Lab 5

Tyron Samaroo

11:59PM March 7, 2020

Load the Boston housing data frame and create the vector y (the median value) and matrix X (all other features) from the data frame. Name the columns the same as Boston except for the first name it "(Intercept)".

```
y = MASS::Boston$medv
X = MASS::Boston[ , 1:13]
```

Run the OLS linear model to get b, the vector of coefficients. Do not use `lm`. This is $(XX^T)^{-1}X^Ty$

```
X = cbind(1,as.matrix(X))
b = solve(t(X) %*% X ) %*% t(X) %*% y
```

Find the hat matrix for this regression `H`. Verify its dimension is correct and verify its rank is correct. $H = X(X^TX)^{-1}X^T$

```
H = X %*% solve(t(X) %*% X) %*% t(X)
dim(H)
```

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

```
pacman::p_load(Matrix)
rankMatrix(H)
```

```
## [1] 14
## attr(,"method")
## [1] "tolNorm2"
## attr(,"useGrad")
## [1] FALSE
## attr(,"tol")
## [1] 1.123546e-13
```

Verify this is a projection matrix by verifying the two sufficient conditions. Use the `testthat` library's `expect_equal(matrix1, matrix2, tolerance = 1e-2)`.

```
pacman::p_load(testthat)
expect_equal(H, t(H))
expect_equal(H, H %*% H)
```

Find the matrix that projects onto the space of residuals `Hcomp` and find its rank. Is this rank expected?

```
I = diag(nrow(H))
Hcomp = I - H
rankMatrix(H)
```

```
## [1] 14
## attr(,"method")
## [1] "tolNorm2"
## attr(,"useGrad")
## [1] FALSE
```

```
## attr(,"tol")
## [1] 1.123546e-13
```

```
rankMatrix(Hcomp, tol=1e-2)
```

```
## [1] 492
## attr(,"method")
## [1] "tolNorm2"
## attr(,"useGrad")
## [1] FALSE
## attr(,"tol")
## [1] 0.01
```

Verify this is a projection matrix by verifying the two sufficient conditions. Use the `testthat` library.

```
expect_equal(Hcomp, t(Hcomp))
expect_equal(Hcomp, Hcomp %*% Hcomp)
```

Use `diag` to find the trace of both `H` and `Hcomp`.

```
sum(diag(H))
```

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

```
sum(diag(Hcomp))
```

```
## [1] 492
```

Do you have a conjecture about the trace of an orthogonal projection matrix?

trace is equal to the rank

Find the eigendecomposition of both `H` and `Hcomp` as `eigenvals_H`, `eigenvecs_H`, `eigenvals_Hcomp`, `eigenvecs_Hcomp`. Verify these results are the correct dimensions.

```
eigen_H = eigen(H)
eigen_Hcomp = eigen(Hcomp)

eigenvals_H = eigen_H$values
eigenvecs_H = eigen_H$vectors
eigenvals_Hcomp = eigen_Hcomp$values
eigenvecs_Hcomp = eigen_Hcomp$vectors

length(eigenvals_H)
```

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

```
dim(eigenvecs_H)
```

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

```
length(eigenvals_Hcomp)
```

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

```
dim(eigenvecs_Hcomp)
```

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

The eigendecomposition suffers from numerical error which is making them become imaginary. We can coerce imaginary numbers back to real by using the `Re` function. There is also lots of numerical error. Use the `Re` function to coerce to real and the `round` function to round all four objects to the nearest 10 digits.

```r
eigenvals_H = round(as.numeric(eigenvals_H), 10)
eigenvecs_H = round(Re(eigenvecs_H), 10)
eigenvals_Hcomp = round(as.numeric(eigenvals_Hcomp), 10)
eigenvecs_Hcomp = round(Re(eigenvecs_Hcomp), 10)
```

Print out the eigenvalues of both `H` and `Hcomp`. Is this expected?

```r
eigenvals_H
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [38] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [75] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [112] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [149] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [186] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [223] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [260] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [297] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [334] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [371] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [408] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [445] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [482] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```r
eigenvals_Hcomp
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [260] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [297] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [334] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [371] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [408] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [445] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [482] 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Find the length of all eigenvectors of `H` in one line.

```r
apply(eigenvecs_H, MARGIN =2, FUN = function(v){
  sqrt(sum(v^2))
})
```

```
##   [1] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##   [8] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##  [15] 0.7198274 0.7198274 0.7111798 0.7111798 0.6940586 0.6940586 1.0000000
##  [22] 0.7122684 0.7122684 0.7192493 0.7192493 0.7847285 0.7847285 0.8376704
##  [29] 0.8376704 0.8282487 0.8282487 0.7586616 0.7586616 0.9528857 0.9528857
##  [36] 0.6589249 0.6589249 0.7628320 0.7628320 0.6864124 0.6864124 0.6944800
##  [43] 0.6944800 0.8044639 0.8044639 0.9385394 0.9385394 0.7220811 0.7220811
##  [50] 0.7623481 0.7623481 0.7133975 0.7133975 0.7339666 0.7339666 0.7071697
##  [57] 0.7071697 0.6972923 0.6972923 0.6866897 0.6866897 0.7156528 0.7156528
##  [64] 0.7069016 0.7069016 0.7091819 0.7091819 0.7043276 0.7043276 0.7375631
```

```
##  [71] 0.7375631 0.7200799 0.7200799 0.7067433 0.7067433 1.0000000 0.7291502
##  [78] 0.7291502 0.9861821 0.9861821 0.7181425 0.7181425 0.7289067 0.7289067
##  [85] 0.7069350 0.7069350 0.7147271 0.7147271 0.7278385 0.7278385 0.7016417
##  [92] 0.7016417 0.6985773 0.6985773 0.6978982 0.6978982 0.6983338 0.6983338
##  [99] 0.7001943 0.7001943 0.7051763 0.7051763 0.7555366 0.7555366 1.0000000
## [106] 0.6876725 0.6876725 0.7150475 0.7150475 0.7486783 0.7486783 0.7261142
## [113] 0.7261142 0.6985145 0.6985145 0.6836561 0.6836561 1.0000000 0.7187066
## [120] 0.7187066 0.6905980 0.6905980 0.7179283 0.7179283 0.7222461 0.7222461
## [127] 0.7207965 0.7207965 0.7004181 0.7004181 0.7014897 0.7014897 0.6990003
## [134] 0.6990003 0.7115622 0.7115622 0.7237343 0.7237343 0.7703283 0.7703283
## [141] 0.7899225 0.7899225 0.7259246 0.7259246 0.7076892 0.7076892 0.7727441
## [148] 0.7727441 0.7179819 0.7179819 0.7213128 0.7213128 0.6696315 0.6696315
## [155] 0.7021261 0.7021261 0.7045330 0.7045330 0.7363510 0.7363510 1.0000000
## [162] 0.6941660 0.6941660 0.7753994 0.7753994 0.7540773 0.7540773 0.7070886
## [169] 0.7070886 0.6903144 0.6903144 0.6988027 0.6988027 0.7120509 0.7120509
## [176] 0.7158204 0.7158204 0.7312673 0.7312673 1.0000000 0.6920332 0.6920332
## [183] 0.6866946 0.6866946 0.7373332 0.7373332 0.7270449 0.7270449 0.7073476
## [190] 0.7073476 0.7085796 0.7085796 0.7045089 0.7045089 0.7563864 0.7563864
## [197] 0.7618808 0.7618808 0.7044355 0.7044355 0.8477404 0.8477404 0.7109985
## [204] 0.7109985 0.6987366 0.6987366 0.7708438 0.7708438 0.6986430 0.6986430
## [211] 0.7336177 0.7336177 1.0000000 0.6825439 0.6825439 0.6940425 0.6940425
## [218] 0.7314948 0.7314948 0.7171839 0.7171839 0.7281819 0.7281819 1.0000000
## [225] 0.7031402 0.7031402 0.6951803 0.6951803 0.7152899 0.7152899 0.7281453
## [232] 0.7281453 0.7160614 0.7160614 0.7208316 0.7208316 0.7089619 0.7089619
## [239] 0.7120048 0.7120048 0.7095858 0.7095858 0.8786539 0.8786539 0.6955125
## [246] 0.6955125 0.7157227 0.7157227 1.0000000 0.7616379 0.7616379 0.7383818
## [253] 0.7383818 0.7198086 0.7198086 0.7054253 0.7054253 0.7185741 0.7185741
## [260] 0.7113626 0.7113626 0.7210185 0.7210185 0.7061893 0.7061893 0.9134175
## [267] 0.9134175 0.7129986 0.7129986 0.7232002 0.7232002 0.7393014 0.7393014
## [274] 0.7209286 0.7209286 0.7185819 0.7185819 0.7586169 0.7586169 0.7560925
## [281] 0.7560925 0.7144534 0.7144534 0.7047713 0.7047713 0.7334610 0.7334610
## [288] 0.7207679 0.7207679 0.6943252 0.6943252 1.0000000 0.7289015 0.7289015
## [295] 0.7186150 0.7186150 0.7316530 0.7316530 0.8856238 0.8856238 0.7071873
## [302] 0.7071873 0.7283910 0.7283910 0.7209085 0.7209085 0.7375309 0.7375309
## [309] 0.7114813 0.7114813 0.7944529 0.7944529 0.7303951 0.7303951 0.7186130
## [316] 0.7186130 0.8186602 0.8186602 0.6977128 0.6977128 0.7075066 0.7075066
## [323] 0.7688476 0.7688476 0.6874561 0.6874561 0.7118374 0.7118374 1.0000000
## [330] 0.6779750 0.6779750 0.7146818 0.7146818 0.7260265 0.7260265 0.7126152
## [337] 0.7126152 1.0000000 0.7150297 0.7150297 0.7228253 0.7228253 0.7251544
## [344] 0.7251544 0.7112836 0.7112836 0.7131162 0.7131162 0.7254483 0.7254483
## [351] 0.7059673 0.7059673 0.7087931 0.7087931 0.7081581 0.7081581 0.7231071
## [358] 0.7231071 0.7004617 0.7004617 0.8706722 0.8706722 0.7317309 0.7317309
## [365] 0.7945923 0.7945923 1.0000000 0.7322846 0.7322846 0.7326250 0.7326250
## [372] 0.7174917 0.7174917 0.7517336 0.7517336 0.7375921 0.7375921 0.6711866
## [379] 0.6711866 0.6895559 0.6895559 0.6880716 0.6880716 0.7022134 0.7022134
## [386] 0.7630476 0.7630476 0.7072201 0.7072201 0.7656041 0.7656041 0.6909803
## [393] 0.6909803 0.6955879 0.6955879 0.7054902 0.7054902 0.8115084 0.8115084
## [400] 0.6961595 0.6961595 1.0000000 0.6879843 0.6879843 0.7306674 0.7306674
## [407] 0.8083010 0.8083010 0.8121622 0.8121622 0.6934602 0.6934602 0.7164527
## [414] 0.7164527 0.7074909 0.7074909 0.7179766 0.7179766 0.7076450 0.7076450
## [421] 0.7497639 0.7497639 1.0000000 0.8140732 0.8140732 0.7093069 0.7093069
## [428] 0.9117919 0.9117919 0.9473980 0.9473980 0.7074843 0.7074843 0.7350700
## [435] 0.7350700 0.7417871 0.7417871 0.7547830 0.7547830 0.7366121 0.7366121
## [442] 0.7769113 0.7769113 0.7810845 0.7810845 0.6892008 0.6892008 0.8055246
```

```
## [449] 0.8055246 0.6814249 0.6814249 0.7272492 0.7272492 0.7020231 0.7020231
## [456] 0.7611428 0.7611428 0.7367486 0.7367486 0.7048714 0.7048714 0.7400683
## [463] 0.7400683 1.0000000 0.7952697 0.7952697 0.6912670 0.6912670 0.8435088
## [470] 0.8435088 0.7646925 0.7646925 0.7883719 0.7883719 0.8329977 0.8329977
## [477] 0.7629826 0.7629826 1.0000000 0.7070124 0.7070124 0.7649891 0.7649891
## [484] 0.6532954 0.6532954 0.7107735 0.7107735 1.0000000 0.7442381 0.7442381
## [491] 1.0000000 0.7999522 0.7999522 1.0000000 0.7369916 0.7369916 0.7339597
## [498] 0.7339597 0.9876976 0.9876976 0.9150144 0.9150144 0.8738908 0.8738908
## [505] 1.0000000 1.0000000
```

Is this expected? What is the convention for eigenvectors in R's `eigen` function?

Yes. The convention is length 1.

The first p+1 eigenvectors are the columns of X but they are in arbitrary order. Find the column that represents the one-vector.

```r
head(eigenvecs_H[, 3])
```

```
## [1] 0.04561114 0.04803979 0.04756035 0.04872081 0.04788055 0.04985263
```

Why is it not exactly 506 1's?

Numeric error

Use the first p+1 eigenvectors as a model matrix and run the OLS model of medv on that model matrix.

```r
mod1 = lm(y ~ X)
mod2 = lm(y ~ eigenvecs_H[, 1:14])
summary(mod1)
```

```
##
## Call:
## lm(formula = y ~ X)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.595  -2.730  -0.518   1.777  26.199
##
## Coefficients: (1 not defined because of singularities)
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.646e+01  5.103e+00   7.144 3.28e-12 ***
## X                   NA         NA      NA       NA
## Xcrim       -1.080e-01  3.286e-02  -3.287 0.001087 **
## Xzn          4.642e-02  1.373e-02   3.382 0.000778 ***
## Xindus       2.056e-02  6.150e-02   0.334 0.738288
## Xchas        2.687e+00  8.616e-01   3.118 0.001925 **
## Xnox        -1.777e+01  3.820e+00  -4.651 4.25e-06 ***
## Xrm          3.810e+00  4.179e-01   9.116  < 2e-16 ***
## Xage         6.922e-04  1.321e-02   0.052 0.958229
## Xdis        -1.476e+00  1.995e-01  -7.398 6.01e-13 ***
## Xrad         3.060e-01  6.635e-02   4.613 5.07e-06 ***
## Xtax        -1.233e-02  3.760e-03  -3.280 0.001112 **
## Xptratio    -9.527e-01  1.308e-01  -7.283 1.31e-12 ***
## Xblack       9.312e-03  2.686e-03   3.467 0.000573 ***
## Xlstat      -5.248e-01  5.072e-02 -10.347  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 4.745 on 492 degrees of freedom
## Multiple R-squared:  0.7406, Adjusted R-squared:  0.7338
## F-statistic: 108.1 on 13 and 492 DF,  p-value: < 2.2e-16
```

```
summary(mod2)
```

```
##
## Call:
## lm(formula = y ~ eigenvecs_H[, 1:14])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.5885  -2.7528  -0.5003   1.6738  26.1135
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)             2.874e+08  3.403e+08   0.845    0.399
## eigenvecs_H[, 1:14]1    1.238e+08  1.466e+08   0.845    0.399
## eigenvecs_H[, 1:14]2   -2.909e+08  3.444e+08  -0.845    0.399
## eigenvecs_H[, 1:14]3   -6.559e+09  7.766e+09  -0.845    0.399
## eigenvecs_H[, 1:14]4    6.044e+08  7.156e+08   0.845    0.399
## eigenvecs_H[, 1:14]5    9.666e+07  1.144e+08   0.845    0.399
## eigenvecs_H[, 1:14]6   -3.000e+08  3.551e+08  -0.845    0.399
## eigenvecs_H[, 1:14]7    2.716e+08  3.216e+08   0.845    0.399
## eigenvecs_H[, 1:14]8    1.631e+07  1.931e+07   0.845    0.399
## eigenvecs_H[, 1:14]9   -9.690e+08  1.147e+09  -0.845    0.399
## eigenvecs_H[, 1:14]10  -8.130e+06  9.626e+06  -0.845    0.399
## eigenvecs_H[, 1:14]11  -2.030e+08  2.404e+08  -0.845    0.399
## eigenvecs_H[, 1:14]12   1.552e+08  1.838e+08   0.845    0.399
## eigenvecs_H[, 1:14]13   3.570e+08  4.227e+08   0.845    0.399
## eigenvecs_H[, 1:14]14  -3.189e+07  3.776e+07  -0.845    0.399
##
## Residual standard error: 4.747 on 491 degrees of freedom
## Multiple R-squared:  0.741,  Adjusted R-squared:  0.7336
## F-statistic: 100.3 on 14 and 491 DF,  p-value: < 2.2e-16
```

Is b about the same above (in arbitrary order)?

NO, the eigen vectors are scaled to be unit length

Calculate $\hat{y}$ using the hat matrix.

```
y_hat= H %*% y
```

Calculate e two ways: (1) the difference of y and $\hat{y}$ and (2) the projection onto the space of the residuals. Verify the two means of calculating the residuals provide the same results via `expect_equal`.

```
e1 = y - y_hat
e2 = Hcomp %*% y
expect_equal(e1, e2)
```

Calculate R^2 using the angle relationship between the responses and their predictions. $

```
length_of_vec = function(v){sqrt(sum(v^2))}
y_avg_adj = y - mean(y)
y_yhat_adj = y_hat - mean(y)
(sum(y_avg_adj *
```

```
    y_yhat_adj
    )
   /
   (length_of_vec(y_avg_adj) *
     length_of_vec(y_yhat_adj))
   ) ** 2
```

```
## [1] 0.7406427
```

```
(
  sum(
    y_avg_adj * y_yhat_adj
  )
  /
  (
    length_of_vec(y_avg_adj) * length_of_vec(y_yhat_adj))) ^ 2
```

```
## [1] 0.7406427
```

Find the cosine-squared of $y - \bar{y}$ and $\hat{y} - \bar{y}$ and verify it is the same as R^2.

```
summary(mod1)$r.squared
```

```
## [1] 0.7406427
```

```
theta_in_rad = cos(
               ((y_avg_adj) %*% y_yhat_adj)
               /
               (length_of_vec(y_avg_adj) * length_of_vec(y_yhat_adj))
             )
(theta_in_rad * 180 / pi)
```

```
##          [,1]
## [1,] 37.35559
```

```
theta_in_rad ^ 2
```

```
##           [,1]
## [1,] 0.4250755
```

Verify $\hat{y}$ and e are orthogonal.

```
sum(y_hat*e1)
```

```
## [1] -4.991219e-08
```

Verify $\hat{y} - \bar{y}$ and e are orthogonal.

```
sum((y_hat -mean(y)) *e1)
```

```
## [1] 2.832455e-09
```

Verify the sum of squares identity which we learned was due to the Pythagorean Theorem (applies since the projection is specifically orthogonal). You need to compute all three quantities first: SST, SSR and SSE.

```
SSE = sum((y - y_hat)^2)
SST = sum((y - mean(y))^2)
SSR = sum((y_hat - mean(y))^2)
1 - (SSE / SST)
```

```
## [1] 0.7406427
```

```r
expect_equal(SSR/SST,1 - (SSE / SST), (SST-SSE) / SST)
a = sqrt(SSR)
b = sqrt(SSE)
c = sqrt(SST)
expect_equal(c^2,a^2 + b^2)
```

Create a matrix that is $(p+1) \times (p+1)$ full of NA's. Label the columns the same columns as X. Do not label the rows. For the first row, find the OLS estimate of the y regressed on the first column only and put that in the first entry. For the second row, find the OLS estimates of the y regressed on the first and second columns of X only and put them in the first and second entries. For the third row, find the OLS estimates of the y regressed on the first, second and third columns of X only and put them in the first, second and third entries, etc. For the last row, fill it with the full OLS estimates.

```r
new_matrix= matrix(NA,ncol(X),ncol(X))
dim(new_matrix)
```

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

```r
colnames(new_matrix) = colnames(X)

# for(i in 1:ncol(new_matrix)){
#   matrix_col_i = X[ , 1:i, drop = FALSE] # entire column
#   new_matrix[i, 1:i] = solve(t(matrix_col_i) %*% matrix_col_i ) %*% t(matrix_col_i) %*% y  #regressin
# }

for(i in 1:ncol(new_matrix)){
  matrix_col_i = X[ , 1:i, drop = FALSE]
  for(j in 1:i){
    b = solve(t(matrix_col_i) %*% matrix_col_i ) %*% t(matrix_col_i) %*% y
    new_matrix[i, 1:i] = b
  }
}
#new_matrix
round(new_matrix,2) # easier to view
```

```
##                crim    zn indus chas     nox   rm   age   dis   rad   tax ptratio
##   [1,]  22.53    NA    NA    NA   NA      NA   NA    NA    NA    NA    NA      NA
##   [2,]  24.03 -0.42    NA    NA   NA      NA   NA    NA    NA    NA    NA      NA
##   [3,]  22.49 -0.35 0.12    NA   NA      NA   NA    NA    NA    NA    NA      NA
##   [4,]  27.39 -0.25 0.06 -0.42   NA      NA   NA    NA    NA    NA    NA      NA
##   [5,]  27.11 -0.23 0.06 -0.44 6.89      NA   NA    NA    NA    NA    NA      NA
##   [6,]  29.49 -0.22 0.06 -0.38 7.03   -5.42   NA    NA    NA    NA    NA      NA
##   [7,] -17.95 -0.18 0.02 -0.14 4.78   -7.18 7.34    NA    NA    NA    NA      NA
##   [8,] -18.26 -0.17 0.01 -0.13 4.84   -4.36 7.39 -0.02    NA    NA    NA      NA
##   [9,]   0.83 -0.20 0.06 -0.23 4.58  -14.45 6.75 -0.06 -1.76    NA    NA      NA
##  [10,]   0.16 -0.18 0.06 -0.21 4.54  -13.34 6.79 -0.06 -1.75 -0.05    NA      NA
##  [11,]   2.99 -0.18 0.07 -0.10 4.11  -12.59 6.66 -0.05 -1.73  0.16 -0.01      NA
##  [12,]  27.15 -0.18 0.04 -0.04 3.49  -22.18 6.08 -0.05 -1.58  0.25 -0.01   -1.00
##  [13,]  20.65 -0.16 0.04 -0.03 3.22  -20.48 6.12 -0.05 -1.55  0.28 -0.01   -1.01
##  [14,]  36.46 -0.11 0.05  0.02 2.69  -17.77 3.81  0.00 -1.48  0.31 -0.01   -0.95
##        black lstat
##   [1,]    NA    NA
##   [2,]    NA    NA
##   [3,]    NA    NA
##   [4,]    NA    NA
```

```
## [5,]    NA    NA
## [6,]    NA    NA
## [7,]    NA    NA
## [8,]    NA    NA
## [9,]    NA    NA
## [10,]   NA    NA
## [11,]   NA    NA
## [12,]   NA    NA
## [13,] 0.01    NA
## [14,] 0.01 -0.52
```

Examine this matrix. Why are the estimates changing from row to row as you add in more predictors

We are examining one feature at a time and seeing how far its away from the mean.

Clear the workspace and load the diamonds dataset in the package `ggplot2`.

```
rm(list=ls())
pacman::p_load(ggplot2)
data("diamonds")
```

Extract y, the price variable and `col`, the nominal variable "color" as vectors.

```
y = diamonds$price
col = diamonds$color
```

Convert the `col` vector to X which contains an intercept and an appropriate number of dummies. Let the color G be the refernce category as it is the modal color. Name the columns of X appropriately. The first should be "(Intercept)". Delete `col`.

```
#Problem col is ordered factor cant just use relevel
# col = droplevels(col,"G")
# X = cbind(1,col)
#
# X
X = matrix(1,nrow(diamonds)) #


for(lev in levels(col)){
  if(lev != "G"){
    X = cbind(X, col == lev)
  }
}
colnames(X) = c("Intercept","D","E","F","H","I","J")
```

Repeat the iterative exercise above we did for Boston here.

```
new_matrix= matrix(NA,ncol(X),ncol(X))
colnames(new_matrix) = colnames(X)
for(i in 1:ncol(new_matrix)){
  matrix_col_i = X[ , 1:i, drop = FALSE]
  for(j in 1:i){
    b = solve(t(matrix_col_i) %*% matrix_col_i ) %*% t(matrix_col_i) %*% y
    new_matrix[i, 1:i] = b
  }
}

price_model = lm(price ~ color, diamonds)
```

```r
round(new_matrix,2)
```

```
##      Intercept        D        E       F      H       I       J
## [1,]   3932.80       NA       NA      NA     NA      NA      NA
## [2,]   4042.38  -872.42       NA      NA     NA      NA      NA
## [3,]   4295.54 -1125.59 -1218.79      NA     NA      NA      NA
## [4,]   4491.23 -1321.28 -1414.48 -766.34     NA      NA      NA
## [5,]   4493.17 -1323.22 -1416.42 -768.28  -6.50      NA      NA
## [6,]   4262.94 -1092.99 -1186.19 -538.06 223.72  828.93      NA
## [7,]   3999.14  -829.18  -922.38 -274.25 487.53 1092.74 1324.68
```

```r
summary(price_model)
```

```
## 
## Call:
## lm(formula = price ~ color, data = diamonds)
## 
## Residuals:
##    Min     1Q Median     3Q    Max 
##  -4989  -2619  -1376   1374  15654 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept)  4124.73      18.64 221.294  < 2e-16 ***
## color.L      2126.73      57.02  37.295  < 2e-16 ***
## color.Q       200.50      54.26   3.695  0.00022 ***
## color.C      -254.36      51.08  -4.979 6.41e-07 ***
## color^4        40.88      46.92   0.871  0.38361    
## color^5      -228.88      44.36  -5.160 2.48e-07 ***
## color^6        87.92      40.22   2.186  0.02880 *  
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 3927 on 53933 degrees of freedom
## Multiple R-squared:  0.03128,    Adjusted R-squared:  0.03117 
## F-statistic: 290.2 on 6 and 53933 DF,  p-value: < 2.2e-16
```

Why didn't the estimates change as we added more and more features?

The estimates did change we we added more featues.

Model `price` with both `color` and `clarity` with and without an intercept and report the coefficients.

```r
with_inter_diamond_price_model = lm(price ~ color + clarity, diamonds)

without_inter_diamond_price_model = lm(price ~ 0 +color + clarity, diamonds)
summary(with_inter_diamond_price_model)
```

```
## 
## Call:
## lm(formula = price ~ color + clarity, data = diamonds)
## 
## Residuals:
##    Min     1Q Median     3Q    Max 
##  -6046  -2469  -1308   1164  16858 
## 
## Coefficients:
```

```
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3813.01      26.89 141.775  < 2e-16 ***
## color.L      2137.54      56.34  37.941  < 2e-16 ***
## color.Q        16.27      53.83   0.302 0.762406
## color.C      -221.70      50.39  -4.400 1.09e-05 ***
## color^4        77.36      46.28   1.671 0.094668 .
## color^5      -259.95      43.73  -5.945 2.78e-09 ***
## color^6         2.12      39.76   0.053 0.957479
## clarity.L   -1718.62      97.41 -17.642  < 2e-16 ***
## clarity.Q    -594.15      95.30  -6.235 4.56e-10 ***
## clarity.C     681.36      81.97   8.313  < 2e-16 ***
## clarity^4    -248.60      65.71  -3.783 0.000155 ***
## clarity^5     806.88      53.76  15.010  < 2e-16 ***
## clarity^6    -228.71      46.90  -4.876 1.08e-06 ***
## clarity^7     191.11      41.40   4.616 3.92e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3870 on 53926 degrees of freedom
## Multiple R-squared:  0.05937,    Adjusted R-squared:  0.05915
## F-statistic: 261.8 on 13 and 53926 DF,  p-value: < 2.2e-16
```

```r
summary(without_inter_diamond_price_model)
```

```
##
## Call:
## lm(formula = price ~ 0 + color + clarity, data = diamonds)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##  -6046  -2469  -1308   1164  16858
##
## Coefficients:
##           Estimate Std. Error t value Pr(>|t|)
## colorD     2747.66      52.12  52.722  < 2e-16 ***
## colorE     2757.08      44.20  62.375  < 2e-16 ***
## colorF     3462.31      43.77  79.097  < 2e-16 ***
## colorG     3841.91      40.49  94.887  < 2e-16 ***
## colorH     4167.61      46.44  89.734  < 2e-16 ***
## colorI     4780.83      56.10  85.225  < 2e-16 ***
## colorJ     4933.65      76.03  64.890  < 2e-16 ***
## clarity.L -1718.62      97.41 -17.642  < 2e-16 ***
## clarity.Q  -594.15      95.30  -6.235 4.56e-10 ***
## clarity.C   681.36      81.96   8.313  < 2e-16 ***
## clarity^4  -248.60      65.71  -3.783 0.000155 ***
## clarity^5   806.88      53.76  15.010  < 2e-16 ***
## clarity^6  -228.71      46.90  -4.876 1.08e-06 ***
## clarity^7   191.11      41.40   4.616 3.92e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3870 on 53926 degrees of freedom
## Multiple R-squared:  0.523,  Adjusted R-squared:  0.5228
## F-statistic:  4223 on 14 and 53926 DF,  p-value: < 2.2e-16
```

Which coefficients did not change between the models and why?

The clarity since they are numeric values.

Create a 2x2 matrix with the first column 1's and the next column iid normals. Find the absolute value of the angle (in degrees, not radians) between the two columns.

```
matrix_two = matrix(NA,2,2)
matrix_two[,1] = 1
matrix_two[,2] = rnorm(1)
theta_in_rad = acos(matrix_two[,1]  %*% matrix_two[,2]  / sqrt(sum(matrix_two[,1] ^2) * sum(matrix_two[
theta_in_rad * 180 / pi #Getting 0 or 180
```

```
##      [,1]
## [1,]  180
```

Repeat this exercise $Nsim = 1e5$ times and report the average absolute angle.

```
theta_sum = 0
for(i in 1:1e5){
  theta_sum = theta_sum + acos(matrix_two[,1]  %*% matrix_two[,2]  / sqrt(sum(matrix_two[,1] ^2) * sum(m
}
(theta_sum/1e5)* 180 / pi
```

```
##      [,1]
## [1,]  180
```

Create a 2xn matrix with the first column 1's and the next column iid normals. Find the absolute value of the angle (in degrees, not radians) between the two columns. For $n \in 10, 50, 100, 200, 500, 1000$, report the average absolute angle over $Nsim = 1e5$ simulations.

```
matrix_three = matrix(NA,2,10)
matrix_three[,1] = 1
matrix_three[,1:10] = rnorm(10)
theta_in_rad = acos(matrix_three[,1]  %*% matrix_three[,2]  / sqrt(sum(matrix_three[,1] ^2) * sum(matri
theta_in_rad * 180 / pi
```

```
##          [,1]
## [1,] 96.79672
```

```
for(i in 1:1e5){
  theta_sum = theta_sum + acos(matrix_three[,1]  %*% matrix_three[,2]  / sqrt(sum(matrix_three[,1] ^2) *
}
(theta_sum/1e5)* 180 / pi
```

```
##          [,1]
## [1,] 276.7967
```

What is this absolute angle converging to? Why does this make sense?

#TO-DO

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.

```
y = rnorm(100)
X = matrix(NA,100,2)
X[,1] = 1
X[,2] = rnorm(100)
```

```r
b = solve(t(X) %*% X ) %*% t(X) %*% y
y_hat = X %*% b
summary(y_hat)
```

```
##        V1
##  Min.   :-0.06150
##  1st Qu.: 0.02066
##  Median : 0.04091
##  Mean   : 0.04500
##  3rd Qu.: 0.07404
##  Max.   : 0.13723
```

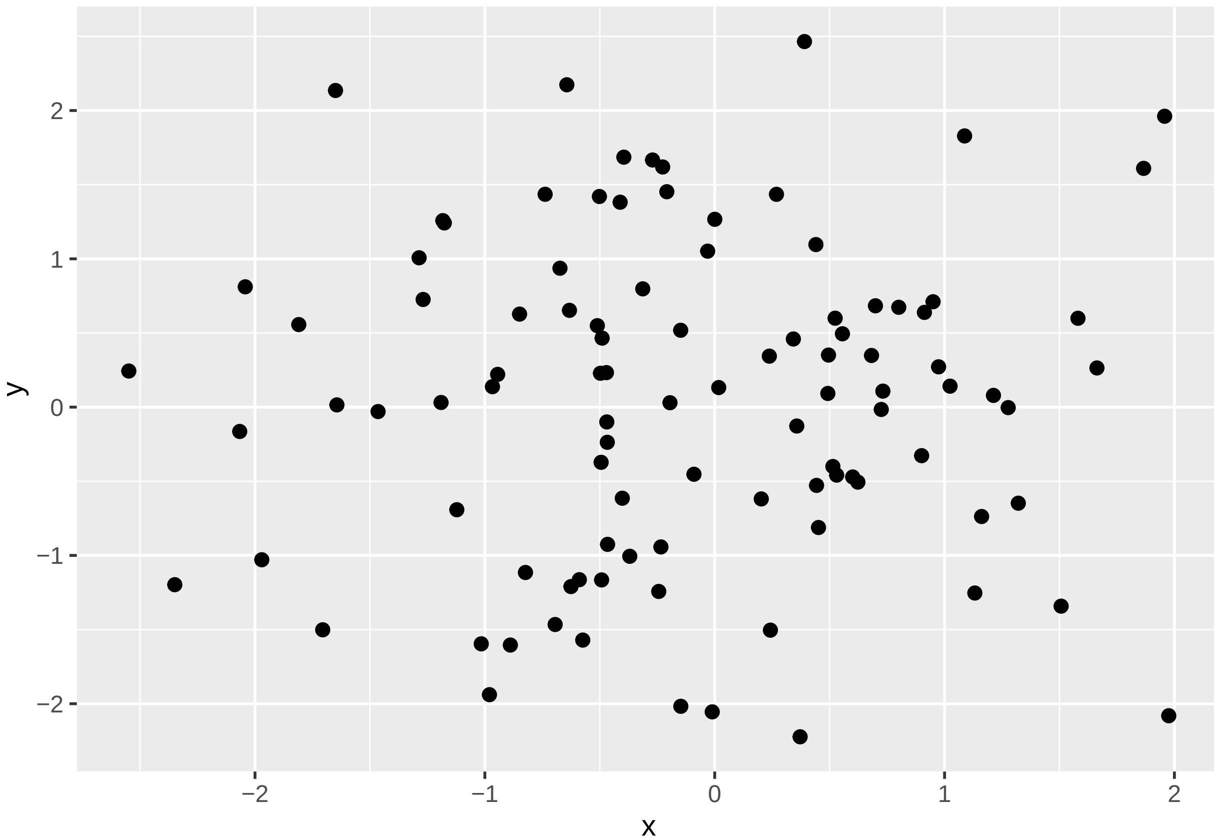
```r
SSE = sum((y - y_hat)^2)
SST = sum((y - mean(y))^2)
SSR = sum((y_hat - mean(y))^2)
1 - (SSE / SST)
```

```
## [1] 0.00161014
```

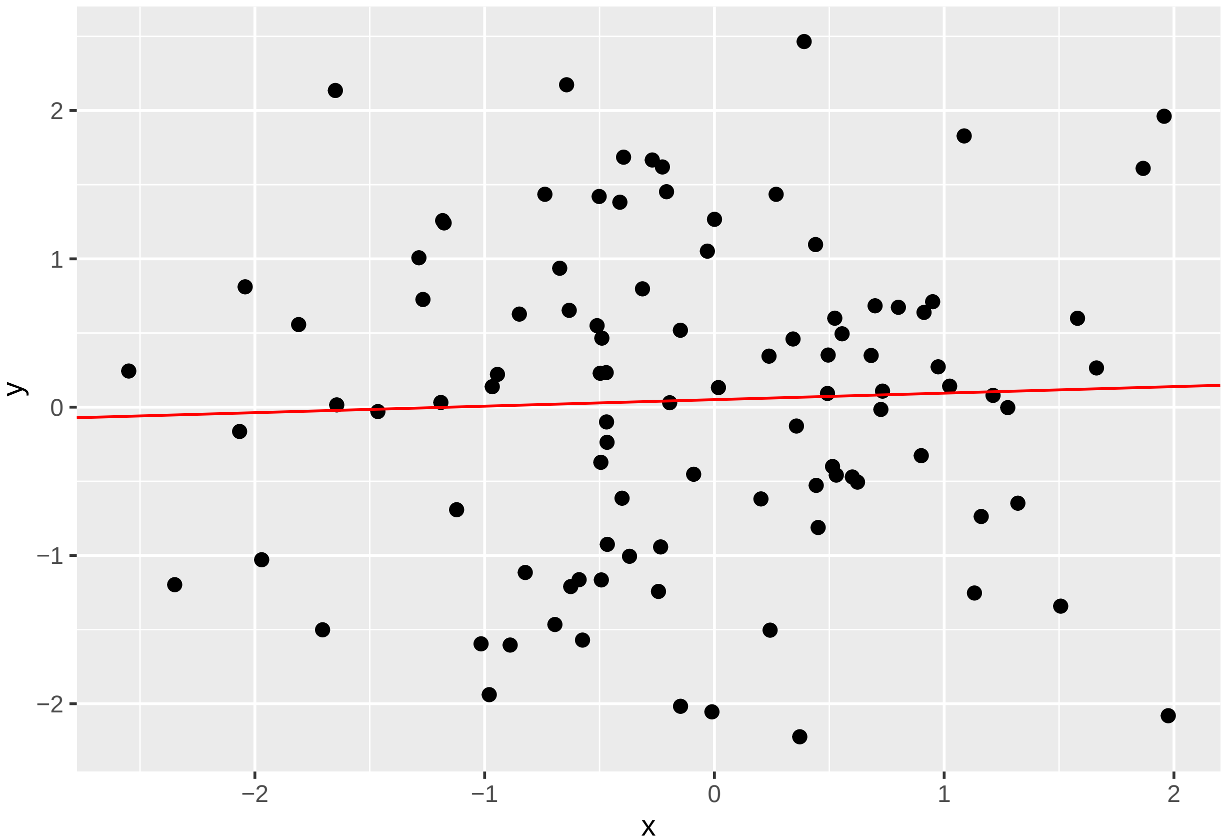
```r
model = lm(y ~ X)
summary(model)
```

```
##
## Call:
## lm(formula = y ~ X)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.2897 -0.7769  0.0414  0.6215  2.3976
##
## Coefficients: (1 not defined because of singularities)
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.05046    0.10935   0.461    0.646
## X1                NA         NA      NA       NA
## X2           0.04392    0.11049   0.398    0.692
##
## Residual standard error: 1.085 on 98 degrees of freedom
## Multiple R-squared:  0.00161,    Adjusted R-squared:  -0.008578
## F-statistic: 0.158 on 1 and 98 DF,  p-value: 0.6918
```

```r
simple_df = data.frame(x = X[,2], y = y)
simple_viz_obj = ggplot(simple_df, aes(x, y)) +
  geom_point(size = 2)
simple_viz_obj
```




```
b_0 = model$coefficients[1]
b_1 = model$coefficients[3]
simple_ls_regression_line = geom_abline(intercept = b_0, slope = b_1, color = "red")
simple_viz_obj + simple_ls_regression_line
```



Write a for loop to each time bind a new column of 100 standard iid normals to the matrix X and find the R^2 each time until the number of columns is 100. Create a vector to save all R^2's. What happened??

```
X = matrix(1,100,1)
for(i in 2:100){
  X = cbind(X,rnorm(100))
}
b = solve(t(X) %*% X ) %*% t(X) %*% y
y_hat = X %*% b

SSE = sum((y - y_hat)^2)
SST = sum((y - mean(y))^2)
SSR = sum((y_hat - mean(y))^2)
1 - (SSE / SST) # It becomes just 100%
```

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

```
dim(X)
```

```
## [1] 100 100
```

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

```
y = rnorm(101)
X = matrix(1,101,1)
for(i in 1:100){
  X = cbind(X,rnorm(100))
}
b = solve(t(X) %*% X ) %*% t(X) %*% y
```

```
## Error in solve.default(t(X) %*% X): system is computationally singular: reciprocal condition number =
```

```r
y_hat = X %*% b
```

```
## Error in X %*% b: non-conformable arguments
```

```r
SSE = sum((y - y_hat)^2)
```

```
## Error in eval(expr, envir, enclos): dims [product 100] do not match the length of object [101]
```

```r
SST = sum((y - mean(y))^2)
SSR = sum((y_hat - mean(y))^2)
dim(X)
```

```
## [1] 101 101
```

```r
1 - (SSE / SST) # It becomes just 100%
```

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