

STAT-S 610 Final Project

BJKill

12/3/2020

When simulating our data for our linear model, we need to know

1. How many observations or data points we have, n , such that $n \in \mathbb{N}$
2. How many predictor variables we have, p , such that $p \in \mathbb{N}$, $1 \leq p \leq n - 2$
3. How many of those predictor variables are the “good” ones, k , such that $k \in \mathbb{N}$, $1 \leq k \leq n$
4. How many times we generate the data and run backwards elimination on it, m , such that $m \in \mathbb{N}$
5. The significance level we will be using, α , such that $\alpha \in (0, 1)$

Let's say we have

```
n <- 100      # observations
p <- 30       # predictor vars
k <- 10       # valid predictor vars
m <- 1000    # simulations
alpha <- 0.10 # sig level
```

The first thing we must do is to generate the data. As I mentioned in my presentation, many of these functions can be done with the right version of `apply`, but my brain is just wired to think in nested for loops. I have commented out the checks for garbage inputs into the sub-functions we will be calling, but have identical versions of them all in our main function. This was done simply in order to speed up computation time knitting time. We will verify that the checks work in the main function later.

```
make_model_matrix <- function(n, p) {
  # if (n <= 0 | p <= 0 | is.wholenumber(n) == FALSE | is.wholenumber(p) == FALSE) {return('n and p must be positive integers')} if (n
  # < p+2) {return('n must be at least p+2')} else {
  X <- matrix(nrow = n, ncol = p)
  for (i in 1:p) {
    X[, i] <- rnorm(n)
  }
  return(X)
  # }
}
```

Here is what it gives us:

```
X_mat <- make_model_matrix(n, p)
dim(X_mat)
```

```
## [1] 100 30
```

```
head(X_mat)
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]      [,8]      [,9]      [,10]      [,11]
## [1,]  0.1614674 -0.3144838338  0.3125229  0.7456893 -1.2729457 -1.53860746 -0.8542103 -1.1320693  0.4887743 -0.07105842 -0.7630028
## [2,]  0.9980984  0.0009598075 -1.4834659 -0.4990812  0.4547841  2.27406948  0.8046591 -0.6893381 -1.2055905  0.77462832 -1.0587532
## [3,] -0.8561152 -0.0951000756  0.3772398 -1.0895278 -0.8184469 -1.83938992  2.0557592  0.9134045 -0.6244473  0.44465277 -2.2069251
## [4,]  0.2136356  0.4916502896  0.5304936  0.3605384 -0.3901891 -0.07935598 -0.8342381  2.0198508 -0.6898735 -1.24062785  0.4497749
## [5,]  0.6071477 -0.2389555228 -0.3567721  0.2391191 -0.1018418 -0.03031830  0.2019412 -0.9606820  0.3300564  2.55796609  1.6278476
## [6,]  0.8408528 -0.3549555821  0.1567076 -1.5873811 -0.9539751 -0.38179386 -0.1764600 -1.3507109  1.8196188  0.70806835  1.2029001
##           [,12]      [,13]      [,14]      [,15]      [,16]      [,17]      [,18]      [,19]      [,20]      [,21]      [,22]
## [1,]  2.59901644 -1.6233202  1.01076342  0.44169529 -1.2436496  2.2438031  1.7157048 -0.1331566 -0.1795518  0.40699902 -0.3092473
## [2,] -1.12688064 -1.2766897 -0.71273742 -0.07729913  1.5807764  1.4014500  0.9598206  1.0952755  2.2016400 -0.74825949  1.1216917
## [3,] -0.62528325 -2.4049046 -0.02020352  0.37187049  1.6422714 -0.2032231  0.1187949  0.2873141  0.1110810 -3.36800267  1.3927623
## [4,] -1.73911710  1.1538119 -0.71855003 -0.47966532  1.6119475 -1.3629940  0.5337006 -0.7295574  2.1907279  0.09739504  0.7676709
## [5,] -0.04345282 -0.6792818 -0.17266229  2.39752328 -0.7896784 -0.1351154 -0.7753906  0.7055959 -0.6808899  0.27398253  1.0217329
## [6,] -0.63743109  1.2230692  0.21986904 -0.39244187 -0.9533765  0.2338860 -1.2622322 -0.9383114 -1.0757532 -1.26585138  0.9619381
##           [,23]      [,24]      [,25]      [,26]      [,27]      [,28]      [,29]      [,30]
## [1,]  0.48396755 -0.8275547  0.4178089 -0.8116463 -0.61092870 -0.32785982 -0.75395689  0.3944085
## [2,] -0.57864587 -0.7834855 -0.9011562 -0.7324000 -1.26772810 -1.96158268 -0.01337311 -1.9209070
```

```
## [3,] -1.88492981 -0.5376749 -0.9252172 -0.6875047 -0.93973310  2.02046527  0.49643436  2.3461419
## [4,] -0.06730266  0.6978291 -0.1439633 -0.1723251  0.06166004  0.08204156 -0.74821969 -0.5721033
## [5,] -1.82221734  1.2862141  2.6541049 -0.4469794 -0.87346590 -0.71708027  1.28069477  0.8426553
## [6,]  1.01043859 -0.7564579  2.1669588  1.6007886 -0.34299322 -0.44130204  1.92881468  1.0087734
```

Then, we will use the first k predictor variables as the basis for generating our y values. For simplicity, we will not have an intercept, we will give each predictor variable the same coefficient as its index, and we will use a standard normal error term, like so:

$$Y \sim N(0, 1) + \sum_{i=1}^k i * X_i$$

or,

$$Y \sim 1X_1 + 2X_2 + 3X_3 + \dots + kX_k + \epsilon$$

Here is how we'll do it. It is also important to note that in OLS regression, the order of the predictor variables does not matter, so using the first k predictors instead of randomly selecting which k predictors to use is both mathematically allowable and computationally preferable.

```
make_response_vector <- function(pred_mat, k) {
  # if (k <= 0 | k > ncol(pred_mat) | is.wholenumber(k) == FALSE) {return('k must be a positive integer not exceeding p')} else {
  Y <- vector(length = nrow(pred_mat))
  for (i in 1:nrow(pred_mat)) {
    Y[i] <- rnorm(1)
    for (j in 1:k) {
      Y[i] <- Y[i] + pred_mat[i, j] * j
    }
  }
  return(Y)
  # }
}
```

Using our example `X_mat` from before, here is what we get for our response values.

```
Y_vec <- make_response_vector(X_mat, k)
length(Y_vec)
```

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

```
Y_vec
```

##	[1]	-24.6157216	6.9607563	2.7974322	-6.6505140	21.7118494	-0.2353181	31.3681685	-19.6589840	1.2484673	16.0880041
##	[11]	0.6367024	-11.1487801	25.5288000	-20.0959004	-13.6278434	8.3969409	2.4731348	-31.1726381	-8.8371043	-1.4962839
##	[21]	-26.9803511	5.9434981	-6.0031100	-28.2157961	-33.7318430	-13.4487053	18.1352961	0.9852221	24.4880983	28.9921577
##	[31]	4.9412871	6.5369545	21.4808499	-16.2534541	-27.2485202	-10.7817175	-16.3907447	-5.0046534	-14.4255625	14.3344543
##	[41]	-12.6779444	16.5813189	46.1097295	-30.8976053	2.4865287	-5.8767470	-4.3518464	-0.7172193	-2.3363681	10.7433375
##	[51]	-20.6806645	-13.2677245	-7.1249871	0.5683726	24.7899838	-5.0703946	-31.2094203	14.2500106	-23.2714699	33.7472020
##	[61]	-30.8586587	20.1164384	1.4058339	16.5737095	17.7861747	1.6269354	-2.5198015	2.4203655	-24.7278734	-2.1333788
##	[71]	-2.8283452	-7.5748757	-15.2454965	-21.1355821	23.4796892	-14.8453364	11.5802164	25.0550952	33.8428971	23.9993155
##	[81]	16.9020473	-27.0141270	-8.0172250	0.9178520	-24.6093807	1.8307235	-15.3564223	0.2469825	-13.1879971	12.5497687
##	[91]	-7.1698922	-8.9672215	24.4310033	16.1005390	15.0629246	-29.6993114	45.9489097	-26.4734385	-54.9251399	-5.6795000

We will then create a function that can generate the data and combine the response and the predictors into a single data frame in order for us to use R's built-in `lm` function.

```
make_data_frame <- function(n, p, k) {  
  # if (n <= 0 | p <= 0 | is.wholenumber(n) == FALSE | is.wholenumber(p) == FALSE) {return('n and p must be positive integers')} if (n  
  # < p+2) {return('n must be at least p+2')} if (k <= 0 | k > p | is.wholenumber(k) == FALSE) {return('k must be a positive integer  
  # not exceeding p')} else {  
  X <- make_model_matrix(n, p)  
  Y <- make_response_vector(X, k)  
  df <- data.frame(cbind(X, Y))  
  return(df)  
  # }  
}
```

Let's use this to create a new data frame and see what we get.

```
our_df <- make_data_frame(n, p, k)
head(our_df)
```

##	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
## 1	1.7679049	0.4831702	-1.7354787	0.7797558	0.6034892	1.5433094	-0.96349416	0.4119409	-1.4903467	-2.6779619	-0.25452715
## 2	-0.6798487	-0.7575751	-0.4098459	-0.7434239	-0.1800716	0.3421516	1.09939987	-1.0988883	0.8541668	-0.1604395	-0.15237197
## 3	1.8713542	0.8575405	-1.0347921	1.0809463	0.1585023	-1.1476865	-0.76106270	-2.1922108	-1.2007447	0.6135068	1.01456172
## 4	1.4172229	-0.5804503	-0.4572078	-0.3219929	0.9983584	-0.2784367	-0.91912684	-1.4756648	-0.8465778	-0.9578710	-0.41207782
## 5	1.1716558	0.8340092	0.6389959	-1.1973365	0.8799427	0.5018874	1.19705859	0.3838243	0.5721985	-0.1748426	-0.04214596
## 6	-0.5206040	1.7169247	1.5332338	2.3357444	2.0822094	-0.7354044	0.04160914	0.2060373	-0.9355297	2.1028316	-1.16947315
##	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21	V22
## 1	-1.0478279	0.8811461	-0.6259443	-1.0765437	0.02244520	0.2570489	-0.09379696	-2.3067059	-0.4516891	0.1573405	1.6734689
## 2	-1.7801673	1.6830354	-0.5232833	0.3242616	0.98260132	1.8063506	-0.29821753	0.4343331	-0.5929843	0.1065772	0.6408329
## 3	-1.0032734	-0.9689242	-0.9480286	-0.1789329	-0.25878659	1.0167919	1.11387008	1.4229245	0.1743411	1.9194459	-1.5849357
## 4	-1.7073372	0.6350172	1.0177732	0.1908465	1.34063223	0.3592389	0.92041068	-0.2468192	0.3164213	0.2764036	0.1880668
## 5	0.1782318	1.0214585	0.2793726	-1.3086144	-0.09587208	1.5371667	0.76849071	-0.9917558	0.2644730	-0.7432466	1.1190972
## 6	-0.4148969	1.0390600	-0.5862362	0.6237040	0.97754898	1.3819485	-0.54553484	0.4632642	0.1318430	-0.0108527	-1.8433771
##	V23	V24	V25	V26	V27	V28	V29	V30	Y		
## 1	0.41346558	-1.6287567	0.8456622	-1.80210779	0.67600136	-0.12705081	-0.92803840	-1.0575406	-29.7129056		
## 2	-1.64137851	1.1907250	-1.1349847	0.89951758	0.82095036	-0.80360081	1.26833331	0.5517811	-0.2645392		
## 3	-1.65418611	1.0545074	0.2066857	-0.06005424	-0.03092841	0.72642321	0.37536996	-0.5989711	-27.2630525		
## 4	-0.26829523	0.7557443	-0.2743891	-0.84442202	-2.20541524	0.09122998	-0.09818968	2.3861294	-35.6580318		
## 5	-0.08930364	0.7028457	0.2411683	-0.91987436	0.69116415	0.09391340	0.59270874	-0.6084353	22.0380384		
## 6	1.68332965	1.4775616	-0.3648195	0.22155271	-0.62754718	0.49197724	0.19481683	-0.8623228	36.1526794		

Now that we can generate a data frame just the way we like it, we can create a function that generates a data frame and systematically eliminates the least significant variable (highest p-value) from the linear model one at a time until all of the variables left have p-values that are at most our pre-determined significance level, α . It will return the coefficient matrix of the final linear model along with the $100(1 - \alpha)\%$ CI for each parameter and an indicator of whether the CI for that parameter contained the known parameter.

```
run_BE <- function(n, p, k, alpha) {
  # if (alpha < 0 | alpha > 1) {return('alpha must be in the interval (0,1)')} if (n <= 0 | p <= 0 | is.wholenumber(n) == FALSE |
  # is.wholenumber(p) == FALSE) {return('n and p must be positive integers')} if (n < p+2) {return('n must be at least p+2')} if (k <=
  # 0 | k > p | is.wholenumber(k) == FALSE) {return('k must be a positive integer not exceeding p')} else {
  df <- make_data_frame(n, p, k)
  lm1 <- lm(Y ~ ., df)
  coef_mat <- summary(lm1)$coefficients
  maxp_ind <- which.max(coef_mat[-1, 4])
  maxp_val <- coef_mat[1 + maxp_ind, 4]
  while (maxp_val > alpha) {
    rem_inx <- maxp_ind
    df <- df[, -rem_inx]
    lm1 <- lm(Y ~ ., df)
    coef_mat <- summary(lm1)$coefficients
    maxp_ind <- which.max(coef_mat[-1, 4])
    maxp_val <- coef_mat[1 + maxp_ind, 4]
  }
  display <- cbind(coef_mat, confint(lm1, level = (1 - alpha)), vector(length = nrow(coef_mat)))
  colnames(display)[7] <- "Known Param in CI?"
  display[1, 7] <- (0 >= display[1, 5]) & (0 <= display[1, 6])
  for (i in 2:nrow(display)) {
    index <- as.numeric(str_sub(rownames(display)[i], 2, -1))
    display[i, 7] <- (index >= display[i, 5]) & (index <= display[i, 6])
  }
  return(display)
  # }
}
```

To take a quick peek under the hood, let's create a data frame and see what the while loop is checking for.

```
our_df2 <- make_data_frame(n, p, k)
our_lm <- lm(Y ~ ., our_df2)
our_coef_mat <- summary(our_lm)$coefficients
our_coef_mat
```

```
##              Estimate Std. Error      t value      Pr(>|t|)
## (Intercept) -0.0799516068  0.1309783 -0.610418716 5.435899e-01
## V1          1.2160119934  0.1525734  7.970015037 2.268122e-11
```

```
## V2      2.0884960539  0.1310433 15.937448231 8.020131e-25
## V3      3.0118519207  0.1381639 21.799121140 1.164332e-32
## V4      3.8905248293  0.1458172 26.680833778 4.434320e-38
## V5      4.9677376422  0.1488221 33.380386656 2.586796e-44
## V6      5.9545436102  0.1182482 50.356293776 3.827667e-56
## V7      6.9559420708  0.1266365 54.928431904 1.099519e-58
## V8      8.0430566513  0.1499161 53.650399208 5.379739e-58
## V9      9.2471014424  0.1642426 56.301480005 2.076176e-59
## V10     10.1552936913  0.1522660 66.694427668 2.146193e-64
## V11     -0.0925615200  0.1348740 -0.686281241 4.948355e-01
## V12      0.0068104329  0.1343237  0.050701632 9.597098e-01
## V13     -0.0074236560  0.1153403 -0.064363088 9.488672e-01
## V14      0.3713335235  0.1412931  2.628108094 1.057381e-02
## V15     -0.1379491571  0.1660886 -0.830575498 4.090782e-01
## V16      0.0007371667  0.1519198  0.004852340 9.961424e-01
## V17      0.0247515074  0.1363547  0.181522936 8.564893e-01
## V18      0.0010934586  0.1193441  0.009162238 9.927161e-01
## V19     -0.0606519324  0.1388853 -0.436705240 6.636887e-01
## V20      0.1211687323  0.1189404  1.018735068 3.118875e-01
## V21     -0.1230698074  0.1257051 -0.979036041 3.309824e-01
## V22      0.0685339222  0.1279635  0.535573793 5.939753e-01
## V23     -0.0895274209  0.1429171 -0.626429244 5.331005e-01
## V24     -0.0678413138  0.1356226 -0.500221293 6.185110e-01
## V25      0.0023689799  0.1319486  0.017953810 9.857275e-01
## V26     -0.2373978236  0.1361093 -1.744171124 8.558250e-02
## V27      0.1252121236  0.1289646  0.970903393 3.349876e-01
## V28     -0.1324319327  0.1440187 -0.919546837 3.610132e-01
## V29     -0.0586682004  0.1398836 -0.419407293 6.762214e-01
## V30      0.1160729280  0.1390044  0.835030380 4.065826e-01
```

```
our_maxp_ind <- which.max(our_coef_mat[-1, 4])
our_maxp_ind
```

```
## V16
## 16
```

```
our_maxp_val <- our_coef_mat[1 + our_maxp_ind, 4]
our_maxp_val
```

```
## [1] 0.9961424
```



```
our_maxp_val > alpha
```

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

```
our_rem_inx <- our_maxp_ind  
our_df2 <- our_df2[, -our_rem_inx]  
head(our_df2)
```

##	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
## 1	1.1734594	0.4397714	0.2864831	0.7358692	0.2360170	-1.3563064	1.7209722	-0.8313202	-0.2840616	0.65334025	1.3583196
## 2	1.1691151	0.1691236	-0.9808937	-1.6898011	-0.2845974	0.1515754	-1.0843542	1.3292604	-0.7444682	0.03850351	-0.2414840
## 3	-0.8014314	0.9124348	-0.8810455	-0.6597879	1.9135440	0.2959848	-0.9439043	1.5772797	0.3643282	-0.04342656	0.4571844
## 4	-1.0864494	-1.7059113	1.2409741	-0.4878513	-1.3147370	-1.7645785	0.2834523	-0.5008412	-0.4795585	-2.63525421	-0.8539967
## 5	-1.2894204	0.5933352	1.4414945	-0.8572927	0.5887850	0.9231547	1.2376153	-0.4213638	-0.2269342	0.63732088	2.2010050
## 6	-0.6149584	-0.4436477	0.5262200	-0.5127327	0.2764098	-0.5561630	0.2343169	0.4852381	1.6702283	0.97810359	-0.9497285
##	V12	V13	V14	V15	V17	V18	V19	V20	V21	V22	V23
## 1	-0.2785297	-0.9576929	-0.5710268	-0.5208085	1.0633661	1.39236188	-1.1375914	2.1277752	0.3230704	0.6471092	-0.7363282
## 2	3.5904074	-1.0905328	0.2423051	-2.0185345	0.5944241	0.89209752	-1.0691636	1.0966848	0.1162827	-0.7048193	0.8857983
## 3	-1.0263678	0.3731205	-0.3610651	-0.6887143	-0.4936827	-1.32780144	0.6189362	-0.9652856	-0.1944159	0.3483525	1.1076371
## 4	1.4674480	-0.9545986	0.3977361	2.0396582	-0.1384568	0.35643626	0.1935913	0.8647432	-1.0639277	0.5260272	-1.0530272
## 5	-0.9264428	-1.1434351	0.9180648	-0.9112722	-1.5984791	0.49512106	-0.4422673	0.4351992	0.2701557	-1.0411711	0.5274090
## 6	-0.7322569	0.3505157	-1.3917777	0.9763070	1.9826395	0.02468542	0.3000094	-1.4060703	0.4733145	-0.2417249	0.5597421
##	V24	V25	V26	V27	V28	V29	V30	Y			
## 1	-1.3048861	0.6727092	-1.0989617	0.8584685	0.17312712	1.0607715	1.50513326	7.992035			
## 2	0.7226522	0.3694905	-0.4022894	-2.0960383	1.55042987	-2.1822013	-1.49440834	-12.074794			
## 3	1.2647087	0.3890624	-2.3112631	0.3236895	0.50969947	3.1128754	0.09844672	16.773191			
## 4	0.9398877	0.6382616	-0.2053542	2.8591224	-0.01312739	0.5327015	-0.42146554	-51.905363			
## 5	0.3555550	-0.4063627	1.9453242	-1.0814392	0.68715155	1.8339050	-1.07161339	17.709714			
## 6	0.7421536	0.2906891	-0.1802185	0.1241192	-0.08078882	0.2809936	-0.19961959	26.003872			

```
our_lm <- lm(Y ~ ., our_df2)  
our_coef_mat <- summary(our_lm)$coefficients  
our_coef_mat
```

##	Estimate	Std. Error	t value	Pr(> t)
## (Intercept)	-0.0800065262	0.1295530	-0.617558345	5.388704e-01
## V1	1.2160070045	0.1514762	8.027708987	1.624277e-11
## V2	2.0885818376	0.1289145	16.201289089	2.208702e-25
## V3	3.0117526658	0.1356619	22.200439088	2.090534e-33
## V4	3.8904564614	0.1440945	26.999336091	9.708439e-39
## V5	4.9675839586	0.1443704	34.408597511	1.344504e-45
## V6	5.9546166598	0.1164453	51.136613956	3.540469e-57

```
## V7      6.9559968970 0.1252272 55.546999600 1.240990e-59
## V8      8.0432956747 0.1405772 57.216231669 1.632277e-60
## V9      9.2471848681 0.1621694 57.021775288 2.061325e-60
## V10     10.1553480460 0.1507649 67.358853903 2.183512e-65
## V11     -0.0926517284 0.1326290 -0.698577979 4.871303e-01
## V12      0.0068105111 0.1333609 0.051068291 9.594166e-01
## V13     -0.0075185317 0.1128561 -0.066620534 9.470737e-01
## V14      0.3714366734 0.1386835 2.678304848 9.213899e-03
## V15     -0.1378212899 0.1628093 -0.846519587 4.001477e-01
## V17      0.0247145232 0.1351656 0.182846244 8.554472e-01
## V18      0.0009647788 0.1155263 0.008351164 9.933606e-01
## V19     -0.0605755669 0.1370016 -0.442152382 6.597420e-01
## V20      0.1211817159 0.1180579 1.026460198 3.082087e-01
## V21     -0.1231192565 0.1243932 -0.989758824 3.257010e-01
## V22      0.0686167419 0.1259111 0.544961969 5.875118e-01
## V23     -0.0896062502 0.1409729 -0.635627613 5.270918e-01
## V24     -0.0678621824 0.1345827 -0.504241540 6.156752e-01
## V25      0.0023400279 0.1308687 0.017880727 9.857849e-01
## V26     -0.2373259630 0.1343313 -1.766721472 8.163418e-02
## V27      0.1251768464 0.1278365 0.979195083 3.308557e-01
## V28     -0.1322903782 0.1400222 -0.944781570 3.480202e-01
## V29     -0.0586842795 0.1388419 -0.422669858 6.738317e-01
## V30      0.1162074395 0.1352358 0.859294670 3.931112e-01
```

```
our_maxp_ind <- which.max(our_coef_mat[-1, 4])
our_maxp_ind
```

```
## V18
## 17
```

```
our_maxp_val <- our_coef_mat[1 + our_maxp_ind, 4]
our_maxp_val
```

```
## [1] 0.9933606
```

If any of the variables have a p-value greater than alpha, the `run_BE` function will repeat that process of removing the variable with the highest p-value until it settles on a model where all of the variables have significant p-values.

Now, when it comes to cleaning up and speeding up my code, I made HUGE time improvements by tinkering with how I set up `run_BE`. At first, I think I may have had it in my head that minimizing the amount of code I typed would minimize the runtime; however, that's definitely not the case. Originally, I had something like this:

```
run_BE_OLD <- function(n, p, k, alpha) {
  df <- make_data_frame(n, p, k)
  while (summary(lm(Y ~ ., df))$coefficients[1 + which.max(summary(lm(Y ~ ., df))$coefficients[-1, 4]), 4] > alpha) {
    rem_inx <- which.max(summary(lm(Y ~ ., df))$coefficients[-1, 4])
    df <- df[, -rem_inx]
  }
  display <- cbind(summary(lm(Y ~ ., df))$coefficients, confint(lm(Y ~ ., df)))
  display <- cbind(display, vector(length = nrow(display)))
  colnames(display)[7] <- "Known Param in CI?"
  display[1, 7] <- (0 >= display[1, 5]) & (0 <= display[1, 6])
  for (i in 2:nrow(display)) {
    display[i, 7] <- (as.numeric(str_sub(rownames(display)[i], 2, -1)) >= display[i, 5]) & (as.numeric(str_sub(rownames(display)[i],
    2, -1)) <= display[i, 6])
  }
  return(display)
}
```

If we look carefully, this function has to generate the linear model and extract the coefficient matrix three separate times in one iteration! That's extremely inefficient, and the runtime of my code improved by what felt like an order of magnitude when I re-wrote the while loop so that it only had to generate the linear model, extract the coefficient matrix, identify the max p-value, check it against alpha, and potentially remove it from the model exactly once each iteration. It was easily the biggest “eureka” moment I had throughout this entire process. The time it takes a human to read the code does not determine how long it takes a computer to process the code; I turned four lines of code into thirteen and saved hours of cumulative computation time (if you count every time I ran the code or knitted the document).

Let us quickly compare the runtimes of these two functions by using the `microbenchmark` package/function.

```
microbenchmark(run_BE_OLD(n,p,k,alpha))
```

```
## Unit: milliseconds
##           expr      min       lq      mean   median       uq      max  neval
## run_BE_OLD(n, p, k, alpha) 103.372 127.0174 134.0197 133.0382 139.6649 179.3644   100
```

```
microbenchmark(run_BE(n,p,k,alpha))
```

```
## Unit: milliseconds
##           expr  min       lq      mean   median       uq      max  neval
## run_BE(n, p, k, alpha) 35.1 40.87245 45.22704 43.0921 46.34205 156.4734   100
```

So, as expected, making the machine compute the linear model only once instead of three times per iteration cut our runtime down to about a third of what it was before. Nice!

Okay, let's try it on for size and see what happens.

```
BE <- run_BE(n,p,k,alpha); BE
```

##	Estimate	Std. Error	t value	Pr(> t)	5 %	95 % Known Param in CI?
## (Intercept)	0.03609392	0.1107029	0.3260432	7.451658e-01	-0.1479335 0.22012130	1
## V1	0.82622253	0.1071910	7.7079438	1.822441e-11	0.6480331 1.00441199	1
## V2	1.86431629	0.1272093	14.6554998	2.446158e-25	1.6528494 2.07578323	1
## V3	3.01161963	0.1135093	26.5319180	9.292974e-44	2.8229270 3.20031228	1
## V4	4.11231638	0.1203036	34.1828265	1.375562e-52	3.9123292 4.31230352	1
## V5	5.08221277	0.1250431	40.6436927	8.194834e-59	4.8743469 5.29007865	1
## V6	5.92472483	0.1081912	54.7616202	8.979474e-70	5.7448728 6.10457689	1
## V7	7.00844544	0.1107836	63.2624936	3.748222e-75	6.8242839 7.19260697	1
## V8	7.91748513	0.1125356	70.3553870	3.895270e-79	7.7304111 8.10455912	1
## V9	9.01686564	0.1210325	74.4995253	2.747459e-81	8.8156667 9.21806455	1
## V10	10.02873274	0.1017032	98.6078656	7.099628e-92	9.8596661 10.19779942	1
## V26	-0.21922870	0.1185120	-1.8498441	6.769239e-02	-0.4162376 -0.02221982	0

Once more, with feeling!

```
BE <- run_BE(n,p,k,alpha); BE
```

##	Estimate	Std. Error	t value	Pr(> t)	5 %	95 % Known Param in CI?
## (Intercept)	-0.0493934	0.09425199	-0.5240569	6.016029e-01	-0.206132431 0.10734563	1
## V1	0.8731852	0.08894031	9.8176539	1.198286e-15	0.725279359 1.02109101	1
## V2	1.9201862	0.10743918	17.8723097	1.569685e-30	1.741517198 2.09885528	1
## V3	2.9695736	0.08254605	35.9747535	3.279551e-53	2.832301331 3.10684593	1
## V4	4.2092370	0.10277712	40.9549994	9.732419e-58	4.038320874 4.38015317	0
## V5	5.0330977	0.10149948	49.5874242	1.620398e-64	4.864306237 5.20188914	1
## V6	6.0682717	0.10337466	58.7017331	1.433276e-70	5.896361860 6.24018153	1
## V7	7.0149174	0.09817048	71.4564861	1.100237e-77	6.851661974 7.17817276	1
## V8	8.0279866	0.09204711	87.2160634	6.064646e-85	7.874914229 8.18105896	1
## V9	8.8983211	0.09616067	92.5359728	4.165207e-87	8.738408002 9.05823425	1
## V10	9.8675235	0.08815721	111.9309832	4.488618e-94	9.720919906 10.01412700	1
## V11	0.1547330	0.08965310	1.7259081	8.799743e-02	0.005641834 0.30382419	0
## V14	0.2227150	0.10041647	2.2179130	2.922778e-02	0.055724564 0.38970541	0
## V15	-0.1940068	0.09104935	-2.1307871	3.599548e-02	-0.345419881 -0.04259367	0
## V21	-0.1784749	0.09694679	-1.8409573	6.911667e-02	-0.339695316 -0.01725447	0

Now that we know our BE program works, we can have it run m times and compute aggregate data of our m simulations. We want to know the proportion of times our model creates confidence intervals that contain the known parameter, as well as the proportion of the simulations that each variable was significant.

```
run_simulation <- function(n, p, k, alpha, m) {
  if (m <= 0 | is.wholenumber(m) == FALSE) {
    return("m must be a positive integer")
  }
  if (alpha < 0 | alpha > 1) {
    return("alpha must be in the interval (0,1)")
  }
  if (n <= 0 | p <= 0 | is.wholenumber(n) == FALSE | is.wholenumber(p) == FALSE) {
    return("n and p must be positive integers")
  }
  if (n < p + 2) {
    return("n must be at least p+2")
  }
  if (k <= 0 | k > p | is.wholenumber(k) == FALSE) {
    return("k must be a positive integer not exceeding p")
  } else {
    CI_freq <- vector(length = p + 1)
    sig_freq <- vector(length = p + 1)
    full_df <- make_data_frame(n, p, k)
    var_names <- rownames(summary(lm(Y ~ ., full_df))$coefficients)
    names(CI_freq) <- var_names
    names(sig_freq) <- var_names
    for (i in 1:m) {
      display <- run_BE(n, p, k, alpha)
      CI_freq[1] <- CI_freq[1] + display[1, 7]
      sig_freq[1] <- sig_freq[1] + as.numeric(display[1, 4] <= alpha)
      for (j in 2:nrow(display)) {
        index <- as.numeric(str_sub(rownames(display)[j], 2, -1)) + 1
        CI_freq[index] <- CI_freq[index] + display[j, 7]
        sig_freq[index] <- sig_freq[index] + 1
      }
    }
    CI_perc <- CI_freq/m
    sig_perc <- sig_freq/m
    accuracy_mat <- cbind(round(CI_perc * 100, 2), round((sig_freq/m) * 100, 2))
    if (p > 1) {
      accuracy_mat <- rbind(accuracy_mat, c(mean(accuracy_mat[2:(k + 1), 1]), mean(accuracy_mat[c(1, (k + 2):(p + 1)), 2])))
      rownames(accuracy_mat)[p + 2] <- "Averages"
    }

    colnames(accuracy_mat) <- c("% Param in CI", "% Param Significant")
  }
}
```

```

    return(accuracy_mat)
  }
}

```

Let's quickly parse through an iteration of the nested for loop to see what it's doing for us.

```

CI_freq <- vector(length = p + 1)
sig_freq <- vector(length = p + 1)
full_df <- make_data_frame(n, p, k)
var_names <- rownames(summary(lm(Y ~ ., full_df))$coefficients)
names(CI_freq) <- var_names
names(sig_freq) <- var_names

display <- run_BE(n, p, k, alpha)
display

```

##	Estimate	Std. Error	t value	Pr(> t)	5 %	95 %	Known Param in CI?
## (Intercept)	-0.1739077	0.10656486	-1.631942	1.062654e-01	-0.35105618	0.003240865	1
## V1	1.0002224	0.09859247	10.145018	1.812342e-16	0.83632684	1.164118029	1
## V2	1.9948441	0.11715578	17.027280	1.367091e-29	1.80008976	2.189598513	1
## V3	3.0786585	0.10264737	29.992570	5.556688e-48	2.90802227	3.249294821	1
## V4	3.8398642	0.11676776	32.884628	3.250453e-51	3.64575485	4.033973549	1
## V5	4.9929563	0.11390896	43.832866	1.448921e-61	4.80359928	5.182313333	1
## V6	6.0795220	0.12684097	47.930272	7.615535e-65	5.86866739	6.290376577	1
## V7	6.9357272	0.11975413	57.916391	7.409853e-72	6.73665341	7.134800941	1
## V8	8.0644052	0.11250987	71.677315	7.783586e-80	7.87737400	8.251436467	1
## V9	9.0253760	0.12385123	72.872721	1.859560e-80	8.81949137	9.231260541	1
## V10	10.0104099	0.09721617	102.970630	1.624573e-93	9.84880222	10.172017590	1
## V13	0.1701079	0.09501330	1.790359	7.683529e-02	0.01216218	0.328053674	0

```

CI_freq[1] <- CI_freq[1] + display[1, 7]
CI_freq

```

## (Intercept)	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
## 1	0	0	0	0	0	0	0	0	0	0
## V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21
## 0	0	0	0	0	0	0	0	0	0	0
## V22	V23	V24	V25	V26	V27	V28	V29	V30		
## 0	0	0	0	0	0	0	0	0		

```
sig_freq[1] <- sig_freq[1] + as.numeric(display[1, 4] <= alpha)
sig_freq
```

```
## (Intercept)      V1      V2      V3      V4      V5      V6      V7      V8      V9      V10
##           0         0         0         0         0         0         0         0         0         0
##      V11      V12      V13      V14      V15      V16      V17      V18      V19      V20      V21
##           0         0         0         0         0         0         0         0         0         0
##      V22      V23      V24      V25      V26      V27      V28      V29      V30
##           0         0         0         0         0         0         0         0         0
```

```
index <- as.numeric(str_sub(rownames(display)[2], 2, -1)) + 1
index
```

```
## [1] 2
```

```
CI_freq[index] <- CI_freq[index] + display[2, 7]
CI_freq
```

```
## (Intercept)      V1      V2      V3      V4      V5      V6      V7      V8      V9      V10
##           1         1         0         0         0         0         0         0         0         0
##      V11      V12      V13      V14      V15      V16      V17      V18      V19      V20      V21
##           0         0         0         0         0         0         0         0         0         0
##      V22      V23      V24      V25      V26      V27      V28      V29      V30
##           0         0         0         0         0         0         0         0         0
```

```
sig_freq[index] <- sig_freq[index] + 1
sig_freq
```

```
## (Intercept)      V1      V2      V3      V4      V5      V6      V7      V8      V9      V10
##           0         1         0         0         0         0         0         0         0         0
##      V11      V12      V13      V14      V15      V16      V17      V18      V19      V20      V21
##           0         0         0         0         0         0         0         0         0         0
##      V22      V23      V24      V25      V26      V27      V28      V29      V30
##           0         0         0         0         0         0         0         0         0
```

```
# To finish up the loop for our one generated data set:
for (j in 3:nrow(display)) {
  index <- as.numeric(str_sub(rownames(display)[j], 2, -1)) + 1
  CI_freq[index] <- CI_freq[index] + display[j, 7]
  sig_freq[index] <- sig_freq[index] + 1
}
CI_freq
```

## (Intercept)	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
## 1	1	1	1	1	1	1	1	1	1	1
## V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21
## 0	0	0	0	0	0	0	0	0	0	0
## V22	V23	V24	V25	V26	V27	V28	V29	V30		
## 0	0	0	0	0	0	0	0	0		

sig_freq

## (Intercept)	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
## 0	1	1	1	1	1	1	1	1	1	1
## V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21
## 0	0	1	0	0	0	0	0	0	0	0
## V22	V23	V24	V25	V26	V27	V28	V29	V30		
## 0	0	0	0	0	0	0	0	0		

Alright, let's go ahead and give it a whirl. We'll start with our current values of $n = 100$, $p = 30$, $k = 10$, $\alpha = 0.1$, and $m = 1000$. Appended to the end is the average % of the time it correctly captured the known parameters and the % of the time it incorrectly found a “bad” parameter significant.

```
output <- run_simulation(n,p,k,alpha,m); output
```

##	% Param in CI	% Param Significant
## (Intercept)	89.30	10.70000
## V1	84.70	100.00000
## V2	85.40	100.00000
## V3	86.90	100.00000
## V4	85.80	100.00000
## V5	85.80	100.00000
## V6	87.70	100.00000
## V7	86.70	100.00000
## V8	86.50	100.00000
## V9	85.80	100.00000
## V10	86.80	100.00000
## V11	0.00	11.70000
## V12	0.00	14.70000
## V13	0.00	14.60000
## V14	0.00	13.20000
## V15	0.00	13.10000
## V16	0.00	12.30000
## V17	0.00	12.20000
## V18	0.00	13.20000
## V19	0.00	13.10000
## V20	0.00	11.30000
## V21	0.00	12.70000
## V22	0.00	12.00000
## V23	0.00	12.40000
## V24	0.00	12.30000
## V25	0.00	13.10000
## V26	0.00	12.30000
## V27	0.00	12.70000
## V28	0.00	11.20000
## V29	0.00	11.90000
## V30	0.00	11.30000
## Averages	86.21	12.47619

Now that we know `run_simulation` works the way we want, let's verify that our checks for invalid input values work properly.

```
n<-10; p<-30; k<-15; alpha<-0.10; m <- 1000  
output <- run_simulation(n,p,k,alpha,m); output
```

```
## [1] "n must be at least p+2"
```

```
n<-100; p<-30.5; k<-15; alpha<-0.10; m <- 1000  
output <- run_simulation(n,p,k,alpha,m); output
```

```
## [1] "n and p must be positive integers"
```

```
n<-100; p<-30; k<-35; alpha<-0.10; m <- 1000  
output <- run_simulation(n,p,k,alpha,m); output
```

```
## [1] "k must be a positive integer not exceeding p"
```

```
n<-100; p<-30; k<-15; alpha<-1.10; m <- 1000  
output <- run_simulation(n,p,k,alpha,m); output
```

```
## [1] "alpha must be in the interval (0,1)"
```

```
n<-100; p<-30; k<-15; alpha<-0.10; m <- 1000.2  
output <- run_simulation(n,p,k,alpha,m); output
```

```
## [1] "m must be a positive integer"
```

```
n<-100; p<-30; k<- -15; alpha<-0.10; m <- 1000  
output <- run_simulation(n,p,k,alpha,m); output
```

```
## [1] "k must be a positive integer not exceeding p"
```

One might expect that the model to be less accurate if it is given it less data/information. We originally gave it 100 data points. Let's see what happens if we halve that to $n = 50$.

```
n<-50; p<-30; k<- 15; alpha<-0.10; m <- 1000
output <- run_simulation(n,p,k,alpha,m); output
```

##	% Param in CI	% Param Significant
## (Intercept)	82.50000	17.50000
## V1	77.90000	99.80000
## V2	80.30000	100.00000
## V3	80.30000	100.00000
## V4	81.20000	100.00000
## V5	82.50000	100.00000
## V6	82.00000	100.00000
## V7	81.30000	100.00000
## V8	83.30000	100.00000
## V9	82.00000	100.00000
## V10	81.40000	100.00000
## V11	80.20000	100.00000
## V12	82.50000	100.00000
## V13	80.30000	100.00000
## V14	82.70000	100.00000
## V15	82.40000	100.00000
## V16	0.00000	13.70000
## V17	0.00000	15.20000
## V18	0.00000	15.40000
## V19	0.00000	16.20000
## V20	0.00000	16.40000
## V21	0.00000	15.00000
## V22	0.00000	18.10000
## V23	0.00000	14.40000
## V24	0.00000	15.50000
## V25	0.00000	15.70000
## V26	0.00000	15.60000
## V27	0.00000	16.10000
## V28	0.00000	14.90000
## V29	0.00000	15.30000
## V30	0.00000	16.50000
## Averages	81.35333	15.71875

Here, let's give the model less data and fewer variables to work with, but let's make most of them "good".

```
p<-10; k<-8; n<-20  
output <- run_simulation(n,p,k,alpha,m); output
```

##	% Param in CI	% Param Significant
## (Intercept)	84.700	15.3
## V1	80.300	88.0
## V2	86.500	99.8
## V3	85.600	100.0
## V4	86.100	100.0
## V5	85.700	100.0
## V6	85.000	100.0
## V7	85.900	100.0
## V8	85.500	100.0
## V9	0.000	12.7
## V10	0.000	12.8
## Averages	85.075	13.6

Now, let's revert back to our original input parameters and change the alpha to see what happens.

```
n<-100; p<-30; k<-15; alpha<-0.02
output <- run_simulation(n,p,k,alpha,m); output
```

##	% Param in CI	% Param Significant
## (Intercept)	98.50000	1.5
## V1	98.20000	100.0
## V2	98.50000	100.0
## V3	97.40000	100.0
## V4	97.70000	100.0
## V5	97.50000	100.0
## V6	97.80000	100.0
## V7	97.40000	100.0
## V8	97.60000	100.0
## V9	97.50000	100.0
## V10	98.20000	100.0
## V11	97.30000	100.0
## V12	98.10000	100.0
## V13	97.10000	100.0
## V14	98.10000	100.0
## V15	97.60000	100.0
## V16	0.00000	2.6
## V17	0.00000	2.6
## V18	0.00000	2.1
## V19	0.00000	2.1
## V20	0.00000	2.4
## V21	0.00000	2.3
## V22	0.00000	2.9
## V23	0.00000	2.5
## V24	0.00000	1.8
## V25	0.00000	1.6
## V26	0.00000	2.2
## V27	0.00000	1.8
## V28	0.00000	1.8
## V29	0.00000	1.3
## V30	0.00000	2.1
## Averages	97.73333	2.1

Finally, let's make it really work. Let's say we have 500 data points on 100 predictor variables, of which 35 of them are "valid". We will run the simulation 10,000 times using $\alpha = 0.05$. Let's see how it plays out!

```
n<-500; p<-100; k<-35; alpha<-0.05; m<-10000
output <- run_simulation(n,p,k,alpha,m); output
```

##	% Param in CI	% Param Significant
## (Intercept)	94.26000	5.740000
## V1	94.21000	100.000000
## V2	94.01000	100.000000
## V3	94.56000	100.000000
## V4	94.37000	100.000000
## V5	93.87000	100.000000
## V6	94.32000	100.000000
## V7	94.20000	100.000000
## V8	94.43000	100.000000
## V9	94.70000	100.000000
## V10	94.36000	100.000000
## V11	94.23000	100.000000
## V12	94.22000	100.000000
## V13	94.13000	100.000000
## V14	93.93000	100.000000
## V15	94.11000	100.000000
## V16	94.18000	100.000000
## V17	94.18000	100.000000
## V18	94.17000	100.000000
## V19	94.39000	100.000000
## V20	94.04000	100.000000
## V21	94.44000	100.000000
## V22	94.39000	100.000000
## V23	94.07000	100.000000
## V24	93.95000	100.000000
## V25	94.13000	100.000000
## V26	94.25000	100.000000
## V27	94.44000	100.000000
## V28	94.07000	100.000000
## V29	93.88000	100.000000
## V30	94.49000	100.000000
## V31	93.79000	100.000000
## V32	93.87000	100.000000
## V33	94.44000	100.000000
## V34	94.29000	100.000000
## V35	94.37000	100.000000
## V36	0.00000	5.330000

## V37	0.00000	5.770000
## V38	0.00000	5.600000
## V39	0.00000	5.600000
## V40	0.00000	5.420000
## V41	0.00000	5.970000
## V42	0.00000	5.740000
## V43	0.00000	5.220000
## V44	0.00000	5.640000
## V45	0.00000	5.660000
## V46	0.00000	5.780000
## V47	0.00000	5.450000
## V48	0.00000	5.250000
## V49	0.00000	5.470000
## V50	0.00000	5.620000
## V51	0.00000	5.650000
## V52	0.00000	5.420000
## V53	0.00000	5.700000
## V54	0.00000	5.980000
## V55	0.00000	5.220000
## V56	0.00000	5.370000
## V57	0.00000	5.710000
## V58	0.00000	5.680000
## V59	0.00000	5.960000
## V60	0.00000	5.490000
## V61	0.00000	6.010000
## V62	0.00000	5.610000
## V63	0.00000	5.840000
## V64	0.00000	5.480000
## V65	0.00000	5.690000
## V66	0.00000	5.740000
## V67	0.00000	6.140000
## V68	0.00000	5.640000
## V69	0.00000	5.430000
## V70	0.00000	5.990000
## V71	0.00000	5.110000
## V72	0.00000	5.450000
## V73	0.00000	5.590000
## V74	0.00000	5.660000
## V75	0.00000	5.770000
## V76	0.00000	5.530000
## V77	0.00000	5.740000
## V78	0.00000	5.340000
## V79	0.00000	5.660000
## V80	0.00000	5.460000

## V81	0.00000	5.490000
## V82	0.00000	5.690000
## V83	0.00000	5.570000
## V84	0.00000	5.770000
## V85	0.00000	5.630000
## V86	0.00000	5.340000
## V87	0.00000	5.570000
## V88	0.00000	5.350000
## V89	0.00000	5.700000
## V90	0.00000	5.550000
## V91	0.00000	6.000000
## V92	0.00000	5.280000
## V93	0.00000	5.350000
## V94	0.00000	5.490000
## V95	0.00000	5.930000
## V96	0.00000	5.790000
## V97	0.00000	5.500000
## V98	0.00000	5.660000
## V99	0.00000	5.820000
## V100	0.00000	5.510000
## Averages	94.21371	5.610758

In conclusion, it is clear to me that our assumptions about how inferential statistics applies to regression coefficients in multiple linear regression problems are being violated. Every time we run a simulation, our confidence intervals contain the known parameter at a lower percentage than they should. i.e.

$$Pr(b_j^L \leq \beta_j \leq b_j^U)_{obs} < Pr(b_j^L \leq \beta_j \leq b_j^U)_{pred}$$

$$Pr(b_j^L \leq \beta_j \leq b_j^U)_{obs} < (1 - \alpha)$$

Additionally, OLS regression finds the unused predictor variables to be significant at a higher rate than it should. i.e.

$$Pr(\text{Type I Error})_{obs} > Pr(\text{Type I Error})_{pred}$$

$$Pr(\text{Type I Error})_{obs} > \alpha$$

Now, this only seems to be an issue when $p > 1$. When we're living in the world of simple linear regression, our model is about as accurate as we would predict. e.g.

```
output2 <- run_simulation(100,1,1,0.10,1000); output2
```

```
##           % Param in CI % Param Significant
## (Intercept)      88.1          11.9
## V1              88.4          100.0
```

```
output2 <- run_simulation(100,1,1,0.10,1000); output2
```

```
##           % Param in CI % Param Significant
## (Intercept)      89.2          10.8
## V1              92.0          100.0
```

```
output2 <- run_simulation(100,1,1,0.10,1000); output2
```

```
##           % Param in CI % Param Significant
## (Intercept)      89.1          10.9
## V1              88.2          100.0
```

```
output2 <- run_simulation(100,1,1,0.10,1000); output2
```

```
##           % Param in CI % Param Significant
## (Intercept)      89.4          10.6
## V1              90.7          100.0
```

There is a lot of literature out there on model selection and post-model inference, and many/most authors have suggested that the reason for the difference between expectation and reality stems from the added randomness that comes in the model selection process itself. Be it backward elimination, forward selection, or any other type of stochastic model selection process, the process itself adds variability and randomness that isn't taken into account by our standard t-tests for the significance of OLS regression coefficients.