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

```r
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,] -0.01888188  0.47530840 -0.5427674 -0.8821775 -0.13507070  0.65334492 -0.2147378 -0.2096654 -0.15248592  0.86072465  1.4811648
## [2,]  1.28950260  1.82577817 -0.7154668  1.9184075  0.21265417 -0.75894783  0.7624225 -0.8892883 -1.01784983  1.05192264 -1.1817283
## [3,] -0.89595573 -1.66444870  1.4417953 -0.3083865 -0.41308194 -1.17511593 -0.4953423  1.5215521 -1.90696596  0.41389044  0.3675944
## [4,]  0.04567270  0.65162147 -0.6465896 -2.0958047  2.28405994  0.37535044 -1.5080627 -0.8727232 -0.40629947 -0.51613481  0.4729415
## [5,] -0.67007532  1.00650468  1.1741168  0.2101680 -0.06119666 -0.02558273 -0.9313971  0.2266290 -0.49459556  0.02874242  1.1774039
## [6,]  0.77400892  0.05878613 -0.7565729 -0.2188504  1.59591536  0.91919313  0.2110231  0.1704626 -0.01758815  0.91543933  1.4816979
##            [,12]       [,13]       [,14]      [,15]       [,16]      [,17]      [,18]      [,19]      [,20]        [,21]      [,22]
## [1,]  0.07824386  0.44039399 -1.26690519  0.6112086  0.06820309  0.5897634 -1.8027202  0.9844126 -0.2433463  1.452477941 -0.7331951
## [2,]  2.08318208  0.05810341 -0.13904671 -1.6878284 -0.99931822 -0.4763537 -0.6604590  0.5486020 -0.4707947  0.239555888 -1.9741330
## [3,] -1.93031135  0.45282847  0.68690878 -0.7291094  0.64925271  0.5842508 -0.3498689  0.3847821 -0.3085067 -0.005832967 -0.4765115
## [4,] -0.83994260 -0.30282420  0.35769286 -0.8433641 -0.54432120 -0.8925326 -2.8202031  1.2694539 -0.7218715 -0.174072145 -0.4179794
## [5,] -0.01425256 -0.56570205 -0.01819993  1.1418481  0.32586166 -1.0321641  1.2026461 -1.9553835  1.1333882  0.502449611  0.6429869
## [6,]  1.44077270 -0.53130581 -1.70104107 -1.3893909 -0.40822301  0.2016952  2.3184549 -0.3684023  0.6225547 -1.666562842  0.8353368
##           [,23]       [,24]       [,25]       [,26]      [,27]      [,28]      [,29]     [,30]
## [1,] -1.5995055  0.68430129 -0.59565724 -2.21256441 -1.2113234 -0.1513810  1.3996004 0.4102721
## [2,]  1.7477181  1.12083655 -0.09989669  1.06278948 -0.8354381 -0.1150799  0.1704598 0.1911783
```

```
## [3,] -2.0056534 -0.10042238  0.50876045  0.38817142  0.1308237  1.6798520 -1.1888651 -2.2172264
## [4,] -0.1368212 -0.04309795 -0.91524182  0.04378366 -0.6106172 -0.9657368  1.1814260  0.4501621
## [5,]  0.2030568 -1.37407936 -0.65199915  0.66769052  1.9981094  1.3617019 -0.2171393 -2.0602567
## [6,]  0.7094828 -0.08164620  0.08460603 -1.23223835  1.8965680 -1.2893134 -0.2303106  0.9963312
```

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:

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

$$or,$$

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

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]   5.7486487   6.7120486 -14.9399382 -22.9185138  -4.8152147  22.4926240 -13.3394961  26.6385104 -25.5187849  -1.3856967
##  [11]  -5.0547208   4.7638737  24.7391616 -17.0735916  -3.0405902  36.5731856  13.7304452   6.3224867  18.0916198 -12.4772679
##  [21]  19.9702720  -4.4630669  26.6471640  -9.1135330  26.9544315  17.1293913  15.8091875  12.0298815 -14.4807672 -19.1878536
##  [31]  30.6454878  30.6552658  -0.6658776  14.1003281  18.2886486 -21.5610719 -15.8431387 -16.4699913  10.2753503  -3.0795472
##  [41]  -4.1956356  -3.0915801  -7.9040431   8.1706206  10.5008220 -17.6128725  -7.0708258 -23.3498652   9.0660783   8.3870644
##  [51]  12.5703167  -1.4616909 -24.3316583  -4.0004670  36.1622255  -9.3543237  -0.6431452  -3.4212922 -23.9121900 -23.9812067
##  [61] -28.9840954 -49.3932364 -22.5019134 -15.4908588   4.5603624   5.0990726 -16.1294066  -2.7120608 -22.2371853  -5.1947715
##  [71]  30.5767135  -2.1673931  -7.2147041 -21.8652099   2.1130731 -15.3968218 -10.1695143   1.0599943  -3.8249384 -10.6661325
##  [81]  14.6642554 -19.8555783   6.0310658  18.4312652  38.4673008  -0.7647263 -18.4480733  16.2875830  29.4903428   6.3852021
##  [91]  19.5217834  15.8476135 -33.2063331 -21.1811082   7.9560131  15.0282649 -15.1353396  18.5238805 -17.9649531  12.6936644
```

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.

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

```
##           V1         V2         V3          V4         V5         V6         V7          V8          V9         V10        V11
## 1  0.4716609 -0.5467950 -0.6379441  3.31025018 -0.4377132 -0.5104265  0.6613304 -0.72579582  0.02389505  0.31343527  0.6390630
## 2  0.6807395  0.0928063 -1.5690482  0.01842481 -0.3487358  2.3750422 -0.5611734  0.54499318 -0.34715611  0.08638206  1.7909488
## 3 -2.3469013  0.1918362 -0.0633253  0.14078024 -0.8441282  0.6436644 -0.8204565  0.47828275  0.91243113 -0.67626595  2.7236969
## 4 -2.6849963 -0.3489079 -0.3024382 -0.07763463  1.0601906 -1.1314427  0.2271634 -0.05803075 -0.07651846 -0.33489001 -0.8013691
## 5 -0.7067908 -1.0225375 -0.1662779  0.17159949  0.4046575 -1.3164160  0.7663547 -0.79570046  0.07286727 -1.11460044 -0.2062126
## 6  0.4340525  0.7416590 -0.3788548 -1.74197733  0.4375055  0.6009514 -0.6037975  0.60579419  0.08268743  0.48036147 -0.8423103
##          V12        V13        V14         V15        V16         V17        V18        V19        V20        V21        V22
## 1  0.1621202 -2.0074667  1.0833309  0.26274648  0.1703593 -0.74434842 -0.3080922  0.6377719  0.9489180  0.5913264  0.8052987
## 2 -0.8022875  0.5238910 -0.5344112  0.07767616 -1.4567525  0.79833746 -1.0113004  1.7281384  2.0038599  0.7394718 -1.0347020
## 3 -1.4439684 -0.7067452 -0.0685071  0.92164204 -1.2404188 -0.49104019 -0.7107079  1.4819808 -0.1274744  0.2218319 -0.5472613
## 4  1.5041399 -0.1168656  1.4235977 -1.42707970  0.9010018  0.70221043  0.4732513 -0.9035223  1.6740627  0.9941537 -0.2004004
## 5 -1.1540161  1.0266816  0.4624729 -0.70253988 -0.5392674 -0.68212631 -0.7676600 -1.5773886  0.8351858 -0.2002292 -0.6827608
## 6 -0.3378949  1.3013658 -0.2082940 -0.03714877 -0.0704557  0.01470371 -2.1596157 -1.0521083 -1.8428795  0.5551239  0.3932408
##           V23        V24        V25        V26        V27        V28        V29        V30          Y
## 1 -1.75313688 -0.7229254  1.0447042 -1.6039686  2.0969676  1.3231948  1.1824031 -0.7279445   7.298693
## 2 -1.13334262 -1.0266475 -1.2804380  2.3235545  0.3151978 -0.1939865  0.6547813 -0.6941529   7.581017
## 3  0.92501540  3.3641046 -1.4435312 -0.1692659  0.8178191 -0.8159136 -0.2457253  0.1346359  -1.802129
## 4  0.49669413  0.8165107  0.3133377 -1.0579735 -0.6183460  0.8399664 -0.8535299  0.9501371  -8.439072
## 5  0.04078674  0.1621513 -0.8827370  0.9195504  0.2737437 -1.4188221  0.9663408 -1.6644732 -19.697345
## 6  1.08758454 -0.3949437  1.1463372 -2.2894429 -1.3863564  1.1250673  0.8097635 -0.4739675   5.518526
```

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) -0.095539020 0.11465381 -0.83328253 4.075606e-01
## V1           0.971813741 0.10827616  8.97532485 3.309142e-13
```

```
## V2           1.845119496 0.10909140 16.91351926 3.026789e-26
## V3           2.967771920 0.11917493 24.90265340 3.325159e-36
## V4           3.977352815 0.09974721 39.87432829 2.217368e-49
## V5           4.896180654 0.11646398 42.04030133 6.645823e-51
## V6           5.968010120 0.10313646 57.86518359 3.259293e-60
## V7           6.894886860 0.10630410 64.86002579 1.427210e-63
## V8           8.122522334 0.12414068 65.42998024 7.878548e-64
## V9           9.063932483 0.10164167 89.17535620 5.329896e-73
## V10          9.854644027 0.10665850 92.39436434 4.707367e-74
## V11         -0.002191531 0.12149144 -0.01803856 9.856601e-01
## V12          0.086896929 0.12170879  0.71397413 4.776515e-01
## V13         -0.133754739 0.11354361 -1.17800324 2.428422e-01
## V14          0.172843150 0.10266342  1.68359044 9.678026e-02
## V15         -0.041353083 0.10698976 -0.38651441 7.003051e-01
## V16          0.021537604 0.12754430  0.16886370 8.663979e-01
## V17         -0.008724735 0.11280231 -0.07734536 9.385727e-01
## V18          0.022436651 0.12342477  0.18178402 8.562852e-01
## V19          0.005111238 0.13093213  0.03903731 9.689734e-01
## V20         -0.016985461 0.11587017 -0.14659045 8.838827e-01
## V21         -0.029036625 0.10644023 -0.27279747 7.858237e-01
## V22         -0.047155575 0.11067902 -0.42605705 6.713924e-01
## V23          0.138867759 0.09978303  1.39169718 1.684844e-01
## V24          0.033352410 0.11288630  0.29545136 7.685364e-01
## V25         -0.161808909 0.11049859 -1.46435273 1.476390e-01
## V26         -0.073389909 0.10883428 -0.67432712 5.023564e-01
## V27          0.104679858 0.10834076  0.96620939 3.373138e-01
## V28          0.008935961 0.11429556  0.07818292 9.379088e-01
## V29         -0.219333263 0.12844927 -1.70754779 9.221645e-02
## V30         -0.149895068 0.12526271 -1.19664556 2.355402e-01
```

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

```
## V11
##  11
```

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

```
## [1] 0.9856601
```

```r
our_maxp_val > alpha
```

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

```r
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        V12
## 1 -0.3930639  1.2655221  1.0200058 -0.01079562 1.4186216 -0.96335659  0.33051280 -0.39813635  1.0664855 -0.3100600 -0.9577696
## 2 -0.3601064  1.4585719  1.1913495 -1.43520296 0.3460144 -1.33248928  0.76770607  0.76945478  1.3113225  0.2511063 -1.2635435
## 3  0.1552269  0.7578593  1.1771084  0.94620380 1.7672076 -0.90237767  0.87224205  0.09748762  0.6473298 -0.2628370 -1.9601658
## 4 -2.1459879 -1.1539599  0.6775746 -1.07698523 1.3230010  1.66550443  0.07733465 -0.88770121 -2.2894576 -1.1738859  0.7465976
## 5  0.3555581 -1.1732355  0.1798234 -1.19483256 2.5763884  0.25857061  0.57758974 -1.08497847 -0.3643420  1.5646968 -1.0290683
## 6 -0.8161288 -1.0650329 -2.2839739  1.14077666 0.8288837 -0.07457638 -1.45979573 -0.54972749 -2.1946986 -0.3071779  0.4586771
##          V13         V14        V15         V16        V17        V18         V19         V20         V21         V22         V23
## 1 -0.6469668 -0.07565067  0.2202806  0.06889022  0.4005974 -0.3833696 -1.30698924  0.02868716  0.75213185 -0.29859510  1.56311496
## 2 -0.1401344  0.39106372 -0.2237693  0.47635490 -0.4697781  1.4073355 -1.07642853 -0.20909455  0.67788251 -0.04773706  0.04839978
## 3 -1.7323066  0.41065850  2.0585589  0.02601211  0.4365873 -0.9310433 -0.52385245 -0.49031394  0.21065882  1.04939420 -0.53586431
## 4  1.6892149 -0.05234469  0.4560284 -0.43746356 -1.4435148  1.0409676 -0.51375639  1.37586271 -2.00638208 -0.15844460 -1.39846299
## 5  0.4559442 -0.18116593  0.1511929 -1.41097077  1.2758122  0.9682092  0.03039255  0.71503313  0.67560794 -0.04574755  1.02708569
## 6 -0.2080526 -1.33271407  1.5473833 -0.30569123  0.6554688 -0.5553936  0.36328304 -0.93131757  0.07509536 -1.49427886  0.86333875
##          V24        V25         V26        V27        V28         V29         V30         Y
## 1  1.3114185 -0.8659288  0.42597910  1.1961159  0.2197817  0.89125268 -0.36550510  12.03594
## 2  1.6211934 -0.3601223 -0.84710178 -1.5676831 -0.6712516 -1.62375189  0.86271218  19.90090
## 3 -0.2771774 -0.2507861 -1.55653549  0.3894258 -0.2956823 -0.24686614  0.01115944  22.47213
## 4  0.8319598  0.7299175  1.24519043  1.1130822 -0.5398224  1.78207406 -0.80760644 -31.77147
## 5  1.0878605 -0.8942806 -1.80248767  0.4039988  0.9252324 -0.98815205  0.73412000  15.27436
## 6 -0.9428832 -2.0205143 -0.04574888 -1.0503945 -0.6771197  0.06331202  0.29288943 -40.42953
```

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

```
##                 Estimate Std. Error     t value     Pr(>|t|)
## (Intercept) -0.095824458 0.11274285 -0.84993826 3.982571e-01
## V1           0.971743898 0.10743148  9.04524350 2.184407e-13
## V2           1.845189905 0.10824028 17.04716537 1.275110e-26
## V3           2.967732189 0.11830069 25.08634828 1.034525e-36
## V4           3.977204959 0.09869746 40.29693170 3.586069e-50
## V5           4.896254101 0.11555869 42.37028209 1.226368e-51
## V6           5.967730942 0.10123789 58.94760076 2.111961e-61
```

```
## V7           6.894802057 0.10543905 65.39135434 1.684443e-64
## V8           8.122230555 0.12220033 66.46651861 5.475500e-65
## V9           9.063933239 0.10091328 89.81903330 4.908934e-74
## V10          9.854770536 0.10566499 93.26429530 3.597890e-75
## V12          0.086912516 0.12083356  0.71927467 4.743662e-01
## V13         -0.133934078 0.11229695 -1.19267778 2.370228e-01
## V14          0.172996457 0.10157785  1.70309240 9.298731e-02
## V15         -0.041421407 0.10615646 -0.39019205 6.975794e-01
## V16          0.021599758 0.12658408  0.17063567 8.650025e-01
## V17         -0.008615314 0.11183190 -0.07703807 9.388130e-01
## V18          0.022047347 0.12065239  0.18273444 8.555346e-01
## V19          0.005379399 0.12915323  0.04165130 9.668953e-01
## V20         -0.016953188 0.11502611 -0.14738557 8.832514e-01
## V21         -0.029236141 0.10510535 -0.27816034 7.817093e-01
## V22         -0.047394025 0.10909931 -0.43441177 6.653256e-01
## V23          0.139026626 0.09868133  1.40884431 1.633079e-01
## V24          0.033856112 0.10859435  0.31176679 7.561445e-01
## V25         -0.162286749 0.10650762 -1.52371017 1.320864e-01
## V26         -0.073649937 0.10710236 -0.68765935 4.939396e-01
## V27          0.104531619 0.10725448  0.97461308 3.331083e-01
## V28          0.008719511 0.11284941  0.07726678 9.386317e-01
## V29         -0.219379014 0.12750391 -1.72056698 8.974735e-02
## V30         -0.150015151 0.12418931 -1.20795547 2.311297e-01
```

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

```
## V19
##  18
```

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

```
## [1] 0.9668953
```

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) 100.5057 123.005 130.7432 128.3114 135.1976 184.9868   100
```

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

```
## Unit: milliseconds
##                    expr     min       lq     mean   median      uq      max neval
##  run_BE(n, p, k, alpha) 28.4988 40.70405 44.78655 43.43335 47.3383 145.3264   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) -0.08540285 0.11039708 -0.773597 4.412661e-01 -0.268944335  0.09813863                  1
## V1           0.96092702 0.09842856  9.762685 1.233814e-15  0.797283897  1.12457015                  1
## V2           1.98232226 0.11616178 17.065185 1.672723e-29  1.789196631  2.17544788                  1
## V3           2.99791622 0.11516368 26.031786 7.850566e-43  2.806450001  3.18938244                  1
## V4           3.90397275 0.10797991 36.154622 3.436634e-54  3.724449962  4.08349554                  1
## V5           4.70320025 0.10498028 44.800798 7.006985e-62  4.528664516  4.87773598                  0
## V6           6.03290521 0.13225604 45.615348 1.556769e-62  5.813021956  6.25278846                  1
## V7           7.23263153 0.11421869 63.322664 1.456846e-74  7.042736408  7.42252664                  0
## V8           7.97944692 0.10471744 76.199788 1.959902e-81  7.805348166  8.15354567                  1
## V9           9.04929255 0.10912279 82.927610 1.375191e-84  8.867869649  9.23071546                  1
## V10          9.94538386 0.11598443 85.747578 7.762292e-86  9.752553098 10.13821463                  1
## V13         -0.26299551 0.10681288 -2.462208 1.578047e-02 -0.440578046 -0.08541297                  0
## V25          0.18156571 0.10335771  1.756673 8.249208e-02  0.009727588  0.35340384                  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) -0.08681536 0.10784216  -0.8050225 4.230013e-01 -0.266109125  0.09247841                  1
## V1           0.86656034 0.12182390   7.1132210 3.017288e-10  0.664021117  1.06909957                  1
## V2           2.03970930 0.10317053  19.7702699 6.206674e-34  1.868182372  2.21123623                  1
## V3           2.98620175 0.12041499  24.7992521 3.253274e-41  2.786004923  3.18639858                  1
## V4           3.92630067 0.11235112  34.9466985 5.473368e-53  3.739510492  4.11309085                  1
## V5           5.13050295 0.10567118  48.5515822 8.381621e-65  4.954818554  5.30618735                  1
## V6           6.03798194 0.10339641  58.3964390 1.425042e-71  5.866079484  6.20988439                  1
## V7           7.02420430 0.11430121  61.4534549 1.866598e-73  6.834171984  7.21423662                  1
## V8           8.00351343 0.11086183  72.1935879 2.001723e-79  7.819199271  8.18782758                  1
## V9           9.03903645 0.10309120  87.6800032 1.141480e-86  8.867641421  9.21043147                  1
## V10         10.13226169 0.09899154 102.3548272 1.845505e-92  9.967682583 10.29684080                  1
## V20          0.18867653 0.11116556   1.6972571 9.322287e-02  0.003857414  0.37349564                  0
## V24          0.22741803 0.10461671   2.1738214 3.243461e-02  0.053486760  0.40134931                  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")
```

```r
        return(accuracy_mat)
    }
}
```

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

```r
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) -0.2603802 0.10097715 -2.578605 1.160087e-02 -0.4282605 -0.09249993                  0
## V1           0.9682821 0.12166551  7.958558 6.007317e-12  0.7660062  1.17055796                  1
## V2           2.0144663 0.10417350 19.337607 2.989896e-33  1.8412719  2.18766072                  1
## V3           3.1747540 0.11646417 27.259491 2.196737e-44  2.9811257  3.36838240                  1
## V4           3.8381399 0.10549005 36.383904 2.050824e-54  3.6627566  4.01352313                  1
## V5           4.9119647 0.09739660 50.432609 3.434498e-66  4.7500373  5.07389213                  1
## V6           6.1304840 0.11302870 54.238296 7.429831e-69  5.9425673  6.31840073                  1
## V7           7.0199648 0.10550292 66.538110 2.134926e-76  6.8445602  7.19536947                  1
## V8           8.1567172 0.09681428 84.251182 3.526573e-85  7.9957579  8.31767649                  1
## V9           8.6374796 0.10854301 79.576565 4.751391e-83  8.4570207  8.81793861                  0
## V10         10.1555268 0.11632037 87.306520 1.648026e-86  9.9621375 10.34891611                  1
## V13          0.1845535 0.10037560  1.838629 6.938178e-02  0.0176733  0.35143368                  0
## V16         -0.2146740 0.10867685 -1.975342 5.139933e-02 -0.3953555 -0.03399249                  0
```

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

```
## (Intercept)          V1          V2          V3          V4          V5          V6          V7          V8          V9         V10
##           0           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
```

```r
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
##           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
```

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

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

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

```
## (Intercept)          V1          V2          V3          V4          V5          V6          V7          V8          V9         V10
##           0           1           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
```

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

```
## (Intercept)          V1          V2          V3          V4          V5          V6          V7          V8          V9         V10
##           1           1           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
```

```r
# 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
##           0           1           1           1           1           1           1           1           1           0           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
##           1           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           1           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.

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

```
##             % Param in CI % Param Significant
## (Intercept)         85.40            14.60000
## V1                  87.40           100.00000
## V2                  87.90           100.00000
## V3                  86.90           100.00000
## V4                  88.20           100.00000
## V5                  86.50           100.00000
## V6                  87.10           100.00000
## V7                  86.50           100.00000
## V8                  86.60           100.00000
## V9                  85.10           100.00000
## V10                 88.10           100.00000
## V11                  0.00            10.40000
## V12                  0.00            12.30000
## V13                  0.00            13.20000
## V14                  0.00            12.20000
## V15                  0.00            14.60000
## V16                  0.00            13.00000
## V17                  0.00            12.10000
## V18                  0.00            11.40000
## V19                  0.00            11.80000
## V20                  0.00            14.40000
## V21                  0.00            11.60000
## V22                  0.00            11.70000
## V23                  0.00            11.50000
## V24                  0.00            13.50000
## V25                  0.00            12.30000
## V26                  0.00            12.00000
## V27                  0.00            10.90000
## V28                  0.00            10.90000
## V29                  0.00            12.20000
## V30                  0.00            12.40000
## Averages            87.03            12.33333
```

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)         81.60             18.4000
## V1                  79.80             99.8000
## V2                  82.30            100.0000
## V3                  82.60            100.0000
## V4                  81.50            100.0000
## V5                  80.20            100.0000
## V6                  81.60            100.0000
## V7                  81.40            100.0000
## V8                  80.50            100.0000
## V9                  80.60            100.0000
## V10                 81.20            100.0000
## V11                 82.30            100.0000
## V12                 81.80            100.0000
## V13                 80.20            100.0000
## V14                 82.70            100.0000
## V15                 80.50            100.0000
## V16                  0.00             14.8000
## V17                  0.00             15.2000
## V18                  0.00             16.1000
## V19                  0.00             14.8000
## V20                  0.00             15.1000
## V21                  0.00             15.7000
## V22                  0.00             13.8000
## V23                  0.00             17.0000
## V24                  0.00             15.1000
## V25                  0.00             15.5000
## V26                  0.00             15.1000
## V27                  0.00             15.5000
## V28                  0.00             16.2000
## V29                  0.00             16.3000
## V30                  0.00             16.4000
## Averages            81.28             15.6875
```

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)        85.8000            14.20000
## V1                 77.6000            87.30000
## V2                 86.3000            99.90000
## V3                 87.7000           100.00000
## V4                 87.2000           100.00000
## V5                 86.1000           100.00000
## V6                 84.8000           100.00000
## V7                 85.5000           100.00000
## V8                 85.3000           100.00000
## V9                  0.0000            12.30000
## V10                 0.0000            12.40000
## Averages           85.0625            12.96667
```

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)      97.30000              2.7000
## V1               97.80000            100.0000
## V2               98.00000            100.0000
## V3               98.40000            100.0000
## V4               97.10000            100.0000
## V5               96.20000            100.0000
## V6               98.40000            100.0000
## V7               98.20000            100.0000
## V8               96.80000            100.0000
## V9               97.80000            100.0000
## V10              97.50000            100.0000
## V11              98.40000            100.0000
## V12              97.90000            100.0000
## V13              96.20000            100.0000
## V14              98.20000            100.0000
## V15              97.20000            100.0000
## V16               0.00000              2.2000
## V17               0.00000              2.8000
## V18               0.00000              2.4000
## V19               0.00000              2.2000
## V20               0.00000              1.2000
## V21               0.00000              2.5000
## V22               0.00000              2.1000
## V23               0.00000              2.1000
## V24               0.00000              1.8000
## V25               0.00000              2.6000
## V26               0.00000              3.9000
## V27               0.00000              3.0000
## V28               0.00000              2.8000
## V29               0.00000              2.7000
## V30               0.00000              2.0000
## Averages         97.60667              2.4375
```

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
```

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)          89.5                10.5
## V1                   88.0               100.0
```

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

```
##             % Param in CI % Param Significant
## (Intercept)          90.7                 9.3
## V1                   90.2               100.0
```

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

```
##             % Param in CI % Param Significant
## (Intercept)          90.7                 9.3
## V1                   89.9               100.0
```

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

```
##             % Param in CI % Param Significant
## (Intercept)          91.0                   9
## V1                   89.6                 100
```

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.