    0
## [3,]    0
## [4,]    0
##
## $a_perpendicular
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
## [4,]    4
```

```
#prediction:
result = orthogonal_projection(c(2, 6, 7, 3), c(1, 3, 5, 7))
t(result$a_parallel) %*% result$a_perpendicular
```

```
##               [,1]
## [1,] -3.552714e-15
```

```
#prediction:
result$a_parallel + result$a_perpendicular
```

```
##      [,1]
## [1,]    2
## [2,]    6
## [3,]    7
## [4,]    3
```

```
#prediction:
result$a_parallel / c(1, 3, 5, 7)
```

```
##           [,1]
## [1,] 0.9047619
## [2,] 0.9047619
## [3,] 0.9047619
## [4,] 0.9047619
```

```
#prediction:
```

Create a vector y by simulating n = 100 standard iid normals. Create a matrix of size 100 x 2 and populate the first column by all ones (for the intercept) and the second column by 100 standard iid normals. Find the R^2 of an OLS regression of `y ~ X`. Use matrix algebra.

```
#TO-DO
n = 100
y = rnorm(n=n)
X = cbind(1, rnorm(n=n))
y_hat = X %*% solve(t(X) %*% X) %*% t(X) %*% y
y_bar = mean(y_hat)

SSR = sum((y_hat - y_bar)^2)
SST = sum((y - y_bar)^2)
```

```r
RSQ = SSR/SST

print(RSQ)
```

```
## [1] 0.002215123
```

Write a for loop to each time bind a new column of 100 standard iid normals to the matrix X and find the Rˆ2 each time until the number of columns is 100. Create a vector to save all Rˆ2's. What happened??

```r
#TO-DO
rsqs = array(NA, n)
rsqs[2] = RSQ
rsqs
```

```
##   [1]          NA 0.002215123          NA          NA          NA          NA
##   [7]          NA          NA          NA          NA          NA          NA
##  [13]          NA          NA          NA          NA          NA          NA
##  [19]          NA          NA          NA          NA          NA          NA
##  [25]          NA          NA          NA          NA          NA          NA
##  [31]          NA          NA          NA          NA          NA          NA
##  [37]          NA          NA          NA          NA          NA          NA
##  [43]          NA          NA          NA          NA          NA          NA
##  [49]          NA          NA          NA          NA          NA          NA
##  [55]          NA          NA          NA          NA          NA          NA
##  [61]          NA          NA          NA          NA          NA          NA
##  [67]          NA          NA          NA          NA          NA          NA
##  [73]          NA          NA          NA          NA          NA          NA
##  [79]          NA          NA          NA          NA          NA          NA
##  [85]          NA          NA          NA          NA          NA          NA
##  [91]          NA          NA          NA          NA          NA          NA
##  [97]          NA          NA          NA          NA
```

```r
for(j in 3:n){
  X = cbind(X, rnorm(n))  # adding more columns
  y_hat = X %*% solve(t(X) %*% X) %*% t(X) %*% y
  SSR = sum((y_hat - y_bar)^2)
  SST = sum((y - y_bar)^2)
  rsqs [j] = SSR/SST
}
rsqs
```

```
##   [1]          NA 0.002215123 0.011174512 0.022182164 0.058009793 0.058145409
##   [7] 0.059812805 0.066897525 0.066898220 0.084879133 0.088765983 0.097889910
##  [13] 0.099503035 0.099610783 0.153062277 0.155054917 0.167844571 0.168064786
##  [19] 0.177557400 0.183434250 0.191628399 0.191643113 0.212373755 0.212419466
##  [25] 0.218595918 0.219834880 0.220030253 0.220112508 0.259291171 0.259522466
##  [31] 0.271663235 0.273190255 0.273234618 0.327049524 0.336929636 0.336930166
##  [37] 0.337085676 0.338466997 0.344895571 0.345495087 0.350014929 0.350723206
##  [43] 0.360355903 0.360483960 0.363190735 0.385104798 0.413181507 0.414012912
##  [49] 0.414374558 0.428009969 0.428239988 0.476142695 0.490444178 0.492479593
##  [55] 0.531113724 0.559480526 0.560191366 0.563179870 0.569098173 0.572845601
##  [61] 0.573439571 0.575622449 0.575635118 0.580618326 0.581488543 0.596363422
##  [67] 0.596920138 0.603379978 0.618833208 0.710823662 0.726574840 0.727503906
##  [73] 0.733328135 0.733907717 0.736503853 0.743866837 0.751992759 0.752055046
##  [79] 0.755034024 0.771108970 0.774590615 0.789720090 0.790323915 0.795222188
##  [85] 0.805715210 0.811152154 0.811176477 0.818889855 0.826240932 0.849445144
##  [91] 0.854458927 0.854481141 0.859019106 0.859284801 0.879730102 0.934586524
##  [97] 0.934787350 0.937860421 0.938381483 1.000000000
```

Test that the projection matrix onto this X is the same as I_n. You may have to vectorize the matrices in the `expect_equal` function for the test to work.

```r
pacman::p_load(testthat)
#TO-DO
expect_equal(X %*% solve(t(X) %*% X) %*% t(X) , diag(n))
```

Add one final column to X to bring the number of columns to 101. Then try to compute R^2. What happens?

```r
#TO-DO
# X = cbind(X, rnorm(n=n))
# y_hat = X %*% solve(t(X) %*% X) %*% t(X) %*% y
```

Why does this make sense?

#TO-DO

We can't perform this cause this operation fails because the matrix does is not full rank anymore. X(X$^{TX)}$-1 (y). We also can only add up to n-(p+1)

Let's use the Boston Housing Data for the following exercises

```r
y = MASS::Boston$medv
X = model.matrix(medv ~ ., MASS::Boston) # RETURNS EVERYTHING BUT Y && ~. adds the 1 matrix
p_plus_one = ncol(X)
n = nrow(X)
```

Using your function `orthogonal_projection` orthogonally project onto the column space of X by projecting y on each vector of X individually and adding up the projections and call the sum `yhat_naive`.

```r
yhat_naive = matrix(0, nrow = n, ncol = 1)
for(j in 1:p_plus_one){
  yhat_naive = yhat_naive + orthogonal_projection(y, X[,j])$a_parallel
}
```

How much double counting occurred? Measure the magnitude relative to the true LS orthogonal projection.

```r
yhat = lm(medv ~ ., MASS::Boston)$fitted.values
print(sqrt(sum(yhat_naive^2)) / sqrt(sum(yhat^2)))
```

```
## [1] 8.997118
```

Is this ratio expected? Why or why not?

#TO-DO

Convert X into V where V has the same column space as X but has orthogonal columns. You can use the function `orthogonal_projection`. This is the Gram-Schmidt orthogonalization algorithm (part A).

```
V = matrix(NA, nrow = n, ncol = p_plus_one)
V[, 1] = X[, 1]
for (j in 2 : ncol(X)){
  V[, j] = X[, j]

  for (k in 1:(j-1)){
    v_k = V[, k]
    V[, j] = V[, j, drop = FALSE] - orthogonal_projection(X[,k], v_k)$a_parallel
  }
}
```

Convert V into Q whose columns are the same except normalized. This is the Gram-Schmidt orthogonalization algorithm (part B).

```
Q = matrix(NA, nrow = n, ncol = p_plus_one)
for(j in 1 : p_plus_one){
  Q[, j] = V[, j] / sqrt(sum(V[, j]^2))
}
rm(V)
```

Verify Q^T Q is I_{p+1} i.e. Q is an orthonormal matrix.

```
#TO-DO
# expect_equal(t(Q) %*% Q, diag(p_plus_one))
# print("...")
```

Is your Q the same as what results from R's built-in QR-decomposition function?

```
Q_from_Rs_builtin = qr.Q(qr(X))
# expect_equal(Q, Q_from_Rs_builtin)
```

Is this expected? Why did this happen?

#TO-DO There was no output from expect_equal meaning we successfully remade our Q properly.

Project y onto colsp[Q] and verify it is the same as the OLS fit. You may have to use the function `unname` to compare the vectors since they the entries will likely have different names.

```
#TO-DO
# y_hat_Q = Q %*% t(Q)
# y_hat_from_R = Q_from_Rs_builtin %*% t(Q_from_Rs_builtin) %*% y

# expect_equal(y_hat_Q, y_hat_from_R)
```

Project y onto colsp[Q] one by one and verify it sums to be the projection onto the whole space.

```
# yhat_naive =
```

Split the Boston Housing Data into a training set and a test set where the training set is 80% of the observations. Do so at random.

```
K = 5
n_test = round(n * 1 / K)
n_train = n - n_test
#TO-DO
#Training Data
# We're trying to find the indices that we will get onto train
train_indices = sample(1:n, n_train, replace = FALSE )
X_train = X[train_indices, ]
y_train = y[train_indices]
# Test
test_indices = setdiff(1:n, train_indices)
X_test = X[test_indices,]
y_test = y[test_indices]
```

Fit an OLS model. Find the s_e in sample and out of sample. Which one is greater? Note: we are now using s_e and not RMSE since RMSE has the n-(p + 1) in the denominator not n-1 which attempts to de-bias the error estimate by inflating the estimate when overfitting in high p. Again, we're just using `sd(e)`, the sample standard deviation of the residuals.

```
#TO-DO
b = solve(t(X_train) %*% X_train) %*% t(X_train) %*% y_train
y_hat_train = X_train %*% b
e_train = y_train - y_hat_train
RMSE_e_train = sd(e_train)

y_hat_test = X_test %*% b
e_test = y_test - y_hat_test
RMSE_e_test = sd(e_test)
```

Do these two exercises `Nsim = 1000` times and find the average difference between s_e and ooss_e.

```
#TO-DO
#TO-DO
Nsim = 1000
RMSE_trains = array(NA, Nsim)
RMSE_tests = array(NA, Nsim)

for(nsim in 1 : Nsim){
  train_indices = sample(1:n, n_train, replace = FALSE )
  X_train = X[train_indices, ]
  y_train = y[train_indices]

  test_indices = setdiff(1:n, train_indices)
  X_test = X[test_indices,]
  y_test = y[test_indices]

  b = solve(t(X_train) %*% X_train) %*% t(X_train) %*% y_train
  y_hat_train = X_train %*% b
  e_train = y_train - y_hat_train
  RMSE_trains[nsim] = sd(e_train)

  y_hat_test = X_test %*% b
  e_test = y_test - y_hat_test
  RMSE_tests[nsim] = sd(e_test)


}

  mean(RMSE_trains)
```

```
## [1] 4.649593
```

```
  mean(RMSE_tests)
```

```
## [1] 4.893221
```

We'll now add random junk to the data so that `p_plus_one = n_train` and create a new data matrix `X_with_junk`.

```
X_with_junk = cbind(X, matrix(rnorm(n * (n_train - p_plus_one)), nrow = n))
dim(X)
```

```
## [1] 506  14
```

```
dim(X_with_junk)
```

```
## [1] 506 405
```

Repeat the exercise above measuring the average s_e and ooss_e but this time record these metrics by number of features used. That is, do it for the first column of `X_with_junk` (the intercept column), then do it for the first and second columns, then the first three columns, etc until you do it for all columns of `X_with_junk`. Save these in `s_e_by_p` and `ooss_e_by_p`.

```
#TO-DO
# s_e_by_p = array(NA, n_train)
# ooss_e_by_p = array(NA, n_train)
# X_with_junk_train = X_with_junk[train_indices, ]
# y_train = y[train_indices]
# X_with_junk_test = X_with_junk[test_indices, ]
# y_test = y[test_indices]

# for (j in 1:n_train) {
  # y_hat_test = X_with_junk_test[, 1:j, drop = FALSE] %*% b
  # e_test = y_test - y_hat_test
  # ooss_e_by_p[j] = sd(e_test)
# }
```

You can graph them here:

```
# pacman::p_load(ggplot2)
# ggplot(
  # rbind(
    # data.frame(s_e = s_e_by_p, p = 1 : n_train, series = "in-sample"),
    # data.frame(s_e = ooss_e_by_p, p = 1 : n_train, series = "out-of-sample")
  # )) +
  # geom_line(aes(x = p, y = s_e, col = series))
```

Is this shape expected? Explain.

#TO-DO The in sample error decreases as we increases the number of features. The out of sample error increases as the more features we put because which signals that overfitting is happening.

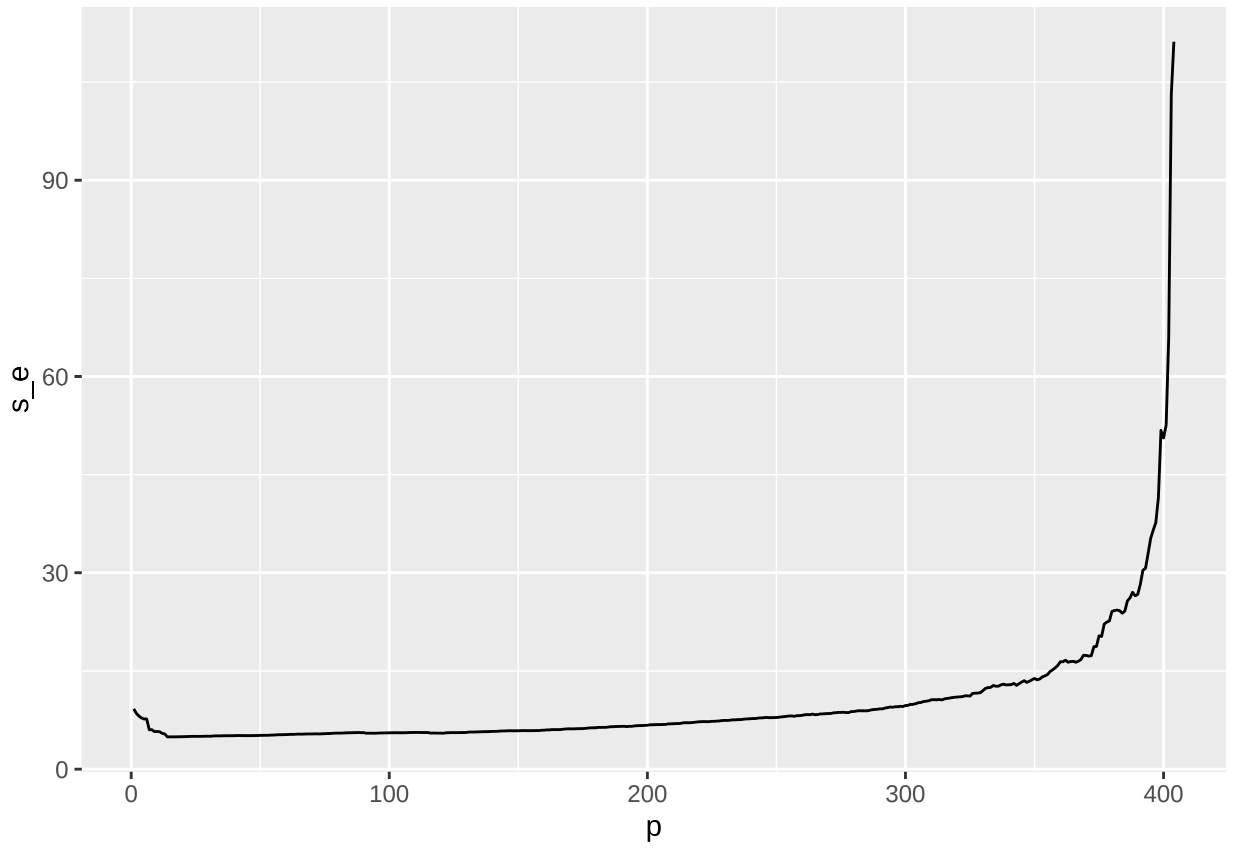
Now repeat the exercise above except use 5-fold CV (K=5 cross validation) for each p. The code below will also plot the oos RMSE. This oos RMSE curve should be similar to the curve in the above problem, but now it will be more stable.

```
K = 5
#oos_e_by_p_k = matrix(NA, nrow = n, ncol = n) #save all residuals here - each row are the residuals fo


set.seed(1984)
temp = rnorm(n)
folds_vec = cut(temp, breaks=quantile(temp, seq(0, 1, length.out=K+1)), include.lowest=TRUE, labels=FALS
head(folds_vec, 100)
```

```
##   [1] 4 2 4 1 5 5 4 1 2 5 4 4 4 5 3 5 3 1 3 1 5 4 2 5 3 4 3 4 2 5 1 1 4 5 4 1 5
##  [38] 3 2 4 3 4 3 1 5 5 2 5 2 5 5 2 1 1 2 1 2 2 2 3 2 4 5 2 5 3 2 2 3 5 2 1 4 2
##  [75] 2 3 4 2 2 2 4 4 2 1 3 5 2 1 5 3 3 2 1 5 5 4 2 2 5 1
```

```
min_ntrain = min(n - table(folds_vec))

ooss_e_by_p = array(NA, min_ntrain)
oos_se_by_p_by_k = matrix(NA, nrow = min_ntrain, ncol = K)


for (j in 1:min_ntrain) {
  all_e_test = array(NA, n)

      for (k in 1:K) {
      # Test train split
      test_indices = which(folds_vec == k)
      train_indices = setdiff(1:n, test_indices)

      #Extract X_train, y_train, X_test, y_test
      X_with_junk_train = X_with_junk[train_indices, ]
      y_train = y[train_indices]
      X_with_junk_test = X_with_junk[test_indices, ]
      y_test = y[test_indices]

      X_with_junk_train_j = X_with_junk_train[ , 1:j, drop = FALSE]
      b = solve(t(X_with_junk_train_j) %*% X_with_junk_train_j) %*% t(X_with_junk_train_j) %*% y_train
      yhat_test = X_with_junk_test[, 1:j, drop=FALSE] %*% b
      e_k = y_test - yhat_test
      all_e_test[test_indices] = e_k
      oos_se_by_p_by_k[j,k] = sd(e_k)
    }
  ooss_e_by_p[j] = sd(all_e_test)
}
```

```
#now plot it
pacman::p_load(ggplot2)
ggplot(data.frame(
    s_e = ooss_e_by_p,
    p = 1 : min_ntrain
  )) +
  geom_line(aes(x = p, y = s_e))
```



Even though the concept of confidence intervals (CIs) will not be on the midterm, construct 95% CIs for each of the oosRMSE measurements by number of features, p. A CI is a real-number interval with a lower bound and upper bound. The formula for the CI is [s_e - 2 * s_s_e, s_e + 2 * s_s_e].

```
#TO-DO
s_s_es_by_j = apply(oos_se_by_p_by_k, 1, sd)
cbind(
 ooss_e_by_p - 2 * s_s_es_by_j / sqrt(K),
 ooss_e_by_p + 2 * s_s_es_by_j / sqrt(K)
)
```

```
##             [,1]      [,2]
##   [1,] 8.069673 10.357631
##   [2,] 7.580231  9.473365
##   [3,] 7.238791  8.996341
##   [4,] 6.726156  8.901798
##   [5,] 6.717750  8.609315
##   [6,] 6.686835  8.652508
##   [7,] 4.909580  7.145575
##   [8,] 4.881283  7.173938
##   [9,] 4.774459  6.724102
##  [10,] 4.773764  6.755036
##  [11,] 4.762907  6.686066
##  [12,] 4.442393  6.505598
##  [13,] 4.393281  6.391162
##  [14,] 4.318245  5.538931
##  [15,] 4.322647  5.538648
##  [16,] 4.326070  5.541187
##  [17,] 4.322268  5.545488
##  [18,] 4.329596  5.549524
##  [19,] 4.334601  5.561769
##  [20,] 4.349983  5.575114
##  [21,] 4.375902  5.586426
##  [22,] 4.385514  5.606221
##  [23,] 4.402836  5.626940
##  [24,] 4.405353  5.634300
##  [25,] 4.405788  5.631513
##  [26,] 4.411585  5.620893
##  [27,] 4.414014  5.626543
##  [28,] 4.421548  5.633224
##  [29,] 4.428912  5.643279
##  [30,] 4.428661  5.644754
##  [31,] 4.424289  5.645154
##  [32,] 4.430564  5.709342
##  [33,] 4.449645  5.715657
##  [34,] 4.448996  5.714153
##  [35,] 4.453653  5.724035
##  [36,] 4.457177  5.749324
##  [37,] 4.450391  5.769170
##  [38,] 4.458420  5.778821
##  [39,] 4.459178  5.765446
##  [40,] 4.468248  5.771820
##  [41,] 4.471443  5.826592
##  [42,] 4.487695  5.806678
##  [43,] 4.472221  5.803565
##  [44,] 4.500423  5.756267
##  [45,] 4.497173  5.756924
##  [46,] 4.507364  5.740323
##  [47,] 4.512924  5.753542
##  [48,] 4.513180  5.779756
##  [49,] 4.518047  5.783050
##  [50,] 4.552409  5.785353
##  [51,] 4.548488  5.798773
##  [52,] 4.566366  5.790490
##  [53,] 4.573750  5.792911
##  [54,]  4.595925   5.803504
##  [55,]  4.614925   5.805605
##  [56,]  4.615911   5.826706
##  [57,]  4.557278   5.973692
##  [58,]  4.536740   5.996872
##  [59,]  4.574112   5.972113
##  [60,]  4.603009   5.979629
##  [61,]  4.621543   5.999257
##  [62,]  4.625854   6.020262
##  [63,]  4.636368   6.015679
##  [64,]  4.654378   6.041732
##  [65,]  4.658875   6.053296
##  [66,]  4.653628   6.041975
##  [67,]  4.670288   6.054073
##  [68,]  4.685843   6.060327
##  [69,]  4.687640   6.056968
##  [70,]  4.693103   6.057943
##  [71,]  4.696075   6.080362
##  [72,]  4.699706   6.081940
##  [73,]  4.669302   6.089859
##  [74,]  4.682876   6.120809
##  [75,]  4.716535   6.130579
##  [76,]  4.754735   6.141756
##  [77,]  4.766621   6.154927
##  [78,]  4.787632   6.189010
##  [79,]  4.804320   6.192820
##  [80,]  4.824586   6.188990
##  [81,]  4.821648   6.192794
##  [82,]  4.832352   6.202618
##  [83,]  4.856587   6.225926
##  [84,]  4.867211   6.249853
##  [85,]  4.873816   6.253553
##  [86,]  4.877597   6.276657
##  [87,]  4.885662   6.288341
##  [88,]  4.890636   6.333771
##  [89,]  4.867894   6.285472
##  [90,]  4.855674   6.305231
##  [91,]  4.781065   6.210438
##  [92,]  4.789481   6.213292
##  [93,]  4.836242   6.153766
##  [94,]  4.835294   6.149513
##  [95,]  4.841053   6.159907
##  [96,]  4.863823   6.166840
##  [97,]  4.872492   6.178427
##  [98,]  4.877108   6.189010
##  [99,]  4.890512   6.192278
## [100,]  4.904511   6.197102
## [101,]  4.947112   6.178091
## [102,]  4.947570   6.178981
## [103,]  4.952495   6.185167
## [104,]  4.954203   6.169821
## [105,]  4.961105   6.172064
## [106,]  4.962911   6.174059
## [107,]  4.969866   6.200944
## [108,]  5.002777   6.222678
## [109,]  5.017166   6.211053
## [110,]  5.026836   6.215071
## [111,]  5.018786   6.215331
## [112,]  5.016177   6.216253
## [113,]  5.000275   6.208870
## [114,]  5.004176   6.208532
## [115,]  5.004950   6.203287
## [116,]  4.892905   6.120653
## [117,]  4.918883   6.125607
## [118,]  4.921964   6.097324
## [119,]  4.914265   6.097684
## [120,]  4.916227   6.098971
## [121,]  4.903063   6.073736
## [122,]  4.978102   6.127961
## [123,]  4.991955   6.147911
## [124,]  5.011416   6.177296
## [125,]  5.012548   6.167994
## [126,]  5.021265   6.152960
## [127,]  5.028407   6.157156
## [128,]  5.037747   6.163872
## [129,]  5.036729   6.162271
## [130,]  5.084229   6.164541
## [131,]  5.113946   6.221968
## [132,]  5.119683   6.226283
## [133,]  5.121028   6.220497
## [134,]  5.139354   6.243187
## [135,]  5.143313   6.250897
## [136,]  5.178901   6.265681
## [137,]  5.191495   6.270901
## [138,]  5.196800   6.278408
## [139,]  5.218474   6.305824
## [140,]  5.228179   6.332579
## [141,]  5.228064   6.331893
## [142,]  5.246482   6.331685
## [143,]  5.300594   6.343781
## [144,]  5.310695   6.348718
## [145,]  5.338193   6.339156
## [146,]  5.369080   6.347360
## [147,]  5.373238   6.364830
## [148,]  5.358247   6.354372
## [149,]  5.364874   6.359959
## [150,]  5.366329   6.359906
## [151,]  5.404876   6.367904
## [152,]  5.435594   6.366820
## [153,]  5.427067   6.346706
## [154,]  5.426542   6.345656
## [155,]  5.427133   6.350713
## [156,]  5.440462   6.354330
## [157,]  5.443534   6.372279
## [158,]  5.463048   6.333457
## [159,]  5.521911   6.384299
## [160,]  5.539813   6.378796
## [161,]  5.522689   6.456538
## [162,]  5.529103  6.468117
## [163,]  5.594882  6.494910
## [164,]  5.607912  6.504190
## [165,]  5.576029  6.516841
## [166,]  5.579512  6.517672
## [167,]  5.652708  6.539262
## [168,]  5.652647  6.591054
## [169,]  5.705896  6.600524
## [170,]  5.706476  6.617434
## [171,]  5.679392  6.620942
## [172,]  5.715618  6.620740
## [173,]  5.729332  6.644845
## [174,]  5.721874  6.664574
## [175,]  5.726551  6.672428
## [176,]  5.740758  6.745273
## [177,]  5.789895  6.767365
## [178,]  5.824263  6.778556
## [179,]  5.850620  6.780374
## [180,]  5.864978  6.801373
## [181,]  5.939112  6.835350
## [182,]  5.914845  6.865959
## [183,]  5.931552  6.865928
## [184,]  5.926882  6.868535
## [185,]  5.998273  6.887868
## [186,]  6.008727  6.955229
## [187,]  6.002921  6.982790
## [188,]  6.068442  6.984612
## [189,]  6.060905  7.009715
## [190,]  6.065553  7.026653
## [191,]  6.072368  7.041224
## [192,]  6.000886  7.026718
## [193,]  6.044852  7.056894
## [194,]  6.044577  7.056335
## [195,]  6.064544  7.149130
## [196,]  6.152265  7.135229
## [197,]  6.189519  7.143184
## [198,]  6.211715  7.128451
## [199,]  6.258286  7.139014
## [200,]  6.286864  7.140736
## [201,]  6.284872  7.246971
## [202,]  6.270151  7.294915
## [203,]  6.300698  7.296718
## [204,]  6.324510  7.314780
## [205,]  6.315274  7.312613
## [206,]  6.334259  7.344184
## [207,]  6.335728  7.349114
## [208,]  6.384547  7.438785
## [209,]  6.377836  7.448252
## [210,]  6.416931  7.458646
## [211,]  6.456728  7.482994
## [212,]  6.474482  7.483796
## [213,]  6.474897  7.538388
## [214,]  6.518587  7.655504
## [215,]  6.528192  7.662300
## [216,]  6.516880   7.660014
## [217,]  6.549233   7.676063
## [218,]  6.586603   7.728331
## [219,]  6.640216   7.748596
## [220,]  6.670404   7.792706
## [221,]  6.714407   7.806436
## [222,]  6.731324   7.828587
## [223,]  6.722158   7.791813
## [224,]  6.704065   7.827766
## [225,]  6.772628   7.874619
## [226,]  6.747863   7.876833
## [227,]  6.800317   7.890095
## [228,]  6.785564   7.933914
## [229,]  6.838053   8.031076
## [230,]  6.856350   8.062126
## [231,]  6.877954   8.024300
## [232,]  6.939353   8.015257
## [233,]  6.950242   8.100795
## [234,]  6.961252   8.093306
## [235,]  7.036861   8.120173
## [236,]  7.048596   8.097783
## [237,]  7.102942   8.156447
## [238,]  7.133223   8.187573
## [239,]  7.113374   8.229633
## [240,]  7.115489   8.323441
## [241,]  7.141348   8.347867
## [242,]  7.154496   8.343856
## [243,]  7.166757   8.454112
## [244,]  7.159159   8.482010
## [245,]  7.177926   8.534180
## [246,]  7.213068   8.620698
## [247,]  7.124571   8.632461
## [248,]  7.280882   8.464456
## [249,]  7.276543   8.495035
## [250,]  7.339225   8.464141
## [251,]  7.380161   8.505928
## [252,]  7.398927   8.559072
## [253,]  7.533193   8.536343
## [254,]  7.577453   8.590068
## [255,]  7.595512   8.643754
## [256,]  7.573662   8.675885
## [257,]  7.525495   8.640346
## [258,]  7.614231   8.725087
## [259,]  7.628481   8.756320
## [260,]  7.692407   8.777438
## [261,]  7.766144   8.851724
## [262,]  7.798704   8.875098
## [263,]  7.801629   8.859606
## [264,]  7.846615   8.993562
## [265,]  7.727959   8.913120
## [266,]  7.853633   8.865003
## [267,]  7.892469   8.971030
## [268,]  7.851558   9.016540
## [269,]  7.925968   9.038106
## [270,]  7.964169   9.056382
## [271,]  7.962090   9.063640
## [272,]  7.986063   9.193511
## [273,]  8.048345   9.201017
## [274,]  8.149606   9.226815
## [275,]  8.141139   9.207232
## [276,]  8.179750   9.218415
## [277,]  8.054660   9.259306
## [278,]  8.059627   9.247561
## [279,]  8.110890   9.493769
## [280,]  8.038365   9.599556
## [281,]  8.088143   9.678545
## [282,]  8.172796   9.679572
## [283,]  8.155260   9.676195
## [284,]  8.163600   9.663091
## [285,]  8.148331   9.669333
## [286,]  8.165918   9.803716
## [287,]  8.246865   9.885982
## [288,]  8.361780   9.914962
## [289,]  8.392569   9.910401
## [290,]  8.435100   9.978272
## [291,]  8.428927   9.972679
## [292,]  8.604515  10.053855
## [293,]  8.708548  10.097859
## [294,]  8.828500  10.161292
## [295,]  8.788267  10.141075
## [296,]  8.899553  10.148694
## [297,]  8.955867  10.118021
## [298,]  8.918919  10.318101
## [299,]  8.883015  10.271320
## [300,]  9.068198  10.367389
## [301,]  9.142028  10.379665
## [302,]  9.366978  10.447903
## [303,]  9.357168  10.465667
## [304,]  9.435552  10.580928
## [305,]  9.527199  10.796423
## [306,]  9.543390  10.840163
## [307,]  9.708552  10.983753
## [308,]  9.751832  11.008048
## [309,]  9.836689  11.043487
## [310,] 10.001123  11.202094
## [311,]  9.995501  11.255867
## [312,]  9.994170  11.185010
## [313,] 10.099024  11.213677
## [314,]  9.984610  11.172071
## [315,]  9.931569  11.487730
## [316,] 10.078190  11.547889
## [317,] 10.131702  11.563513
## [318,] 10.246885  11.615632
## [319,] 10.287783  11.698650
## [320,] 10.341686  11.680785
## [321,] 10.318049  11.766333
## [322,] 10.378544  11.792202
## [323,] 10.425889  11.954747
## [324,] 10.435242  11.964775
## [325,] 10.415830  11.884322
## [326,] 10.647942  12.510659
## [327,] 10.699682  12.533985
## [328,] 10.714864  12.525443
## [329,] 10.759112  12.610524
## [330,] 10.660814  13.334579
## [331,] 10.855924  13.847746
## [332,] 10.866430  14.054298
## [333,] 11.043734  13.957085
## [334,] 11.726857  13.832027
## [335,] 11.528090  13.830699
## [336,] 11.500453  13.854321
## [337,] 11.812931  13.942281
## [338,] 11.926994  14.067116
## [339,] 11.788533  13.936692
## [340,] 11.715967  14.068329
## [341,] 11.724660  14.132617
## [342,] 12.040823  14.171520
## [343,] 11.812642  13.792956
## [344,] 12.081466  14.015497
## [345,] 12.359225  14.259344
## [346,] 12.812506  14.221948
## [347,] 12.499235  14.018582
## [348,] 12.890045  13.974325
## [349,] 13.050080  14.272128
## [350,] 13.447450  14.276104
## [351,] 13.225685  14.087828
## [352,] 13.387547  14.146756
## [353,] 13.708587  14.487745
## [354,] 13.975363  14.534311
## [355,] 14.177553  14.700246
## [356,] 14.578375  15.224140
## [357,] 14.899522  15.458317
## [358,] 14.976516  15.987159
## [359,] 15.306351  16.392824
## [360,] 15.805627  16.995577
## [361,] 15.753847  17.074720
## [362,] 15.971285  17.333961
## [363,] 15.387474  17.265692
## [364,] 15.586758  17.306916
## [365,] 15.806462  17.179453
## [366,] 15.782954  16.879773
## [367,] 15.995710  16.995541
## [368,] 16.169424  17.352739
## [369,] 16.940254  17.896205
## [370,] 16.697133  18.076576
## [371,] 16.669914  17.872292
## [372,] 16.731249  17.952351
## [373,] 16.584225  20.839932
## [374,] 16.633303  20.894093
## [375,] 17.929568  22.810221
## [376,] 18.014195  22.554190
## [377,] 19.562223  24.782448
## [378,] 19.914871  25.038065
## [379,] 19.458589  25.821766
## [380,] 21.632842  26.561442
## [381,] 21.398083  27.083371
## [382,] 21.832697  26.803452
## [383,] 21.730562  26.679790
## [384,] 21.450470  26.198481
## [385,] 21.321242  26.997114
## [386,] 21.679717  29.745979
## [387,] 22.408654  29.941926
## [388,] 21.999238  32.055503
## [389,] 22.214752  30.774942
## [390,] 22.423980  30.998190
## [391,] 21.905324  34.611876
## [392,] 22.197887  38.594063
## [393,] 24.026485  37.334691
## [394,] 24.293867  41.539104
## [395,] 28.555132  41.970682
## [396,] 29.984854  43.086428
## [397,] 33.997605  41.325287
## [398,] 36.999507  45.956837
## [399,] 39.461727  64.029863
## [400,] 41.854820  59.337245
## [401,] 40.848325  64.338179
## [402,] 42.178193  89.684480
## [403,] 49.312048 156.885688
## [404,] 69.455602 152.855091
```