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.

```
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,]  1.3457727 -0.2519484  1.4400756 -0.1882082  0.17834690 -0.3053808 -1.7628976 -0.05660918 -0.05115515  1.1482745  0.1621377
## [2,] -1.3887575  0.2731307 -1.1475133  1.3645432 -0.01107599  0.7647441 -2.7043179 -0.89490740 -0.90140891  0.7264604  0.9049391
## [3,]  0.2143277 -1.0984499  0.6152354 -1.3147012 -0.76347908 -1.9647542  1.8545511 -0.95440821  0.11746656  0.3870545 -0.7990878
## [4,] -0.8160944 -0.6360171 -0.1290645  0.2631148 -2.60572497 -2.5238268  0.9288696  0.07741900  1.00033606 -1.2147388 -1.2870622
## [5,] -1.9973780  0.3683774  0.2547245  0.5173848 -0.73509359 -0.7887287  1.3689973 -1.88562527  1.55507714  0.1189676 -0.3564385
## [6,]  0.1607566 -0.2018441 -0.4218556 -0.4111477 -0.07571910  1.9007648 -1.5891276  0.29662608  0.57704012 -0.4265038 -2.0494143
##            [,12]      [,13]       [,14]      [,15]      [,16]       [,17]       [,18]      [,19]       [,20]      [,21]      [,22]
## [1,] -0.02271983  2.4769497  0.76055165  1.2234809 -0.1970043  0.90166194  0.93899305 -0.6213765  0.15312706 -0.3146953 -1.4539011
## [2,]  0.74177126 -1.0513684  1.48225215 -0.4944622  0.2808099  0.04403863 -1.17944536  0.4569481  0.03984701  2.1980269  1.6168933
## [3,]  2.38803477  0.2954280  0.51564750 -0.3189753 -0.4535812 -1.30801538 -1.90798542  0.1779656  0.40564443  0.8918460 -1.4225436
## [4,] -0.71192712  0.8716671 -0.78918793 -1.8440847  1.3589827 -1.47953914  1.17217614 -0.2885933  1.20515842  2.8782548 -0.6636696
## [5,]  0.36848640  0.6465266  2.58106161  0.2513671  1.3840347  0.18359268 -0.03842933  1.9286059 -0.85886391  1.7065870  1.0596359
## [6,]  1.75980944  1.6221943  0.02557195 -2.6671420  2.0190917 -0.77743576  0.40722695 -0.7119098 -0.31744946 -0.1882473 -1.0212623
##            [,23]       [,24]       [,25]      [,26]      [,27]      [,28]       [,29]      [,30]
## [1,] -0.51290852  0.52482426  0.13837908 -0.9220012 -0.9441332 -1.6083609  0.03342953  0.8474675
## [2,] -0.03841999  0.34858034 -0.07365714 -0.4097705 -0.1794320  2.2870768  1.56111956  0.5685858
## [3,] -0.49546459 -0.03602615 -0.17538050 -0.2157165 -0.8856639  0.1166230 -0.58684657  0.8111844
## [4,]  1.14367465 -0.70056936  0.72139508  0.3886596 -0.5277452  0.3154731 -0.16063914 -0.6322399
## [5,]  1.57808967 -0.82516705  0.08726838 -0.7642209 -1.5376887 -0.4694868  2.34448911 -2.4541902
## [6,]  0.82268148 -0.80930310 -1.46076168  1.2139930  0.2701642  1.2179196 -0.91779427  0.2570692
```

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:

```
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]   2.5637648 -24.4443646 -10.3838260 -25.3045467   1.8041811   0.9555694  34.2213959  14.4323954 -12.8047147 -39.7355902
##  [11]   8.1120691  -4.4811454  -9.0898433 -12.2684011  -1.1234185  32.3722509 -37.1325161 -21.4141611 -25.2903252   8.7270285
##  [21]  -9.3311675 -14.9519456  14.5887520 -13.2205387  22.2321578 -14.1456971  -4.3513766   9.6210308  35.1542966   2.8555341
##  [31]  10.4086286  -8.0582320 -37.4385780 -23.7802098  -4.3989424 -32.6881714  18.0292383  15.5737573  20.0451971   2.9422333
##  [41]  -1.8805761  -1.6246919 -14.2019727 -12.3515506   5.5677029  14.0554924  -4.5848074 -24.1538786  18.4988709  -2.9779601
##  [51]  -8.3269366  -7.5268666   0.6042875   0.6041442   8.1513484 -15.9051572   4.4861103  -9.3681915 -12.2247414 -23.4144981
##  [61]   3.2707126 -36.1535354   2.1078173   4.2416591  -0.7811478   4.7456017   6.0433014  30.0165224   9.8292553 -48.2511030
##  [71]  -2.6307642  -4.5522645 -23.5097859 -43.5051010 -55.8487222 -13.3976602 -30.9632297  -3.6382894   6.3601183  20.6736904
##  [81] -19.4818417 -33.0412744   8.0316182   3.3710446  25.5289059 -16.1382653  -5.5424138  22.6180111   8.4148236   5.4869156
##  [91]  -4.2587088   3.5177368  26.3644974 -16.3725773  15.0232029   3.2074848 -36.8787186   6.3595817  16.5831459   1.2160936
```

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  0.7304422 -0.94828102  1.4440109 -0.092392169  0.6062674  1.3002989 -0.538306202  0.9626231 -1.5484947 -1.3336221  1.0203419
## 2  0.9049910 -0.16196932 -1.0649329  0.009233162 -0.6535056  1.4444661 -0.286886342  0.3300116 -0.5208928  0.8151185  0.1179580
## 3 -0.3009185  0.13378864  0.4132496 -1.137342002  0.1844507  1.2686416 -1.712455546 -0.6303348 -0.4139395  1.5789075 -0.6382983
## 4  0.8408360 -1.22032731  1.1258816 -1.818266178 -0.5518545 -1.9519566  0.006495802 -0.1132621  0.1650425  1.3090212 -1.2847538
## 5 -1.9739751  0.62362611 -0.4947085  0.118327057  0.6251286 -0.7739649  0.098670244 -0.5138421  1.4079106  1.1792084 -0.1367348
## 6  0.5590930 -0.06390567  0.2532904  0.370231399 -0.3421857 -0.6977942  0.905934717  0.2809539  1.4771002 -0.3998452  0.9149923
##          V12         V13        V14         V15        V16         V17           V18       V19         V20        V21        V22
## 1  1.5544072 -1.21791356 -0.8426599 -0.59300282 -1.2284641  0.07489668  0.7656073668 1.1936761 -0.28423250 -1.4397804 -0.8840100
## 2 -0.1533875 -4.03445871 -0.1610376 -0.41063619 -2.1088047  0.04911204  0.0002032093 0.3575833  0.03264283 -0.4414146 -1.3138221
## 3 -1.5571590 -0.44539120  0.5953881 -1.42725199  0.3199459 -0.22622183 -1.1531905003 0.9023302 -1.54266488 -1.3251282  0.1641208
## 4  0.4853816  0.04269614 -0.5887561 -0.64607917 -1.2529503 -0.98436588 -0.1467386181 0.2211253 -0.98047922 -0.6446600  1.2655758
## 5 -0.6785473  0.66127177  0.7993235  0.86414092 -0.2756293  1.34621490 -0.4583440084 2.0979192  0.51241033 -0.8066393 -1.1876884
## 6 -1.1353748 -0.35895542 -0.6948601 -0.07313945  0.7400210  2.43244546 -0.0035264151 0.6839178  1.18883761 -0.1869278  1.2582490
##          V23       V24        V25        V26        V27         V28        V29        V30          Y
## 1 -0.8347934 0.6849468 -1.0718936 -0.9963465  0.7151530  0.65486621 -0.7052068  0.6095125 -10.378149
## 2 -0.6872868 0.6707568 -0.8344908  0.5791586  1.4372235 -0.10081159  0.5771682  0.9867581   7.183991
## 3  0.2794213 0.1802998 -0.1383849  0.1185781 -0.4230059 -0.70984005 -1.4540300  0.2981932  -1.387395
## 4  2.7121543 1.6583619 -0.9964258 -0.5205006  3.0719191  0.61207942  0.8200157 -0.7992795  -4.929567
## 5  0.8435211 0.8639785 -0.6481995  1.1871849  0.5667063  0.09702335 -0.2015709  0.5210937  17.577307
## 6  2.1784746 0.0407561 -0.6895091 -0.3697091  0.4274489 -0.05865210 -0.4307578  0.4021275  14.923892
```

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.

```r
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.

```r
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) -0.005774377  0.1287380 -0.04485372 9.643535e-01
## V1           1.042313101  0.1150132  9.06255534 2.296525e-13
## V2           2.341238787  0.1408336 16.62414857 7.900119e-26
## V3           2.731471816  0.1580827 17.27875290 9.148482e-27
## V4           3.978749262  0.1195910 33.26962770 3.210807e-44
## V5           5.186651537  0.1293458 40.09912303 1.528431e-49
## V6           5.779658423  0.1434585 40.28802404 1.119691e-49
## V7           7.370630295  0.1357299 54.30367817 2.378758e-58
## V8           7.957701170  0.1202797 66.15996962 3.707765e-64
## V9           9.018677303  0.1196065 75.40287694 5.049804e-68
## V10          9.977261055  0.1296698 76.94360434 1.271044e-68
## V11         -0.074766056  0.1328578 -0.56275233 5.754275e-01
## V12         -0.013363706  0.1479791 -0.09030804 9.283041e-01
## V13         -0.067135932  0.1483492 -0.45255345 6.522900e-01
## V14         -0.069607639  0.1139587 -0.61081480 5.433291e-01
## V15         -0.002187485  0.1282401 -0.01705773 9.864398e-01
## V16          0.228003575  0.1216596  1.87411008 6.514953e-02
## V17          0.012316062  0.1422643  0.08657170 9.312627e-01
## V18         -0.194529145  0.1276616 -1.52378758 1.321322e-01
## V19          0.110570603  0.1377799  0.80251642 4.250098e-01
## V20          0.006426615  0.1275648  0.05037922 9.599658e-01
## V21          0.307965204  0.1433512  2.14832605 3.520047e-02
## V22          0.065868627  0.1218368  0.54063013 5.905035e-01
## V23          0.023770783  0.1438195  0.16528205 8.692053e-01
## V24         -0.008976948  0.1351433 -0.06642540 9.472312e-01
## V25         -0.088889537  0.1291209 -0.68842089 4.934959e-01
## V26          0.006277967  0.1212991  0.05175610 9.588726e-01
## V27          0.094463148  0.1293816  0.73011274 4.677929e-01
## V28         -0.053912846  0.1376026 -0.39180103 6.964127e-01
## V29         -0.257326324  0.1413609 -1.82034996 7.304254e-02
## V30         -0.128569922  0.1342184 -0.95791560 3.414498e-01
```

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

```
## V15
```

```
##  15
```

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

```
## [1] 0.9864398
```

```
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  0.1361977  0.803586468  0.9300434 -0.1756002 -0.2334112 -1.5441503  0.1983349 -0.1882036 -0.3091029  1.4653610 -0.1508540
## 2  0.1528664 -0.237992500  0.6050701  1.1942358 -2.1937080 -0.6679149  0.2241225  0.5732466 -0.9717054 -0.6242643  0.2196248
## 3 -0.3266915 -0.006310208 -1.7233081 -0.1164170 -1.5806212  0.5377172  0.6231020 -1.9111771  0.3483734  0.9588830 -0.2473908
## 4  0.5863786  1.214633274 -0.5309240 -0.3886321 -0.5845201 -1.2165417 -0.8978291  0.6893498 -1.3488121 -0.4089663  1.7942796
## 5  0.7703965  1.434386531 -1.1711171  0.9299644  1.3285294  0.9013353 -0.2751991  1.1308981  0.1930897 -0.7274629 -0.9189177
## 6 -0.6470730 -1.128338213 -0.7208803  0.4148619  1.8558624  1.3488814  0.2426671  0.5696905 -0.8793219  1.2827929  0.7211905
##          V12        V13         V14         V16         V17        V18         V19         V20        V21        V22        V23
## 1  0.6562876  0.0316956 -0.05164768 -0.13919803 -0.77918366 -0.2959444  0.39274804 -0.51520139 -0.2912415  0.9975193 -0.8250171
## 2  0.8454906  0.6237651  0.02659827  1.57641672  0.53097286 -0.1403377 -0.44514707  0.03291677  0.2788719  0.2250550  0.5172108
## 3 -0.4547467 -0.2145384 -0.24065125 -1.27365290  0.76544256  1.6556858  1.75706693 -0.71202251 -0.7629958 -1.2174865  1.2603418
## 4  0.1225576  0.5509641  1.58825670  0.19224061  0.04927247 -1.7678158 -1.68163272 -1.32060384 -0.4436180 -0.8109833  1.6845687
## 5 -1.5718037  0.1434318 -1.00453824 -0.34482471  2.80683750  0.3129098 -0.03332275  0.41190188  0.8310107  1.2796095  0.1656147
## 6  0.3726821  0.5277212 -0.26263234 -0.09360914  0.55427494 -0.9078137  0.31688839 -0.02433400 -0.1431550 -0.1782643  0.9288343
##          V24        V25        V26         V27        V28        V29        V30          Y
## 1  1.8922611  1.1180936 -0.1434915  0.69239806  0.3386578  0.6923221  0.3061858   5.945449
## 2  0.1638097  0.2319761  0.1136502 -0.38370704  0.2793635 -0.4708293  0.6830737 -17.872628
## 3 -2.0238588 -0.7243414 -0.4302638 -0.39685478 -0.3359456  0.5367982 -1.0339045  -8.518404
## 4  1.7111293  1.2139399 -0.2262856  0.09717662 -0.6519251  0.1393032 -0.3391796 -26.385867
## 5  1.0760860  1.4296989  0.5178884 -2.13331206  0.6480576 -0.4480785 -1.6462832  16.754984
## 6 -0.1688538 -0.6321181  0.7260995  0.19839273  0.2616659 -0.4381179 -1.2895630  25.897269
```

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

```
##                  Estimate Std. Error     t value     Pr(>|t|)
## (Intercept) -0.005620572  0.1275015 -0.04408240 9.649642e-01
## V1           1.041952285  0.1122411  9.28316153 8.016883e-14
## V2           2.340348845  0.1298759 18.01988119 5.338273e-28
## V3           2.732295421  0.1494498 18.28235943 2.310768e-28
## V4           3.979082870  0.1171355 33.96991488 3.139803e-45
## V5           5.186504190  0.1281321 40.47778604 2.654659e-50
## V6           5.779796657  0.1422030 40.64470287 2.013443e-50
## V7           7.370916556  0.1337231 55.12072336 2.102833e-59
## V8           7.958098963  0.1171517 67.92987188 1.219948e-65
## V9           9.018884734  0.1181340 76.34450725 3.827170e-69
## V10          9.977447064  0.1282845 77.77595124 1.058374e-69
## V11         -0.075001224  0.1311936 -0.57168355 5.693676e-01
## V12         -0.013509941  0.1466719 -0.09210995 9.268738e-01
## V13         -0.066883367  0.1465506 -0.45638415 6.495267e-01
## V14         -0.069272087  0.1114435 -0.62158904 5.362313e-01
## V16          0.227949907  0.1207474  1.88782484 6.319484e-02
## V17          0.012641241  0.1399709  0.09031333 9.282962e-01
## V18         -0.194495566  0.1267316 -1.53470410 1.293643e-01
## V19          0.110377913  0.1363319  0.80962627 4.208987e-01
## V20          0.006395280  0.1266375  0.05050069 9.598672e-01
## V21          0.307372029  0.1380728  2.22615931 2.922453e-02
## V22          0.065797060  0.1208919  0.54426366 5.879896e-01
## V23          0.023584450  0.1423764  0.16564864 8.689110e-01
## V24         -0.009189909  0.1336010 -0.06878620 9.453559e-01
## V25         -0.088720905  0.1278193 -0.69411204 4.899092e-01
## V26          0.006656470  0.1183976  0.05622133 9.553257e-01
## V27          0.094208524  0.1275966  0.73833106 4.627816e-01
## V28         -0.053936936  0.1366093 -0.39482625 6.941716e-01
## V29         -0.257086556  0.1396523 -1.84090469 6.987349e-02
## V30         -0.128069339  0.1300323 -0.98490414 3.280633e-01
```

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

```
## V20
##  19
```

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

```
## [1] 0.9598672
```

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.

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) -0.01331975  0.1132880 -0.1175742 9.066730e-01 -0.201644561  0.1750051                  1
## V1           0.91313266  0.1028633  8.8771430 7.349390e-14  0.742137359  1.0841280                  1
## V2           1.93268834  0.1228015 15.7383090 2.561414e-27  1.728548729  2.1368279                  1
## V3           2.98571433  0.1090180 27.3873621 7.672717e-45  2.804487888  3.1669408                  1
## V4           4.09663402  0.1088063 37.6506980 4.781625e-56  3.915759387  4.2775086                  1
## V5           5.09726267  0.1006317 50.6526560 6.973338e-67  4.929977166  5.2645482                  1
## V6           5.89357399  0.1218619 48.3627462 3.554868e-65  5.690996444  6.0961515                  1
## V7           6.95141876  0.1049896 66.2105650 7.388067e-77  6.776888922  7.1259486                  1
## V8           7.92426509  0.1092693 72.5204817 2.828720e-80  7.742620769  8.1059094                  1
## V9           8.79586957  0.1286906 68.3489742 4.749361e-78  8.581940259  9.0097989                  1
## V10          9.90958513  0.1154669 85.8218719 1.272463e-86  9.717638270 10.1015320                  1
## V19          0.18069284  0.1067976  1.6919178 9.419995e-02  0.003157342  0.3582283                  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) -0.02233093 0.09671406 -0.2308964 8.179501e-01 -0.18316433  0.13850248                  1
## V1           0.84025203 0.09118064  9.2152456 1.979418e-14  0.68862058  0.99188348                  0
## V2           1.93870510 0.09522063 20.3601382 1.935005e-34  1.78035525  2.09705496                  1
## V3           3.11082612 0.09618905 32.3407507 1.520185e-49  2.95086579  3.27078644                  1
## V4           3.85791257 0.09334164 41.3310969 4.642160e-58  3.70268742  4.01313772                  1
## V5           5.05712256 0.10437278 48.4525055 1.085055e-63  4.88355288  5.23069225                  1
## V6           6.12904372 0.12641949 48.4817948 1.032556e-63  5.91881083  6.33927662                  1
## V7           7.02609183 0.08892266 79.0135150 2.426084e-81  6.87821536  7.17396830                  1
## V8           7.83083629 0.10432691 75.0605603 1.790799e-79  7.65734289  8.00432970                  1
## V9           8.91133747 0.09584122 92.9802128 2.783244e-87  8.75195557  9.07071937                  1
## V10         10.01879375 0.10033131 99.8570981 6.826166e-90  9.85194493 10.18564256                  1
## V14         -0.17754699 0.09579322 -1.8534401 6.728865e-02 -0.33684905 -0.01824492                  0
## V16          0.23881491 0.09851247  2.4242099 1.746136e-02  0.07499079  0.40263904                  0
## V20          0.33388792 0.10036095  3.3268709 1.298898e-03  0.16698982  0.50078602                  0
## V30         -0.19695546 0.08484627 -2.3213215 2.266318e-02 -0.33805297 -0.05585794                  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.

```r
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))
        accuracy_mat <- rbind(accuracy_mat, c(mean(accuracy_mat[2:(k + 1), 1]), mean(accuracy_mat[c(1, (k + 2):(p + 1)), 2])))
        colnames(accuracy_mat) <- c("% Param in CI", "% Param Significant")
        rownames(accuracy_mat)[p + 2] <- "Averages"
        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) 0.0423799 0.10004216   0.4236204 6.728763e-01 -0.12392559  0.2086854                  1
## V1          1.1133988 0.10676589  10.4284129 4.766482e-17  0.93591612  1.2908816                  1
## V2          1.9916094 0.10391394  19.1659499 3.670119e-33  1.81886762  2.1643511                  1
## V3          2.8827276 0.10869938  26.5201856 9.620556e-44  2.70203078  3.0634245                  1
## V4          4.2165739 0.10215113  41.2778011 2.244104e-59  4.04676260  4.3863853                  0
## V5          4.8103588 0.11101730  43.3298134 3.827140e-61  4.62580876  4.9949089                  0
## V6          6.0113410 0.11176202  53.7869766 4.167605e-69  5.82555295  6.1971290                  1
## V7          7.0209582 0.10613256  66.1527267 7.966732e-77  6.84452828  7.1973881                  1
## V8          7.9668242 0.10037321  79.3720161 1.129380e-83  7.79996837  8.1336800                  1
## V9          9.0767587 0.08785596 103.3140874 1.214810e-93  8.93071094  9.2228064                  1
## V10         9.9584282 0.10153180  98.0818597 1.131945e-91  9.78964641 10.1272100                  1
## V22         0.2100560 0.10174105   2.0646141 4.190296e-02  0.04092636  0.3791856                  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           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
##           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
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           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; sig_freq
```

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

## (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           0           0           0           0           0           0           0           0           0
##         V22         V23         V24         V25         V26         V27         V28         V29         V30
##           1           0           0           0           0           0           0           0           0
```

Finally, 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)         86.80            13.20000
## V1                  85.40           100.00000
## V2                  87.20           100.00000
## V3                  88.20           100.00000
## V4                  89.00           100.00000
## V5                  87.50           100.00000
## V6                  87.20           100.00000
## V7                  87.20           100.00000
## V8                  86.00           100.00000
## V9                  87.00           100.00000
## V10                 85.70           100.00000
## V11                  0.00            13.50000
## V12                  0.00            13.40000
## V13                  0.00            13.80000
## V14                  0.00            13.90000
## V15                  0.00            13.90000
## V16                  0.00            11.20000
## V17                  0.00            11.70000
## V18                  0.00            12.40000
## V19                  0.00            12.50000
## V20                  0.00            11.40000
## V21                  0.00            11.90000
## V22                  0.00            13.50000
## V23                  0.00            11.20000
## V24                  0.00            11.90000
## V25                  0.00            11.80000
## V26                  0.00            12.90000
## V27                  0.00            13.10000
## V28                  0.00            12.30000
## V29                  0.00            11.40000
## V30                  0.00            11.70000
## Averages            87.04            12.50476
```

One would expect that the model will be more accurate if you give it more data. We originally gave it 100 data points. Let's see what happens if we halve that to $n = 50$.

```
n <- 50
output <- run_simulation(n, p, k, alpha, m)
output
```

```
##             % Param in CI % Param Significant
## (Intercept)         80.80             19.2000
## V1                  80.60             99.9000
## V2                  78.50            100.0000
## V3                  80.70            100.0000
## V4                  79.20            100.0000
## V5                  80.40            100.0000
## V6                  81.30            100.0000
## V7                  80.90            100.0000
## V8                  79.20            100.0000
## V9                  79.10            100.0000
## V10                 79.90            100.0000
## V11                  0.00             16.2000
## V12                  0.00             17.2000
## V13                  0.00             15.2000
## V14                  0.00             15.2000
## V15                  0.00             16.2000
## V16                  0.00             17.8000
## V17                  0.00             18.8000
## V18                  0.00             17.5000
## V19                  0.00             16.2000
## V20                  0.00             17.1000
## V21                  0.00             17.2000
## V22                  0.00             15.9000
## V23                  0.00             15.6000
## V24                  0.00             18.0000
## V25                  0.00             15.9000
## V26                  0.00             15.2000
## V27                  0.00             17.4000
## V28                  0.00             18.3000
## V29                  0.00             14.7000
## V30                  0.00             17.2000
## Averages            79.98             16.7619
```

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)          86.9                13.1
## V1                   80.5                89.3
## V2                   85.7                99.9
## V3                   84.9                99.9
## V4                   87.3               100.0
## V5                   86.3               100.0
## V6                   85.6               100.0
## V7                   86.8               100.0
## V8                   86.9               100.0
## V9                    0.0                11.3
## V10                   0.0                12.8
## Averages             85.5                12.4
```

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<-0.01
output <- run_simulation(n,p,k,alpha,m)
output
```

```
##             % Param in CI % Param Significant
## (Intercept)      98.70000             1.30000
## V1               98.70000           100.00000
## V2               98.80000           100.00000
## V3               99.20000           100.00000
## V4               99.30000           100.00000
## V5               98.90000           100.00000
## V6               98.60000           100.00000
## V7               98.20000           100.00000
## V8               98.60000           100.00000
## V9               98.30000           100.00000
## V10              98.70000           100.00000
## V11              99.20000           100.00000
## V12              98.90000           100.00000
## V13              99.10000           100.00000
## V14              99.20000           100.00000
## V15              98.40000           100.00000
## V16               0.00000             1.20000
## V17               0.00000             0.60000
## V18               0.00000             1.10000
## V19               0.00000             1.00000
## V20               0.00000             0.80000
## V21               0.00000             1.30000
## V22               0.00000             0.70000
## V23               0.00000             0.60000
## V24               0.00000             1.00000
## V25               0.00000             0.80000
## V26               0.00000             1.10000
## V27               0.00000             0.90000
## V28               0.00000             1.00000
## V29               0.00000             0.80000
## V30               0.00000             0.90000
## Averages         98.80667             0.94375
```

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)      94.10000            5.900000
## V1               94.38000          100.000000
## V2               93.99000          100.000000
## V3               93.94000          100.000000
## V4               94.02000          100.000000
## V5               94.48000          100.000000
## V6               94.38000          100.000000
## V7               94.15000          100.000000
## V8               94.53000          100.000000
## V9               94.15000          100.000000
## V10              94.19000          100.000000
## V11              94.45000          100.000000
## V12              94.25000          100.000000
## V13              94.28000          100.000000
## V14              94.29000          100.000000
## V15              93.73000          100.000000
## V16              93.86000          100.000000
## V17              94.33000          100.000000
## V18              94.36000          100.000000
## V19              94.38000          100.000000
## V20              94.12000          100.000000
## V21              94.51000          100.000000
## V22              94.11000          100.000000
## V23              94.13000          100.000000
## V24              94.12000          100.000000
## V25              94.36000          100.000000
## V26              94.28000          100.000000
## V27              94.12000          100.000000
## V28              94.01000          100.000000
## V29              94.24000          100.000000
## V30              93.73000          100.000000
## V31              94.31000          100.000000
## V32              93.94000          100.000000
## V33              94.00000          100.000000
## V34              93.81000          100.000000
## V35              93.92000          100.000000
```

```
## V36                    0.00000              5.570000
## V37                    0.00000              5.490000
## V38                    0.00000              5.700000
## V39                    0.00000              5.430000
## V40                    0.00000              5.870000
## V41                    0.00000              5.350000
## V42                    0.00000              5.570000
## V43                    0.00000              5.900000
## V44                    0.00000              5.270000
## V45                    0.00000              5.330000
## V46                    0.00000              5.540000
## V47                    0.00000              5.610000
## V48                    0.00000              5.530000
## V49                    0.00000              5.870000
## V50                    0.00000              5.630000
## V51                    0.00000              5.210000
## V52                    0.00000              5.610000
## V53                    0.00000              5.700000
## V54                    0.00000              5.580000
## V55                    0.00000              5.290000
## V56                    0.00000              5.560000
## V57                    0.00000              5.620000
## V58                    0.00000              5.610000
## V59                    0.00000              5.490000
## V60                    0.00000              5.590000
## V61                    0.00000              5.360000
## V62                    0.00000              5.540000
## V63                    0.00000              5.280000
## V64                    0.00000              5.380000
## V65                    0.00000              5.420000
## V66                    0.00000              5.510000
## V67                    0.00000              5.530000
## V68                    0.00000              5.580000
## V69                    0.00000              5.450000
## V70                    0.00000              5.570000
## V71                    0.00000              5.350000
## V72                    0.00000              5.360000
## V73                    0.00000              5.480000
## V74                    0.00000              5.370000
## V75                    0.00000              5.380000
## V76                    0.00000              5.540000
## V77                    0.00000              5.840000
## V78                    0.00000              5.740000
## V79                    0.00000              5.530000
```

```
## V80                0.00000            5.850000
## V81                0.00000            5.730000
## V82                0.00000            5.710000
## V83                0.00000            5.490000
## V84                0.00000            5.410000
## V85                0.00000            6.050000
## V86                0.00000            5.460000
## V87                0.00000            6.100000
## V88                0.00000            5.760000
## V89                0.00000            5.570000
## V90                0.00000            5.630000
## V91                0.00000            5.430000
## V92                0.00000            5.730000
## V93                0.00000            5.710000
## V94                0.00000            5.500000
## V95                0.00000            5.830000
## V96                0.00000            5.780000
## V97                0.00000            5.530000
## V98                0.00000            5.260000
## V99                0.00000            5.190000
## V100               0.00000            5.450000
## Averages          94.16714            5.563636
```