Lab 5

Adriana Sham

11:59PM March 17, 2019

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)".

```r
data(Boston, package = "MASS")
y = Boston$medv
X = as.matrix(cbind(1, Boston[, 1 : 13]))
colnames(X)[1] = "(Intercept)"
```

Run the OLS linear model to get b, the vector of coefficients. Do not use `lm`.

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

Find the hat matrix for this regression `H` and find its rank. Is this rank expected?

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

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

```r
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)`.

```r
pacman::p_load(testthat)
expect_equal(H, t(H), tolerance = 1e-2)
expect_equal(H %*% H, H, tolerance = 1e-2)
```

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

```r
I = diag(nrow(H))
H_comp = (I - H)
rankMatrix(H_comp)
```

```
## [1] 497
## 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.

```r
expect_equal(H_comp, t(H_comp), tolerance = 1e-2)
expect_equal(H_comp %*% H_comp, H_comp, tolerance = 1e-2)
```

Calculate $\hat{y}$.

```r
yhat = H %*% y
yhat
```

```
##            [,1]
## 1   30.0038434
## 2   25.0255624
## 3   30.5675967
## 4   28.6070365
## 5   27.9435242
## 6   25.2562845
## 7   23.0018083
## 8   19.5359884
## 9   11.5236369
## 10  18.9202621
## 11  18.9994965
## 12  21.5867957
## 13  20.9065215
## 14  19.5529028
## 15  19.2834821
## 16  19.2974832
## 17  20.5275098
## 18  16.9114013
## 19  16.1780111
## 20  18.4061360
## 21  12.5238575
## 22  17.6710367
## 23  15.8328813
## 24  13.8062853
## 25  15.6783383
## 26  13.3866856
## 27  15.4639765
## 28  14.7084743
## 29  19.5473729
## 30  20.8764282
## 31  11.4551176
## 32  18.0592329
## 33   8.8110574
## 34  14.2827581
## 35  13.7067589
## 36  23.8146353
## 37  22.3419371
## 38  23.1089114
## 39  22.9150261
## 40  31.3576257
## 41  34.2151023
## 42  28.0205641
## 43  25.2038663
## 44  24.6097927
## 45  22.9414918
## 46  22.0966982
## 47  20.4232003
## 48  18.0365509
## 49   9.1065538
## 50  17.2060775
## 51  21.2815254
## 52  23.9722228
## 53  27.6558508
## 54  24.0490181
## 55  15.3618477
## 56  31.1526495
## 57  24.8568698
## 58  33.1091981
## 59  21.7753799
## 60  21.0849356
## 61  17.8725804
## 62  18.5111021
## 63  23.9874286
## 64  22.5540887
## 65  23.3730864
## 66  30.3614836
## 67  25.5305651
## 68  21.1133856
## 69  17.4215379
## 70  20.7848363
## 71  25.2014886
## 72  21.7426577
## 73  24.5574496
## 74  24.0429571
## 75  25.5049972
## 76  23.9669302
## 77  22.9454540
## 78  23.3569982
## 79  21.2619827
## 80  22.4281737
## 81  28.4057697
## 82  26.9948609
## 83  26.0357630
## 84  25.0587348
## 85  24.7845667
## 86  27.7904920
## 87  22.1685342
## 88  25.8927642
## 89  30.6746183
## 90  30.8311062
## 91  27.1190194
## 92  27.4126673
## 93  28.9412276
## 94  29.0810555
## 95  27.0397736
## 96  28.6245995
## 97  24.7274498
## 98  35.7815952
## 99  35.1145459
## 100 32.2510280
## 101 24.5802202
## 102 25.5941347
## 103 19.7901368
## 104 20.3116713
## 105 21.4348259
## 106 18.5399401
## 107 17.1875599
## 108 20.7504903
## 109 22.6482911
## 110 19.7720367
## 111 20.6496586
## 112 26.5258674
## 113 20.7732364
## 114 20.7154831
## 115 25.1720888
## 116 20.4302559
## 117 23.3772463
## 118 23.6904326
## 119 20.3357836
## 120 20.7918087
## 121 21.9163207
## 122 22.4710778
## 123 20.5573856
## 124 16.3666198
## 125 20.5609982
## 126 22.4817845
## 127 14.6170663
## 128 15.1787668
## 129 18.9386859
## 130 14.0557329
## 131 20.0352740
## 132 19.4101340
## 133 20.0619157
## 134 15.7580767
## 135 13.2564524
## 136 17.2627773
## 137 15.8784188
## 138 19.3616395
## 139 13.8148390
## 140 16.4488147
## 141 13.5714193
## 142  3.9888551
## 143 14.5949548
## 144 12.1488148
## 145  8.7282236
## 146 12.0358534
## 147 15.8208206
## 148  8.5149902
## 149  9.7184414
## 150 14.8045137
## 151 20.8385815
## 152 18.3010117
## 153 20.1228256
## 154 17.2860189
## 155 22.3660023
## 156 20.1037592
## 157 13.6212589
## 158 33.2598270
## 159 29.0301727
## 160 25.5675277
## 161 32.7082767
## 162 36.7746701
## 163 40.5576584
## 164 41.8472817
## 165 24.7886738
## 166 25.3788924
## 167 37.2034745
## 168 23.0874875
## 169 26.4027396
## 170 26.6538211
## 171 22.5551466
## 172 24.2908281
## 173 22.9765722
## 174 29.0719431
## 175 26.5219434
## 176 30.7220906
## 177 25.6166931
## 178 29.1374098
## 179 31.4357197
## 180 32.9223157
## 181 34.7244046
## 182 27.7655211
## 183 33.8878732
## 184 30.9923804
## 185 22.7182001
## 186 24.7664781
## 187 35.8849723
## 188 33.4247672
## 189 32.4119915
## 190 34.5150995
## 191 30.7610949
## 192 30.2893414
## 193 32.9191871
## 194 32.1126077
## 195 31.5587100
## 196 40.8455572
## 197 36.1277008
## 198 32.6692081
## 199 34.7046912
## 200 30.0934516
## 201 30.6439391
## 202 29.2871950
## 203 37.0714839
## 204 42.0319312
## 205 43.1894984
## 206 22.6903480
## 207 23.6828471
## 208 17.8544721
## 209 23.4942899
## 210 17.0058772
## 211 22.3925110
## 212 17.0604275
## 213 22.7389292
## 214 25.2194255
## 215 11.1191674
## 216 24.5104915
## 217 26.6033477
## 218 28.3551871
## 219 24.9152546
## 220 29.6865277
## 221 33.1841975
## 222 23.7745666
## 223 32.1405196
## 224 29.7458199
## 225 38.3710245
## 226 39.8146187
## 227 37.5860575
## 228 32.3995325
## 229 35.4566524
## 230 31.2341151
## 231 24.4844923
## 232 33.2883729
## 233 38.0481048
## 234 37.1632863
## 235 31.7138352
## 236 25.2670557
## 237 30.1001074
## 238 32.7198716
## 239 28.4271706
## 240 28.4294068
## 241 27.2937594
## 242 23.7426248
## 243 24.1200789
## 244 27.4020841
## 245 16.3285756
## 246 13.3989126
## 247 20.0163878
## 248 19.8618443
## 249 21.2883131
## 250 24.0798915
## 251 24.2063355
## 252 25.0421582
## 253 24.9196401
## 254 29.9456337
## 255 23.9722832
## 256 21.6958089
## 257 37.5110924
## 258 43.3023904
## 259 36.4836142
## 260 34.9898859
## 261 34.8121151
## 262 37.1663133
## 263 40.9892850
## 264 34.4463409
## 265 35.8339755
## 266 28.2457430
## 267 31.2267359
## 268 40.8395575
## 269 39.3179239
## 270 25.7081791
## 271 22.3029553
## 272 27.2034097
## 273 28.5116947
## 274 35.4767660
## 275 36.1063916
## 276 33.7966827
## 277 35.6108586
## 278 34.8399338
## 279 30.3519266
## 280 35.3098070
## 281 38.7975697
## 282 34.3312319
## 283 40.3396307
## 284 44.6730834
## 285 31.5968909
## 286 27.3565923
## 287 20.1017415
## 288 27.0420667
## 289 27.2136458
## 290 26.9139584
## 291 33.4356331
## 292 34.4034963
## 293 31.8333982
## 294 25.8178324
## 295 24.4298235
## 296 28.4576434
## 297 27.3626700
## 298 19.5392876
## 299 29.1130984
## 300 31.9105461
## 301 30.7715945
## 302 28.9427587
## 303 28.8819102
## 304 32.7988723
## 305 33.2090546
## 306 30.7683179
## 307 35.5622686
## 308 32.7090512
## 309 28.6424424
## 310 23.5896583
## 311 18.5426690
## 312 26.8788984
## 313 23.2813398
## 314 25.5458025
## 315 25.4812006
## 316 20.5390990
## 317 17.6157257
## 318 18.3758169
## 319 24.2907028
## 320 21.3252904
## 321 24.8868224
## 322 24.8693728
## 323 22.8695245
## 324 19.4512379
## 325 25.1178340
## 326 24.6678691
## 327 23.6807618
## 328 19.3408962
## 329 21.1741811
## 330 24.2524907
## 331 21.5926089
## 332 19.9844661
## 333 23.3388800
## 334 22.1406069
## 335 21.5550993
## 336 20.6187291
## 337 20.1609718
## 338 19.2849039
## 339 22.1667232
## 340 21.2496577
## 341 21.4293931
## 342 30.3278880
## 343 22.0473498
## 344 27.7064791
## 345 28.5479412
## 346 16.5450112
## 347 14.7835964
## 348 25.2738008
## 349 27.5420512
## 350 22.1483756
## 351 20.4594409
## 352 20.5460542
## 353 16.8806383
## 354 25.4025351
## 355 14.3248663
## 356 16.5948846
## 357 19.6370469
## 358 22.7180661
## 359 22.2021889
## 360 19.2054806
## 361 22.6661611
## 362 18.9319262
## 363 18.2284680
## 364 20.2315081
## 365 37.4944739
## 366 14.2819073
## 367 15.5428625
## 368 10.8316232
## 369 23.8007290
## 370 32.6440736
## 371 34.6068404
## 372 24.9433133
## 373 25.9998091
## 374  6.1263250
## 375  0.7777981
## 376 25.3071306
## 377 17.7406106
## 378 20.2327441
## 379 15.8333130
## 380 16.8351259
## 381 14.3699483
## 382 18.4768283
## 383 13.4276828
## 384 13.0617751
## 385  3.2791812
## 386  8.0602217
## 387  6.1284220
## 388  5.6186481
## 389  6.4519857
## 390 14.2076474
## 391 17.2122518
## 392 17.2988727
## 393  9.8911664
## 394 20.2212419
## 395 17.9418118
## 396 20.3044578
## 397 19.2955908
## 398 16.3363278
## 399  6.5516232
## 400 10.8901678
## 401 11.8814587
## 402 17.8117451
## 403 18.2612659
## 404 12.9794878
## 405  7.3781636
## 406  8.2111586
## 407  8.0662619
## 408 19.9829479
## 409 13.7075637
## 410 19.8526845
## 411 15.2230830
## 412 16.9607198
## 413  1.7185181
## 414 11.8057839
## 415 -4.2813107
## 416  9.5837674
## 417 13.3666081
## 418  6.8956236
## 419  6.1477985
## 420 14.6066179
## 421 19.6000267
## 422 18.1242748
## 423 18.5217713
## 424 13.1752861
## 425 14.6261762
## 426  9.9237498
## 427 16.3459065
## 428 14.0751943
## 429 14.2575624
## 430 13.0423479
## 431 18.1595569
## 432 18.6955435
## 433 21.5272830
## 434 17.0314186
## 435 15.9609044
## 436 13.3614161
## 437 14.5207938
## 438  8.8197601
## 439  4.8675110
## 440 13.0659131
## 441 12.7060970
## 442 17.2955806
## 443 18.7404850
## 444 18.0590103
## 445 11.5147468
## 446 11.9740036
## 447 17.6834462
## 448 18.1269524
## 449 17.5183465
## 450 17.2274251
## 451 16.5227163
## 452 19.4129110
## 453 18.5821524
## 454 22.4894479
## 455 15.2800013
## 456 15.8208934
## 457 12.6872558
## 458 12.8763379
## 459 17.1866853
## 460 18.5124761
## 461 19.0486053
## 462 20.1720893
## 463 19.7740732
## 464 22.4294077
## 465 20.3191185
## 466 17.8861625
## 467 14.3747852
## 468 16.9477685
## 469 16.9840576
## 470 18.5883840
## 471 20.1671944
## 472 22.9771803
## 473 22.4558073
## 474 25.5782463
## 475 16.3914763
## 476 16.1114628
## 477 20.5348160
## 478 11.5427274
## 479 19.2049630
## 480 21.8627639
## 481 23.4687887
## 482 27.0988732
## 483 28.5699430
## 484 21.0839878
## 485 19.4551620
## 486 22.2222591
## 487 19.6559196
## 488 21.3253610
## 489 11.8558372
## 490  8.2238669
## 491  3.6639967
## 492 13.7590854
## 493 15.9311855
## 494 20.6266205
## 495 20.6124941
## 496 16.8854196
## 497 14.0132079
## 498 19.1085414
## 499 21.2980517
## 500 18.4549884
## 501 20.4687085
## 502 23.5333405
## 503 22.3757189
## 504 27.6274261
## 505 26.1279668
## 506 22.3442123
```

Calculate e as the difference of y and $\hat{y}$ and the projection onto the space of the residuals. Verify the two means of calculating the residuals provide the same results.

```
e = y - yhat
e_2 = H_comp %*% y
expect_equal(e, e_2)
```

Calculate R^2 and RMSE.

```
sse = sum(e^2)
sst = sum((y - mean(y))^2)

Rsquared = 1 - sse / sst
Rsquared
```

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

```
mse = sse / (nrow(X) - ncol(X))
rmse = sqrt(mse) #rmse is standard deviation of errors
rmse
```

```
## [1] 4.745298
```

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

```
t(e) %*% yhat
```

```
##                  [,1]
## [1,] -4.991142e-08
```

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

```
t(e) %*% (yhat - mean(y))
```

```
##               [,1]
## [1,] 2.832162e-09
```

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

```
y_minus_y_bar = y - mean(y)
yhat_minus_y_bar = yhat - mean(y)
len_y_minus_y_bar = sqrt(  sum(y_minus_y_bar^2)  )
len_yhat_minus_y_bar = sqrt(  sum(yhat_minus_y_bar^2)  )

theta = acos( (t(y_minus_y_bar) %*% yhat_minus_y_bar) / (len_y_minus_y_bar * len_yhat_minus_y_bar) )
cos(theta * (180 / pi))
```

```
##           [,1]
## [1,] 0.6962634
```

```
cos_theta_sqrd = cos(theta)^2
cos_theta_sqrd
```

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

Verify the sum of squares identity which we learned was due to the Pythagorean Theorem (applies since the projection is specifically orthogonal).

```
len_y_minus_y_bar^2 - len_yhat_minus_y_bar^2 - sse
```

```
## [1] 5.666152e-09
```

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.

```
M = matrix(NA, nrow = ncol(X), ncol = ncol(X))
colnames(M) = colnames(X)
X_j = X[, 1, drop = FALSE]
b = solve(t(X_j) %*% X_j) %*% t(X_j) %*% y
M[1, 1] = b
X_j_2 = X[ , 1:2]
b = solve(t(X_j_2) %*% X_j_2) %*% t(X_j_2) %*% y
b
```

```
##                   [,1]
## (Intercept) 24.0331062
## crim        -0.4151903
```

```
for(j in 1 : ncol(M)){
  X_j = X[, 1 : j, drop = FALSE]
  b = solve(t(X_j) %*% X_j) %*% t(X_j) %*% y
  M[j, 1:j] = b
}
round(M, 2)
```

```
##       (Intercept)  crim   zn indus chas    nox   rm   age   dis   rad
##  [1,]       22.53    NA   NA    NA   NA     NA   NA    NA    NA    NA
##  [2,]       24.03 -0.42   NA    NA   NA     NA   NA    NA    NA    NA
##  [3,]       22.49 -0.35 0.12    NA   NA     NA   NA    NA    NA    NA
##  [4,]       27.39 -0.25 0.06 -0.42   NA     NA   NA    NA    NA    NA
##  [5,]       27.11 -0.23 0.06 -0.44 6.89     NA   NA    NA    NA    NA
##  [6,]       29.49 -0.22 0.06 -0.38 7.03  -5.42   NA    NA    NA    NA
##  [7,]      -17.95 -0.18 0.02 -0.14 4.78  -7.18 7.34    NA    NA    NA
##  [8,]      -18.26 -0.17 0.01 -0.13 4.84  -4.36 7.39 -0.02    NA    NA
##  [9,]        0.83 -0.20 0.06 -0.23 4.58 -14.45 6.75 -0.06 -1.76    NA
## [10,]        0.16 -0.18 0.06 -0.21 4.54 -13.34 6.79 -0.06 -1.75 -0.05
## [11,]        2.99 -0.18 0.07 -0.10 4.11 -12.59 6.66 -0.05 -1.73  0.16
## [12,]       27.15 -0.18 0.04 -0.04 3.49 -22.18 6.08 -0.05 -1.58  0.25
## [13,]       20.65 -0.16 0.04 -0.03 3.22 -20.48 6.12 -0.05 -1.55  0.28
## [14,]       36.46 -0.11 0.05  0.02 2.69 -17.77 3.81  0.00 -1.48  0.31
##         tax ptratio black lstat
##  [1,]    NA      NA    NA    NA
##  [2,]    NA      NA    NA    NA
##  [3,]    NA      NA    NA    NA
##  [4,]    NA      NA    NA    NA
##  [5,]    NA      NA    NA    NA
##  [6,]    NA      NA    NA    NA
##  [7,]    NA      NA    NA    NA
##  [8,]    NA      NA    NA    NA
##  [9,]    NA      NA    NA    NA
## [10,]    NA      NA    NA    NA
## [11,] -0.01      NA    NA    NA
## [12,] -0.01   -1.00    NA    NA
## [13,] -0.01   -1.01  0.01    NA
## [14,] -0.01   -0.95  0.01 -0.52
```

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

Estimates are changing from row to row because as more relevant piece of information are added, the prediction is becoming more and more acurate.

Clear the workspace and load the diamonds dataset.

```
pacman::p_load(ggplot2)
data(diamonds, package = "ggplot2")
```

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

```
summary(diamonds)
```

```
##      carat               cut        color        clarity
##  Min.   :0.2000   Fair     : 1610   D: 6775   SI1    :13065
##  1st Qu.:0.4000   Good     : 4906   E: 9797   VS2    :12258
##  Median :0.7000   Very Good:12082   F: 9542   SI2    : 9194
##  Mean   :0.7979   Premium  :13791   G:11292   VS1    : 8171
##  3rd Qu.:1.0400   Ideal    :21551   H: 8304   VVS2   : 5066
##  Max.   :5.0100                     I: 5422   VVS1   : 3655
##                                     J: 2808   (Other): 2531
##      depth           table           price             x
##  Min.   :43.00   Min.   :43.00   Min.   :  326   Min.   : 0.000  
##  1st Qu.:61.00   1st Qu.:56.00   1st Qu.:  950   1st Qu.: 4.710  
##  Median :61.80   Median :57.00   Median : 2401   Median : 5.700  
##  Mean   :61.75   Mean   :57.46   Mean   : 3933   Mean   : 5.731  
##  3rd Qu.:62.50   3rd Qu.:59.00   3rd Qu.: 5324   3rd Qu.: 6.540  
##  Max.   :79.00   Max.   :95.00   Max.   :18823   Max.   :10.740  
##                                                                  
##        y                z         
##  Min.   : 0.000   Min.   : 0.000  
##  1st Qu.: 4.720   1st Qu.: 2.910  
##  Median : 5.710   Median : 3.530  
##  Mean   : 5.735   Mean   : 3.539  
##  3rd Qu.: 6.540   3rd Qu.: 4.040  
##  Max.   :58.900   Max.   :31.800  
##
```

```
y = diamonds$price
c = diamonds$color
table(c)
```

```
## c
##     D     E     F     G     H     I     J 
##  6775  9797  9542 11292  8304  5422  2808
```

Convert the "c" 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 c.

```
X = rep(1, nrow(diamonds))
X = cbind(X, diamonds$color == 'D')
X = cbind(X, diamonds$color == 'E')
X = cbind(X, diamonds$color == 'F')
X = cbind(X, diamonds$color == 'H')
X = cbind(X, diamonds$color == 'I')
X = cbind(X, diamonds$color == 'J')
colnames(X) = c("Intercept", "is_D", "is_E", "is_F", "is_H", "is_I", "is_J")
head(X)
```

```
##      Intercept is_D is_E is_F is_H is_I is_J
## [1,]         1    0    1    0    0    0    0
## [2,]         1    0    1    0    0    0    0
## [3,]         1    0    1    0    0    0    0
## [4,]         1    0    0    0    0    1    0
## [5,]         1    0    0    0    0    0    1
## [6,]         1    0    0    0    0    0    1
```

Repeat the iterative exercise above we did for Boston here.

```
M = matrix(NA, nrow = ncol(X), ncol = ncol(X))

colnames(M) = colnames(X)
X_j = X[, 1, drop = FALSE]
b = solve(t(X_j) %*% X_j) %*% t(X_j) %*% y
M[1, 1] = b
X_j_2 = X[ , 1:2]
b = solve(t(X_j_2) %*% X_j_2) %*% t(X_j_2) %*% y
b
```

```
##                  [,1]
## Intercept 4042.3784
## is_D      -872.4243
```

```
for(j in 1 : ncol(M)){
  X_j = X[, 1 : j, drop = FALSE]
  b = solve(t(X_j) %*% X_j) %*% t(X_j) %*% y
  M[j, 1:j] = b
}
round(M, 2)
```

```
##      Intercept     is_D     is_E    is_F   is_H    is_I    is_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
```

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

Letter are independent predictors. Because they are indepentdent of each other they have no effect on the other, knowing more information would not change the outcome.

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.

```
n = 100
intercept = rep(1, 100)
y = rnorm ( n, mean = 0, sd = 1)
X = cbind (intercept, rnorm(n, mean = 0, sd = 1))
H = X %*% solve(t(X) %*% X) %*% t(X)
yhat = H %*% y

e = y - yhat
sse = sum(e^2)
sst = sum((y - mean(y))^2)

Rsquared = 1 - sse / sst
Rsquared
```

```
## [1] 0.008492039
```

```
mse = sse / (nrow(X) - ncol(X))
rmse = sqrt(mse) #rmse is standard deviation of errors
rmse
```

```
## [1] 1.013829
```

```
summary(lm(y~X))
```

```
##
## Call:
## lm(formula = y ~ X)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.83724 -0.57110  0.07318  0.64338  2.90259
##
## Coefficients: (1 not defined because of singularities)
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.07098    0.10574  -0.671    0.504
## Xintercept        NA         NA      NA       NA
## X            0.09343    0.10198   0.916    0.362
##
## Residual standard error: 1.014 on 98 degrees of freedom
## Multiple R-squared:  0.008492,   Adjusted R-squared:  -0.001625
## F-statistic: 0.8393 on 1 and 98 DF,  p-value: 0.3618
```

from the last problem. Find the R^2 of an OLS regression of `y ~ X`. You can use the `summary` function of an `lm` model.

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??

```r
for(i in 2:n) {

  new_cols_iid = rnorm ( n, mean = 0, sd = 1)
  X = cbind (X, new_cols_iid)
  linear_reg = lm(y ~ X)

  summary(linear_reg)$sigma #the RMSE
  Rsquared = c(Rsquared, summary(linear_reg)$r.squared)

}
Rsquared
```

```
##   [1] 0.008492039 0.052887356 0.054012614 0.059156100 0.063780001
##   [6] 0.077240503 0.079713119 0.091184895 0.092413821 0.126168535
##  [11] 0.168587385 0.175677954 0.175687948 0.193411460 0.194686082
##  [16] 0.194688876 0.198910419 0.202867217 0.218978508 0.222701889
##  [21] 0.222839011 0.230533494 0.230862573 0.250466341 0.250581226
##  [26] 0.251706221 0.252849651 0.253025610 0.277259197 0.292188801
##  [31] 0.292844744 0.367445668 0.379340021 0.386792362 0.392478682
##  [36] 0.395776931 0.416242497 0.417452261 0.419005224 0.449659858
##  [41] 0.451438813 0.489032346 0.491618519 0.493122235 0.494028515
##  [46] 0.495275876 0.544293116 0.544519224 0.546753111 0.565910987
##  [51] 0.618482458 0.633212898 0.633470016 0.651020428 0.651160867
##  [56] 0.669681146 0.676007759 0.679154775 0.684644587 0.685463032
##  [61] 0.693072075 0.697172792 0.730243911 0.731428325 0.746038031
##  [66] 0.763043324 0.763251910 0.763646490 0.774706784 0.775000094
##  [71] 0.775672165 0.779129891 0.795241858 0.796666916 0.797790887
##  [76] 0.800895827 0.805241344 0.829468927 0.879592842 0.879618041
##  [81] 0.879943663 0.889060184 0.889088909 0.892001500 0.896633690
##  [86] 0.896789925 0.909233333 0.914048232 0.926855186 0.950042404
##  [91] 0.957919147 0.957945108 0.959981465 0.960381795 0.964135739
##  [96] 0.966233851 0.992754345 0.999843906 1.000000000 1.000000000
```

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

R^2 becomes 1. When more columns are added to the matrix, overfitting happen. As a result, the noise in the training data is picked up and learned as concepts by the model.

```r
X_plus_one_col = cbind (X, rnorm(n, mean = 0, sd = 1))
summary(lm(y ~ X_plus_one_col))$r.squared
```

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