# Royals Coding Challenge

Kenny Miller

12/26/2021

*Goal 1: Generate and evaluate a model to predict the probability of a successful tag up (run scores) on a batted ball Goal 2: Create a UI to allow users to input various information and return a "Go" or "No Go" decision to send a runner Goal 3: Grade outfielders via three tiers (similar to a stoplight) of whether you can run against them (20-80 scale as well for how well they control the run game will be inverse when compared to the tiers)*

---

## Goal 1 Steps/Process

---

Step 1: Data Cleaning/Validation

```
summary(tags)
```

```
##   fielder_pos        inning      batting_run_differential  outs_on_play
##  Min.   :7.000   Min.   :0.000   Min.   :-19.0000         Min.   :-1.00000
##  1st Qu.:7.000   1st Qu.:3.000   1st Qu.: -1.0000         1st Qu.: 0.00000
##  Median :8.000   Median :5.000   Median :  0.0000         Median : 0.00000
##  Mean   :8.036   Mean   :4.731   Mean   :  0.4935         Mean   : 0.01081
##  3rd Qu.:9.000   3rd Qu.:7.000   3rd Qu.:  2.0000         3rd Qu.: 0.00000
##  Max.   :9.000   Max.   :9.000   Max.   : 19.0000         Max.   : 1.00000
##
##   runs_on_play        throw           exchange       throw_dist
##  Min.   :0.0000   Min.   : 26.28   Min.   :0.070   Min.   :  0.0
##  1st Qu.:1.0000   1st Qu.: 72.91   1st Qu.:1.034   1st Qu.:161.0
##  Median :1.0000   Median : 81.70   Median :1.170   Median :187.4
##  Mean   :0.7782   Mean   : 79.81   Mean   :1.337   Mean   :188.3
##  3rd Qu.:1.0000   3rd Qu.: 88.34   3rd Qu.:1.469   3rd Qu.:218.8
##  Max.   :1.0000   Max.   :104.70   Max.   :3.840   Max.   :323.7
##                   NA's   :2675     NA's   :725     NA's   :606
##  time_to_catch     ball_vx_0          ball_vz_0        ball_vy_0
##  Min.   :2.069   Min.   :-53.31399   Min.   :-14.65   Min.   :-87.01
##  1st Qu.:3.871   1st Qu.:-14.72452   1st Qu.: 35.73   1st Qu.: 60.27
##  Median :4.832   Median : -0.07136   Median : 46.05   Median : 71.12
##  Mean   :4.742   Mean   : -0.10686   Mean   : 46.90   Mean   : 69.92
##  3rd Qu.:5.606   3rd Qu.: 14.67701   3rd Qu.: 57.62   3rd Qu.: 80.69
##  Max.   :8.567   Max.   : 47.27419   Max.   : 85.00   Max.   :108.24
##  NA's   :24      NA's   :110         NA's   :110      NA's   :110
```

```
##     ball_v_0        ball_ax_0          ball_az_0         ball_ay_0
##  Min.   : 60.30   Min.   :-26.8376   Min.   :-52.979   Min.   :-69.61
##  1st Qu.: 82.88   1st Qu.: -0.6118   1st Qu.:-26.182   1st Qu.:-34.37
##  Median : 89.21   Median :  0.8750   Median :-22.908   Median :-30.76
##  Mean   : 88.61   Mean   :  0.7534   Mean   :-23.714   Mean   :-30.58
##  3rd Qu.: 94.76   3rd Qu.:  1.7865   3rd Qu.:-20.202   3rd Qu.:-27.14
##  Max.   :112.25   Max.   : 27.0154   Max.   : -7.141   Max.   : 20.65
##  NA's   :112      NA's   :110        NA's   :110       NA's   :110
##   ball_apex_x        ball_apex_y      ball_apex_z      player_px_7_0
##  Min.   :-129.055   Min.   :  2.42   Min.   :  2.142   Min.   :-172.36
##  1st Qu.: -44.939   1st Qu.:140.99   1st Qu.: 52.640   1st Qu.:-139.96
##  Median :   2.410   Median :163.16   Median : 79.829   Median :-132.56
##  Mean   :   1.324   Mean   :163.71   Mean   : 82.949   Mean   :-131.84
##  3rd Qu.:  48.382   3rd Qu.:186.68   3rd Qu.:110.214   3rd Qu.:-124.55
##  Max.   : 133.730   Max.   :259.81   Max.   :190.401   Max.   :  41.62
##  NA's   :107        NA's   :107      NA's   :107       NA's   :1856
##  player_py_7_0    player_px_8_0      player_py_8_0    player_px_9_0
##  Min.   : 67.71   Min.   :-133.658   Min.   : 86.59   Min.   :-37.8
##  1st Qu.:251.56   1st Qu.: -14.956   1st Qu.:307.06   1st Qu.:125.8
##  Median :261.20   Median :   4.332   Median :315.28   Median :133.2
##  Mean   :260.23   Mean   :   1.824   Mean   :314.34   Mean   :132.6
##  3rd Qu.:269.83   3rd Qu.:  18.526   3rd Qu.:322.51   3rd Qu.:140.2
##  Max.   :309.58   Max.   : 127.267   Max.   :356.67   Max.   :171.6
##  NA's   :1856     NA's   :1856       NA's   :1856     NA's   :1856
##  player_py_9_0    landing_location_x landing_location_y  launch_angle
##  Min.   : 97.87   Min.   :-245.671   Min.   :  1.773    Min.   :-73.53
##  1st Qu.:249.72   1st Qu.: -90.976   1st Qu.:243.628    1st Qu.: 24.45
##  Median :258.49   Median :   6.768   Median :281.940    Median : 32.69
##  Mean   :258.21   Mean   :   4.781   Mean   :282.169    Mean   : 33.40
##  3rd Qu.:267.36   3rd Qu.: 101.612   3rd Qu.:322.114    3rd Qu.: 41.71
##  Max.   :309.15   Max.   : 265.284   Max.   :455.924    Max.   : 74.13
##  NA's   :1856     NA's   :107        NA's   :107        NA's   :107
##  launch_direction      launch_speed    runner_season_top_velo   runner_id
##  Min.   :-142.58692   Min.   : 43.83   Min.   : 5.135         Min.   :  1.0
##  1st Qu.: -12.09175   1st Qu.: 84.58   1st Qu.: 8.628         1st Qu.:238.0
##  Median :  -0.22584   Median : 90.84   Median : 8.967         Median :429.0
##  Mean   :  -0.08094   Mean   : 90.21   Mean   : 8.913         Mean   :447.2
##  3rd Qu.:  12.22973   3rd Qu.: 96.31   3rd Qu.: 9.259         3rd Qu.:646.5
##  Max.   :  41.91718   Max.   :114.20   Max.   :10.933         Max.   :970.0
##  NA's   :107          NA's   :107
##    fielder_id    fielding_team_id hitting_team_id    play_id
##  Min.   :  1.0   Min.   : 1.00    Min.   : 1.00   Min.   :   1
##  1st Qu.:137.0   1st Qu.: 8.00    1st Qu.: 8.00   1st Qu.:2058
##  Median :253.0   Median :15.00    Median :15.00   Median :4116
##  Mean   :262.8   Mean   :15.48    Mean   :15.47   Mean   :4116
##  3rd Qu.:380.0   3rd Qu.:23.00    3rd Qu.:23.00   3rd Qu.:6174
##  Max.   :564.0   Max.   :30.00    Max.   :30.00   Max.   :8231
##
```

```
sum(complete.cases(tags))/nrow(tags_original) # can maintain about 56% of the original data
```

```
## [1] 0.5572834
```

```r
tags <- tags[complete.cases(tags),]
# lets move forward with only the cases where we have all of the data
summary(tags)
```

```
##   fielder_pos        inning      batting_run_differential  outs_on_play
##  Min.   :7.000   Min.   :0.000   Min.   :-19.0000         Min.   :-1.00000
##  1st Qu.:7.000   1st Qu.:3.000   1st Qu.: -1.0000         1st Qu.: 0.00000
##  Median :8.000   Median :5.000   Median :  0.0000         Median : 0.00000
##  Mean   :8.028   Mean   :4.682   Mean   :  0.5075         Mean   : 0.01308
##  3rd Qu.:9.000   3rd Qu.:7.000   3rd Qu.:  2.0000         3rd Qu.: 0.00000
##  Max.   :9.000   Max.   :9.000   Max.   : 19.0000         Max.   : 1.00000
##   runs_on_play        throw           exchange       throw_dist
##  Min.   :0.0000   Min.   : 30.30   Min.   :0.070   Min.   :  7.516
##  1st Qu.:1.0000   1st Qu.: 73.42   1st Qu.:1.034   1st Qu.:154.827
##  Median :1.0000   Median : 82.20   Median :1.200   Median :178.119
##  Mean   :0.7713   Mean   : 80.36   Mean   :1.352   Mean   :177.843
##  3rd Qu.:1.0000   3rd Qu.: 88.70   3rd Qu.:1.500   3rd Qu.:203.688
##  Max.   :1.0000   Max.   :104.70   Max.   :3.840   Max.   :320.081
##  time_to_catch     ball_vx_0          ball_vz_0        ball_vy_0
##  Min.   :2.069   Min.   :-53.3140   Min.   :-13.84   Min.   :-87.01
##  1st Qu.:3.671   1st Qu.:-13.9562   1st Qu.: 33.89   1st Qu.: 61.84
##  Median :4.505   Median : -0.3600   Median : 42.39   Median : 72.81
##  Mean   :4.514   Mean   : -0.4717   Mean   : 44.21   Mean   : 71.31
##  3rd Qu.:5.333   3rd Qu.: 13.7293   3rd Qu.: 53.04   3rd Qu.: 82.56
##  Max.   :8.567   Max.   : 43.6837   Max.   : 85.00   Max.   :108.24
##     ball_v_0         ball_ax_0          ball_az_0         ball_ay_0
##  Min.   : 60.30   Min.   :-25.0024   Min.   :-52.979   Min.   :-69.61
##  1st Qu.: 82.34   1st Qu.: -0.6260   1st Qu.:-25.096   1st Qu.:-33.44
##  Median : 88.81   Median :  1.0040   Median :-22.599   Median :-29.28
##  Mean   : 88.34   Mean   :  0.8644   Mean   :-23.112   Mean   :-29.74
##  3rd Qu.: 94.78   3rd Qu.:  1.9801   3rd Qu.:-20.158   3rd Qu.:-26.08
##  Max.   :112.25   Max.   : 26.1772   Max.   : -7.141   Max.   : 20.65
##   ball_apex_x         ball_apex_y      ball_apex_z      player_px_7_0
##  Min.   :-122.6479   Min.   : 34.66   Min.   :  5.536   Min.   :-172.36
##  1st Qu.: -42.9417   1st Qu.:142.25   1st Qu.: 47.641   1st Qu.:-140.22
##  Median :   1.8791   Median :163.24   Median : 70.295   Median :-132.60
##  Mean   :   0.5993   Mean   :163.89   Mean   : 76.216   Mean   :-131.89
##  3rd Qu.:  44.7935   3rd Qu.:186.15   3rd Qu.: 99.008   3rd Qu.:-124.60
##  Max.   : 126.4241   Max.   :259.81   Max.   :179.823   Max.   :  19.48
##  player_py_7_0    player_px_8_0      player_py_8_0    player_px_9_0
##  Min.   : 69.51   Min.   :-133.658   Min.   : 86.59   Min.   : 11.0
##  1st Qu.:251.98   1st Qu.: -14.979   1st Qu.:307.83   1st Qu.:125.9
##  Median :261.58   Median :   4.610   Median :315.86   Median :133.4
##  Mean   :260.79   Mean   :   1.944   Mean   :315.02   Mean   :132.9
##  3rd Qu.:270.20   3rd Qu.:  18.590   3rd Qu.:322.99   3rd Qu.:140.4
##  Max.   :308.24   Max.   : 127.267   Max.   :354.32   Max.   :171.6
##  player_py_9_0    landing_location_x landing_location_y  launch_angle
##  Min.   : 97.87   Min.   :-231.258   Min.   : 92.65     Min.   : 7.415
##  1st Qu.:250.10   1st Qu.: -87.135   1st Qu.:245.71     1st Qu.:22.713
##  Median :258.64   Median :   6.267   Median :282.61     Median :30.310
##  Mean   :258.54   Mean   :   3.583   Mean   :283.23     Mean   :31.689
##  3rd Qu.:267.57   3rd Qu.:  96.359   3rd Qu.:322.20     3rd Qu.:39.158
##  Max.   :309.15   Max.   : 265.284   Max.   :455.92     Max.   :63.992
```

```
##  launch_direction    launch_speed    runner_season_top_velo   runner_id
##  Min.   :-39.4338   Min.   : 63.59   Min.   : 5.135         Min.   :  1.0
##  1st Qu.:-11.4079   1st Qu.: 83.82   1st Qu.: 8.628         1st Qu.:257.0
##  Median : -0.4231   Median : 90.22   Median : 8.966         Median :464.0
##  Mean   : -0.3021   Mean   : 89.76   Mean   : 8.910         Mean   :471.9
##  3rd Qu.: 11.5128   3rd Qu.: 96.12   3rd Qu.: 9.255         3rd Qu.:688.0
##  Max.   : 40.0178   Max.   :114.20   Max.   :10.933         Max.   :970.0
##    fielder_id    fielding_team_id hitting_team_id    play_id
##  Min.   :  2.0   Min.   : 1.00    Min.   : 1.00   Min.   :   1
##  1st Qu.:161.0   1st Qu.: 8.00    1st Qu.: 8.00   1st Qu.:2251
##  Median :274.0   Median :15.00    Median :15.00   Median :4437
##  Mean   :280.5   Mean   :15.39    Mean   :15.27   Mean   :4335
##  3rd Qu.:405.0   3rd Qu.:23.00    3rd Qu.:23.00   3rd Qu.:6498
##  Max.   :564.0   Max.   :30.00    Max.   :30.00   Max.   :8231
```

```
head(tags)
```

```
##    fielder_pos inning batting_run_differential outs_on_play runs_on_play  throw
## 1            8      3                       -1            0            0 87.880
## 2            9      3                        1            0            1 72.619
## 3            8      3                       -2            0            0 84.538
## 5            7      1                        3            0            1 69.015
## 6            7      5                        1            0            1 88.316
## 10           8      6                        1            0            1 83.594
##    exchange throw_dist time_to_catch ball_vx_0 ball_vz_0 ball_vy_0 ball_v_0
## 1     1.879    179.059         2.800   6.78772  25.26944  89.94367 93.67220
## 2     0.901    171.262         5.739  12.52199  55.22027  76.91811 95.51164
## 3     1.000    170.335         5.806  -0.72926  63.24802  60.09668 87.24939
## 5     1.399    149.091         3.800 -30.14105  37.28200  76.39030 90.18818
## 6     1.159    196.690         4.600 -25.07339  43.00801  69.89212 85.80951
## 10    1.168    169.814         4.371  19.64975  39.39109  67.15522 80.29691
##    ball_ax_0 ball_az_0 ball_ay_0 ball_apex_x ball_apex_y ball_apex_z
## 1    0.97379 -22.38001 -23.37335    20.30198    154.5504    29.07879
## 2    1.20653 -20.20171 -37.69940    48.64890    203.3802   111.52706
## 3   -0.67352 -27.31559 -35.17692   -11.10200    155.9389   124.95093
## 5    1.05194 -23.00382 -28.87653   -74.15648    170.3974    54.35086
## 6    1.96600 -22.62036 -29.83686   -73.39374    175.6646    72.70193
## 10  -1.99252 -23.26311 -25.78170    41.62905    168.4944    64.91696
##    player_px_7_0 player_py_7_0 player_px_8_0 player_py_8_0 player_px_9_0
## 1        -124.915       268.383        20.725       306.246       156.964
## 2        -127.998       275.391        18.102       311.225       148.451
## 3        -142.715       251.029         5.463       327.319       128.432
## 5        -138.812       266.322        15.037       309.355       123.190
## 6        -121.891       262.587        17.960       297.468       122.476
## 10       -114.661       262.570        14.531       320.206       155.274
##    player_py_9_0 landing_location_x landing_location_y launch_angle
## 1        247.989           58.67898           280.9302     15.77588
## 2        261.408           96.60028           345.0141     35.18650
## 3        280.118          -31.66954           262.7014     47.60176
## 5        246.867         -131.77686           299.2973     24.12366
## 6        235.240         -131.90174           303.0716     29.54657
## 10       273.469           69.12439           298.6540     29.57287
##    launch_direction launch_speed runner_season_top_velo runner_id fielder_id
## 1           4.19214     94.73694                 6.125         1        212
```

```
## 2           9.06622     97.35416                 7.610         1        226
## 3          -0.26278     87.53115                 7.610         1         68
## 5         -21.17599     91.44045                 8.790         2        116
## 6         -19.58103     85.95590                 8.790         2         97
## 10         16.44019     81.55774                 7.913         3         58
##    fielding_team_id hitting_team_id play_id
## 1                26              18       1
## 2                22              18       2
## 3                 1              18       3
## 5                29              17       5
## 6                 9              17       6
## 10               19               4      10
```

```
sapply(tags, class)
```

```
##              fielder_pos                   inning batting_run_differential
##                "integer"                "integer"                 "integer"
##             outs_on_play             runs_on_play                     throw
##                "integer"                "integer"                 "numeric"
##                 exchange              throw_dist            time_to_catch
##                "numeric"                "numeric"                 "numeric"
##                ball_vx_0                ball_vz_0                 ball_vy_0
##                "numeric"                "numeric"                 "numeric"
##                 ball_v_0                ball_ax_0                 ball_az_0
##                "numeric"                "numeric"                 "numeric"
##                ball_ay_0              ball_apex_x               ball_apex_y
##                "numeric"                "numeric"                 "numeric"
##              ball_apex_z            player_px_7_0             player_py_7_0
##                "numeric"                "numeric"                 "numeric"
##            player_px_8_0            player_py_8_0             player_px_9_0
##                "numeric"                "numeric"                 "numeric"
##            player_py_9_0       landing_location_x        landing_location_y
##                "numeric"                "numeric"                 "numeric"
##             launch_angle         launch_direction              launch_speed
##                "numeric"                "numeric"                 "numeric"
##   runner_season_top_velo                runner_id                fielder_id
##                "numeric"                "integer"                 "integer"
##         fielding_team_id          hitting_team_id                   play_id
##                "integer"                "integer"                 "integer"
```

```
# let's convert outs_on_play
summary(tags$outs_on_play)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -1.00000  0.00000  0.00000  0.01308  0.00000  1.00000
```

```
tags[which(tags$outs_on_play == -1),]
```

```
##      fielder_pos inning batting_run_differential outs_on_play runs_on_play
## 1310           8      8                       -1           -1            0
## 3389           8      7                        3           -1            0
## 4291           9      6                       -1           -1            0
```

```
## 4858          9      5                      13          -1           0
## 7339          7      1                       1          -1           0
## 7978          8      9                       0          -1           0
## 8170          9      8                      -4          -1           0
##       throw exchange throw_dist time_to_catch ball_vx_0 ball_vz_0 ball_vy_0
## 1310 85.482    1.000    157.771         3.203   1.04526  32.88259  74.23479
## 3389 73.481    2.335    139.107         3.771   9.69925  35.28123  59.11075
## 4291 71.200    0.400    128.000         8.567  10.69019  81.05537  56.62723
## 4858 84.400    2.430    124.000         5.734  23.30078  59.96197  69.07785
## 7339 87.200    1.000    189.000         3.700 -14.31919  36.72607  70.02201
## 7978 60.800    2.470    181.000         6.566   0.06412  57.29804  76.44523
## 8170 79.900    1.830    143.000         4.467  27.16637  43.70268  76.95399
##      ball_v_0 ball_ax_0 ball_az_0 ball_ay_0 ball_apex_x ball_apex_y ball_apex_z
## 1310 81.19829   0.56772 -20.16350 -21.83802    15.83598    146.0717    39.73805
## 3389 69.51921   1.39556 -20.85708 -18.84565    15.39841    132.7395    49.91468
## 4291 99.45298  10.54035 -37.68417 -37.88350    68.96240    138.2326   167.47434
## 4858 94.39339   4.96088 -24.34199 -36.72203    99.33398    172.0155   119.49751
## 7339 80.35499  -8.82134 -22.90413 -25.04022   -48.98998    145.0114    50.74777
## 7978 95.53504  -7.69362 -19.97025 -43.37464   -18.38861    192.4814   120.71104
## 8170 92.57350   7.73211 -27.73164 -38.70504    86.06898    153.5965    60.78150
##      player_px_7_0 player_py_7_0 player_px_8_0 player_py_8_0 player_px_9_0
## 1310    -111.68500      268.5310      26.40200      299.2210      139.7810
## 3389    -114.74400      237.2860       7.83700      309.6790      132.7440
## 4291    -112.30995      265.3520      19.05370      328.0933      154.4674
## 4858    -148.93113      264.5295      -9.93432      315.0347      135.8707
## 7339    -142.63027      263.8713     -24.45998      319.6823      132.9910
## 7978     -79.86917      233.8261      18.33127      259.7142      134.2336
## 8170    -119.60792      279.2858      37.50576      324.0410      132.9091
##      player_py_9_0 landing_location_x landing_location_y launch_angle
## 1310      241.8220           47.98860           259.9903     24.22356
## 3389      258.8190           17.90320           242.7035     30.82108
## 4291      258.7099          131.72735           238.4239     54.89488
## 4858      268.4054          192.56861           286.3514     41.08531
## 7339      267.8312         -112.71981           249.7147     27.45183
## 7978      235.1538          -54.09836           324.9084     38.37137
## 8170      254.7073          177.82689           244.0603     28.84380
##      launch_direction launch_speed runner_season_top_velo runner_id fielder_id
## 1310          0.43176     82.37879                  8.480       155        460
## 3389          9.48584     70.58950                  8.961       368        323
## 4291         10.70275    101.23413                  8.872       448        507
## 4858         20.19035     98.23509                  8.202       506        489
## 7339        -11.82824     81.64831                  8.966       812        109
## 7978         -0.47200     94.87539                  8.978       911        149
## 8170         21.08853     92.67605                  8.867       946        512
##      fielding_team_id hitting_team_id play_id
## 1310               17              14    1310
## 3389                4              19    3389
## 4291               11               4    4291
## 4858               26              16    4858
## 7339               11              28    7339
## 7978               15              29    7978
## 8170               27               3    8170
```

```r
# since all runs_on_play in these columns are 0 I will assume
# this is a data entry error and convert these from -1 to 1
tags[which(tags$outs_on_play == -1),"outs_on_play"] <- 1
summary(tags$outs_on_play)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.00000 0.00000 0.01613 0.00000 1.00000
```

```r
tags$outs_on_play <- as.factor(ifelse(tags$outs_on_play == 1, "out", "no_out"))
table(tags$outs_on_play)
```

```
##
## no_out    out
##   4513     74
```

```r
# let's convert runs_on_play
summary(tags$runs_on_play)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  1.0000  1.0000  0.7713  1.0000  1.0000
```

```r
tags$runs_on_play <- as.factor(ifelse(tags$runs_on_play == 1, "run", "no_run"))
table(tags$runs_on_play)
```

```
##
## no_run    run
##   1049   3538
```

```r
summary(tags)
```

```
##   fielder_pos        inning      batting_run_differential outs_on_play
##  Min.   :7.000   Min.   :0.000   Min.   :-19.0000         no_out:4513
##  1st Qu.:7.000   1st Qu.:3.000   1st Qu.: -1.0000         out   :  74
##  Median :8.000   Median :5.000   Median :  0.0000
##  Mean   :8.028   Mean   :4.682   Mean   :  0.5075
##  3rd Qu.:9.000   3rd Qu.:7.000   3rd Qu.:  2.0000
##  Max.   :9.000   Max.   :9.000   Max.   : 19.0000
##  runs_on_play      throw           exchange       throw_dist
##  no_run:1049   Min.   : 30.30   Min.   :0.070   Min.   :  7.516
##  run   :3538   1st Qu.: 73.42   1st Qu.:1.034   1st Qu.:154.827
##                Median : 82.20   Median :1.200   Median :178.119
##                Mean   : 80.36   Mean   :1.352   Mean   :177.843
##                3rd Qu.: 88.70   3rd Qu.:1.500   3rd Qu.:203.688
##                Max.   :104.70   Max.   :3.840   Max.   :320.081
##  time_to_catch     ball_vx_0          ball_vz_0        ball_vy_0
##  Min.   :2.069   Min.   :-53.3140   Min.   :-13.84   Min.   :-87.01
##  1st Qu.:3.671   1st Qu.:-13.9562   1st Qu.: 33.89   1st Qu.: 61.84
##  Median :4.505   Median : -0.3600   Median : 42.39   Median : 72.81
##  Mean   :4.514   Mean   : -0.4717   Mean   : 44.21   Mean   : 71.31
##  3rd Qu.:5.333   3rd Qu.: 13.7293   3rd Qu.: 53.04   3rd Qu.: 82.56
```

```
##  Max.   :8.567   Max.   : 43.6837   Max.   : 85.00   Max.   :108.24
##     ball_v_0        ball_ax_0          ball_az_0         ball_ay_0
##  Min.   : 60.30   Min.   :-25.0024   Min.   :-52.979   Min.   :-69.61
##  1st Qu.: 82.34   1st Qu.: -0.6260   1st Qu.:-25.096   1st Qu.:-33.44
##  Median : 88.81   Median :  1.0040   Median :-22.599   Median :-29.28
##  Mean   : 88.34   Mean   :  0.8644   Mean   :-23.112   Mean   :-29.74
##  3rd Qu.: 94.78   3rd Qu.:  1.9801   3rd Qu.:-20.158   3rd Qu.:-26.08
##  Max.   :112.25   Max.   : 26.1772   Max.   : -7.141   Max.   : 20.65
##   ball_apex_x         ball_apex_y      ball_apex_z      player_px_7_0
##  Min.   :-122.6479   Min.   : 34.66   Min.   :  5.536   Min.   :-172.36
##  1st Qu.: -42.9417   1st Qu.:142.25   1st Qu.: 47.641   1st Qu.:-140.22
##  Median :   1.8791   Median :163.24   Median : 70.295   Median :-132.60
##  Mean   :   0.5993   Mean   :163.89   Mean   : 76.216   Mean   :-131.89
##  3rd Qu.:  44.7935   3rd Qu.:186.15   3rd Qu.: 99.008   3rd Qu.:-124.60
##  Max.   : 126.4241   Max.   :259.81   Max.   :179.823   Max.   :  19.48
##  player_py_7_0    player_px_8_0      player_py_8_0    player_px_9_0
##  Min.   : 69.51   Min.   :-133.658   Min.   : 86.59   Min.   : 11.0
##  1st Qu.:251.98   1st Qu.: -14.979   1st Qu.:307.83   1st Qu.:125.9
##  Median :261.58   Median :   4.610   Median :315.86   Median :133.4
##  Mean   :260.79   Mean   :   1.944   Mean   :315.02   Mean   :132.9
##  3rd Qu.:270.20   3rd Qu.:  18.590   3rd Qu.:322.99   3rd Qu.:140.4
##  Max.   :308.24   Max.   : 127.267   Max.   :354.32   Max.   :171.6
##  player_py_9_0    landing_location_x landing_location_y  launch_angle
##  Min.   : 97.87   Min.   :-231.258   Min.   : 92.65     Min.   : 7.415
##  1st Qu.:250.10   1st Qu.: -87.135   1st Qu.:245.71     1st Qu.:22.713
##  Median :258.64   Median :   6.267   Median :282.61     Median :30.310
##  Mean   :258.54   Mean   :   3.583   Mean   :283.23     Mean   :31.689
##  3rd Qu.:267.57   3rd Qu.:  96.359   3rd Qu.:322.20     3rd Qu.:39.158
##  Max.   :309.15   Max.   : 265.284   Max.   :455.92     Max.   :63.992
##  launch_direction    launch_speed    runner_season_top_velo   runner_id
##  Min.   :-39.4338   Min.   : 63.59   Min.   : 5.135         Min.   :  1.0
##  1st Qu.:-11.4079   1st Qu.: 83.82   1st Qu.: 8.628         1st Qu.:257.0
##  Median : -0.4231   Median : 90.22   Median : 8.966         Median :464.0
##  Mean   : -0.3021   Mean   : 89.76   Mean   : 8.910         Mean   :471.9
##  3rd Qu.: 11.5128   3rd Qu.: 96.12   3rd Qu.: 9.255         3rd Qu.:688.0
##  Max.   : 40.0178   Max.   :114.20   Max.   :10.933         Max.   :970.0
##    fielder_id    fielding_team_id hitting_team_id    play_id
##  Min.   :  2.0   Min.   : 1.00    Min.   : 1.00   Min.   :   1
##  1st Qu.:161.0   1st Qu.: 8.00    1st Qu.: 8.00   1st Qu.:2251
##  Median :274.0   Median :15.00    Median :15.00   Median :4437
##  Mean   :280.5   Mean   :15.39    Mean   :15.27   Mean   :4335
##  3rd Qu.:405.0   3rd Qu.:23.00    3rd Qu.:23.00   3rd Qu.:6498
##  Max.   :564.0   Max.   :30.00    Max.   :30.00   Max.   :8231
```

```r
# outs_on_play and runs_on_play should be mutually exclusive based on Dec 22, 2021 email from Rob
yes_yes <- tags[which(tags$outs_on_play == "out" & tags$runs_on_play == "run"),] # checks out
no_no <- tags[which(tags$outs_on_play == "no_out" & tags$runs_on_play == "no_run"),]
# 975 rows where neither an out nor run occurs
# totally fine because runner does not have to tag on the fly ball

# add in fielding/fielder information
for (i in 1:nrow(tags)) {
  cols <- grep(tags$fielder_pos[i],colnames(tags))
  fielder_x <- tags[i,26] - tags[i,cols[1]]
```

```
  fielder_y <- tags[i,27] - tags[i,cols[2]]
  tags$fielder_distance_x[i] <- fielder_x
  tags$fielder_distance_y[i] <- fielder_y
}
# grep(tags$fielder_pos[1],colnames(tags))

# checking for near zero and zero variance columns
infodensity <- nearZeroVar(tags, saveMetrics= TRUE)
infodensity[infodensity$nzv,] # there is one column to keep an eye on (outs_on_play) for near zero
```

```
##              freqRatio percentUnique zeroVar  nzv
## outs_on_play  60.98649    0.04360148   FALSE TRUE
```

```
# variance but can remove in model build because it'll affect runs_on_play predictive ability
# checking for highly correlated values
highlycorrelated <- findCorrelation(cor_matrix(tags), cutoff = 0.95)
colnames(tags)[highlycorrelated]
```

```
## [1] "player_py_9_0"      "ball_apex_x"        "ball_vz_0"
## [4] "player_px_9_0"      "ball_az_0"          "landing_location_y"
## [7] "fielding_team_id"
```

```
# a handful of columns appear to have high correlations with other variables but I'm not sure
# which ones and I don't feel comfortable removing them currently
```

Step 2: Create Model Training/Holdout Sets

```
write.csv(tags, "Go_NoGo/tag_ups.csv", row.names = F)
data <- tags[-c(4,32:36)] # remove outs_on_play and unique identifiers
set.seed(23); train.rows <- sample(1:nrow(tags), 0.7*nrow(tags)) # 70% train; 30% holdout
TRAIN <- data[train.rows,]
HOLDOUT <- data[-train.rows,]
```

Step 3: Model Building Time

```
# y = runs_on_play
# Set up how generalization error is to be estimated (5-fold crossvalidation shown here)
fitControl <- trainControl(method="cv",number=5, classProbs=TRUE,
                           summaryFunction=twoClassSummary, allowParallel = TRUE)
#The following examples will use AUC as the metric

# vanilla logistic model
set.seed(23); GLM <- train(runs_on_play~.,data=TRAIN,method='glm',
                           trControl=fitControl, preProc = c("center", "scale"))
```

```
## Warning in train.default(x, y, weights = w, ...): The metric "Accuracy" was not
## in the result set. ROC will be used instead.
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
GLM$results # ROC: 0.9387461 ; SD: 0.009542589
```

```
##   parameter       ROC      Sens      Spec       ROCSD     SensSD     SpecSD
## 1      none 0.9387461 0.7029913 0.9414361 0.009542589 0.03493874 0.01505763
```

```
postResample(predict(GLM,newdata=HOLDOUT),HOLDOUT$runs_on_play)  # Accuracy: 0.8925200
```

```
##  Accuracy     Kappa
## 0.8925200 0.6877293
```

```
roc(HOLDOUT$runs_on_play,predict(GLM,newdata=HOLDOUT,type="prob")[,2]) # AUC: 0.9465
```

```
## Setting levels: control = no_run, case = run
```

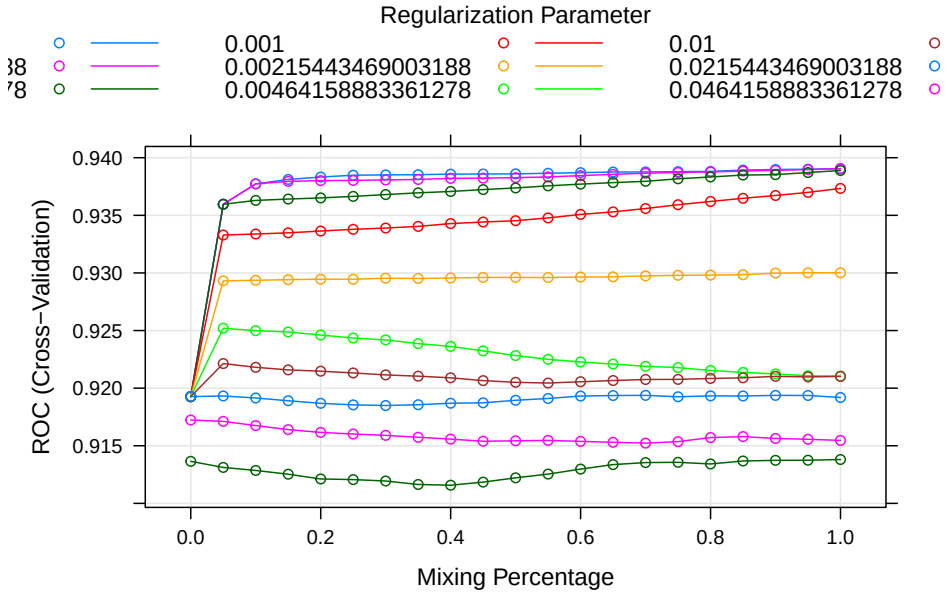
```
## Setting direction: controls < cases
```

```
##
## Call:
## roc.default(response = HOLDOUT$runs_on_play, predictor = predict(GLM,     newdata = HOLDOUT, type = "
##
## Data: predict(GLM, newdata = HOLDOUT, type = "prob")[, 2] in 315 controls (HOLDOUT$runs_on_play no_ru
## Area under the curve: 0.9465
```

```
# regularized logistic regression
glmnetGrid <- expand.grid(alpha = seq(0,1,.05),lambda = 10^seq(-4,-1,length=10))

set.seed(23); GLMnet <- train(runs_on_play~.,data=TRAIN,method='glmnet',
                              trControl=fitControl, tuneGrid=glmnetGrid,
                               preProc = c("center", "scale"))
```

```
## Warning in train.default(x, y, weights = w, ...): The metric "Accuracy" was not
## in the result set. ROC will be used instead.
```

```
# GLMnet  # Look at details of all fits
# Commenting out for clarity of pdf
plot(GLMnet) # See how error changes with choices
```




```
GLMnet$bestTune # best parameters
```

```
##     alpha       lambda
## 202     1 0.0002154435
```

```
GLMnet$results[rownames(GLMnet$bestTune),]  # ROC: 0.9390544 ; SD: 0.009445003
```

```
##     alpha       lambda       ROC      Sens      Spec       ROCSD     SensSD
## 202     1 0.0002154435 0.9390544 0.7029913 0.9438604 0.009445003 0.03623906
##         SpecSD
## 202 0.01612652
```

```
varImp(GLMnet)
```

```
## glmnet variable importance
##
##   only 20 most important variables shown (out of 31)
##
##                         Overall
## ball_apex_z             100.000
## ball_vz_0                98.682
## ball_apex_y              71.458
## ball_vy_0                62.828
## launch_angle             57.438
## time_to_catch            33.941
## landing_location_y       31.737
## launch_speed             28.291
## ball_az_0                11.050
## fielder_distance_y       10.991
## runner_season_top_velo    7.970
```

```
## exchange                5.798
## fielder_pos             5.235
## fielder_distance_x      3.068
## launch_direction        3.068
## throw                   2.871
## throw_dist              1.677
## inning                  1.431
## player_px_8_0           1.417
## ball_ax_0               1.327
```

```
postResample(predict(GLMnet,newdata=HOLDOUT),HOLDOUT$runs_on_play)  # Accuracy: 0.8910675
```

```
##  Accuracy     Kappa
## 0.8910675 0.6835094
```

```
roc(HOLDOUT$runs_on_play,predict(GLMnet,newdata=HOLDOUT,type="prob")[,2]) # AUC: 0.9463
```

```
## Setting levels: control = no_run, case = run
## Setting direction: controls < cases
```

```
##
## Call:
## roc.default(response = HOLDOUT$runs_on_play, predictor = predict(GLMnet,     newdata = HOLDOUT, type
##
## Data: predict(GLMnet, newdata = HOLDOUT, type = "prob")[, 2] in 315 controls (HOLDOUT$runs_on_play no
## Area under the curve: 0.9463
```

```
#### minimal improvement and considerably longer run time from regularized logistic regression
# will drop it from consideration

# vanilla partition (decision tree)
treeGrid <- expand.grid(cp=10^seq(-5,-1,length=25))

set.seed(23); TREE <- train(runs_on_play~.,data=TRAIN,method='rpart', tuneGrid=treeGrid,
                            trControl=fitControl, preProc = c("center", "scale"))
```

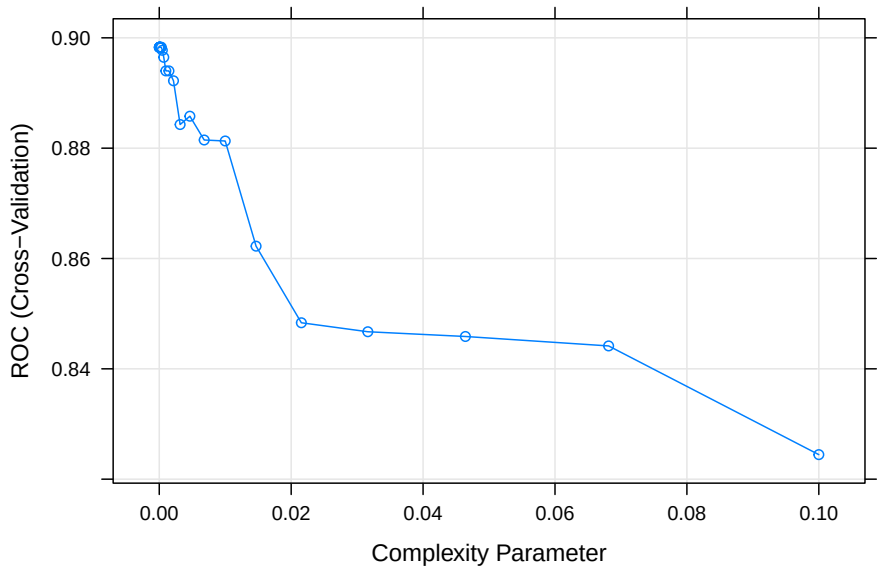
```
## Warning in train.default(x, y, weights = w, ...): The metric "Accuracy" was not
## in the result set. ROC will be used instead.
```

```
TREE  #Look at details of all fits
```

```
## CART
##
## 3210 samples
##   31 predictor
##    2 classes: 'no_run', 'run'
##
## Pre-processing: centered (31), scaled (31)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 2568, 2568, 2568, 2568, 2568
## Resampling results across tuning parameters:
```

```
##
##    cp             ROC        Sens       Spec
##    1.000000e-05  0.8982874  0.6798341  0.9196212
##    1.467799e-05  0.8982874  0.6798341  0.9196212
##    2.154435e-05  0.8982874  0.6798341  0.9196212
##    3.162278e-05  0.8982874  0.6798341  0.9196212
##    4.641589e-05  0.8982874  0.6798341  0.9196212
##    6.812921e-05  0.8982874  0.6798341  0.9196212
##    1.000000e-04  0.8982874  0.6798341  0.9196212
##    1.467799e-04  0.8982874  0.6798341  0.9196212
##    2.154435e-04  0.8982874  0.6798341  0.9196212
##    3.162278e-04  0.8982874  0.6798341  0.9196212
##    4.641589e-04  0.8977404  0.6771130  0.9200253
##    6.812921e-04  0.8964705  0.6784736  0.9192172
##    1.000000e-03  0.8940035  0.6811947  0.9216390
##    1.467799e-03  0.8940035  0.6811947  0.9216390
##    2.154435e-03  0.8922148  0.6961700  0.9244673
##    3.162278e-03  0.8842852  0.7016308  0.9264850
##    4.641589e-03  0.8857910  0.7070543  0.9281004
##    6.812921e-03  0.8814779  0.7166527  0.9297198
##    1.000000e-02  0.8813006  0.7288976  0.9272955
##    1.467799e-02  0.8622338  0.7042401  0.9232543
##    2.154435e-02  0.8483510  0.6988352  0.9163930
##    3.162278e-02  0.8467225  0.6933930  0.9208374
##    4.641589e-02  0.8458636  0.6974746  0.9180091
##    6.812921e-02  0.8441512  0.6131209  0.9378071
##    1.000000e-01  0.8244490  0.7466313  0.8671318
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.0003162278.
```

```
plot(TREE) #See how error changes with choices
```



```
TREE$bestTune # best parameters
```

```
##              cp
## 10 0.0003162278
```

```
TREE$results[rownames(TREE$bestTune),] # ROC: 0.8982874 ; SD: 0.0115885
```

```
##              cp       ROC      Sens      Spec     ROCSD     SensSD     SpecSD
## 10 0.0003162278 0.8982874 0.6798341 0.9196212 0.0115885 0.03909093 0.02276435
```

```
varImp(TREE)
```

```
## rpart variable importance
##
##   only 20 most important variables shown (out of 31)
##
##                         Overall
## ball_vy_0               100.000
## ball_apex_y              99.773
## fielder_distance_y       86.440
## landing_location_y       71.766
## launch_angle             69.269
## landing_location_x       33.236
## ball_apex_x              28.953
## launch_speed             14.126
## ball_v_0                 12.913
## ball_vx_0                11.827
## throw_dist               10.270
## throw                    10.033
## launch_direction          8.803
## ball_vz_0                 7.954
## ball_apex_z               7.303
## time_to_catch             6.062
## runner_season_top_velo    5.521
## ball_ay_0                 4.460
## fielder_distance_x        3.736
## player_py_8_0             2.903
```

```
postResample(predict(TREE,newdata=HOLDOUT),HOLDOUT$runs_on_play)  # Accuracy: 0.8772694
```

```
##  Accuracy     Kappa
## 0.8772694 0.6502266
```

```
roc(HOLDOUT$runs_on_play,predict(TREE,newdata=HOLDOUT,type="prob")[,2]) # AUC: 0.9223
```

```
## Setting levels: control = no_run, case = run
## Setting direction: controls < cases
```

```
##
## Call:
```

```
## roc.default(response = HOLDOUT$runs_on_play, predictor = predict(TREE,     newdata = HOLDOUT, type =
##
## Data: predict(TREE, newdata = HOLDOUT, type = "prob")[, 2] in 315 controls (HOLDOUT$runs_on_play no_
## Area under the curve: 0.9223
```

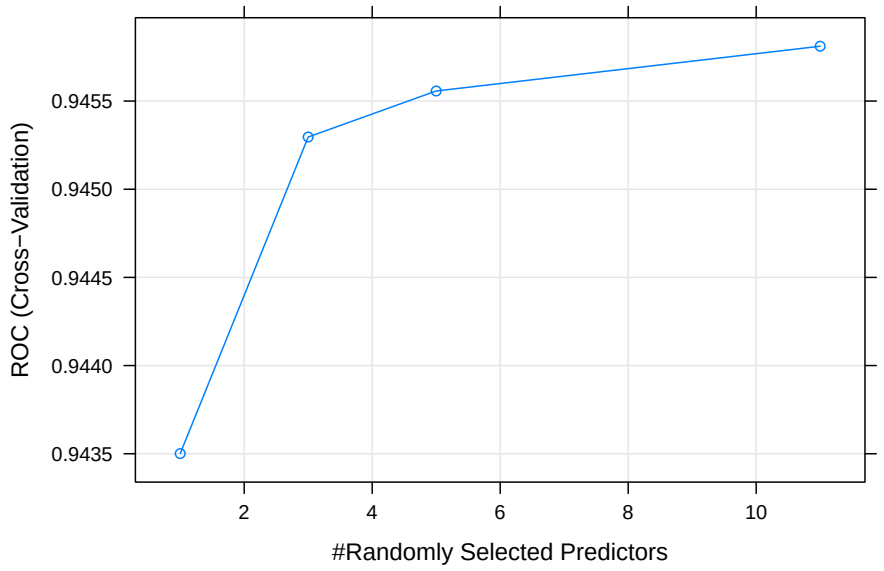
```
#### no improvement for the vanilla tree will drop from consideration

# Random Forest
forestGrid <- expand.grid(mtry=c(1,3,5,11))

# using parallelization
cluster <- makeCluster(detectCores() - 1) #Line 1 for parallelization
registerDoParallel(cluster) #Line 2 for parallelization
set.seed(23); FOREST <- train(runs_on_play~.,data=TRAIN,method='rf',tuneGrid=forestGrid,
                                        trControl=fitControl, preProc = c("center", "scale"))
```

```
## Warning in train.default(x, y, weights = w, ...): The metric "Accuracy" was not
## in the result set. ROC will be used instead.
```

```
stopCluster(cluster) #Line 3 for parallelization
registerDoSEQ() #Line 4 for parallelization

FOREST
```

```
## Random Forest
##
## 3210 samples
##   31 predictor
##    2 classes: 'no_run', 'run'
##
## Pre-processing: centered (31), scaled (31)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 2568, 2568, 2568, 2568, 2568
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##    1    0.9435009  0.6566862  0.9588009
##    3    0.9452960  0.7152269  0.9487048
##    5    0.9455572  0.7261579  0.9474919
##   11    0.9458110  0.7261672  0.9438563
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 11.
```

```
plot(FOREST)
```




```
FOREST$bestTune
```

```
##   mtry
## 4   11
```

```
FOREST$results[rownames(FOREST$bestTune),] # ROC: 0.945811 ; SD: 0.01194926
```

```
##   mtry      ROC      Sens      Spec      ROCSD    SensSD     SpecSD
## 4   11 0.945811 0.7261672 0.9438563 0.01194926 0.0399882 0.01468355
```

```
varImp(FOREST)
```

```
## rf variable importance
##
##   only 20 most important variables shown (out of 31)
##
##                         Overall
## ball_apex_y             100.000
## ball_vy_0                60.693
## fielder_distance_y       46.871
## landing_location_y       38.528
## landing_location_x       24.346
## launch_speed             21.280
## launch_angle             20.671
## ball_apex_x              19.311
## runner_season_top_velo   14.729
## ball_v_0                 14.714
## throw_dist               12.536
```

```
## throw                   11.680
## ball_vx_0               10.132
## ball_apex_z              8.995
## ball_vz_0                8.960
## launch_direction         8.355
## fielder_distance_x       7.954
## exchange                 7.549
## player_py_9_0            7.385
## time_to_catch            6.957
```

```
postResample(predict(FOREST,newdata=HOLDOUT),HOLDOUT$runs_on_play)  # Accuracy: 0.8954248
```

```
##  Accuracy     Kappa
## 0.8954248 0.7002857
```

```
roc(HOLDOUT$runs_on_play,predict(FOREST,newdata=HOLDOUT,type="prob")[,2]) # AUC: 0.9514
```

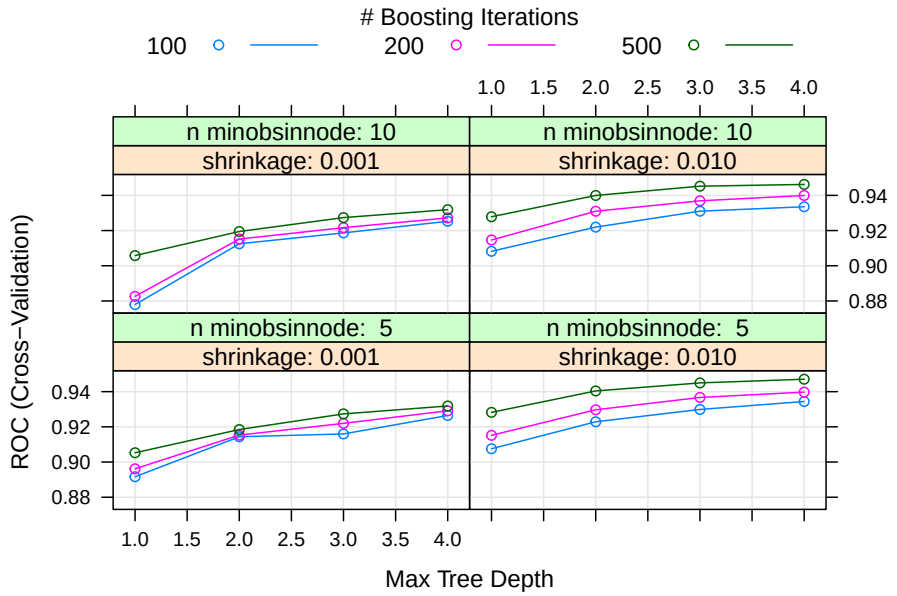
```
## Setting levels: control = no_run, case = run
## Setting direction: controls < cases
```

```
##
## Call:
## roc.default(response = HOLDOUT$runs_on_play, predictor = predict(FOREST,     newdata = HOLDOUT, type
##
## Data: predict(FOREST, newdata = HOLDOUT, type = "prob")[, 2] in 315 controls (HOLDOUT$runs_on_play no
## Area under the curve: 0.9514
```

```
#### random forest does well and has a similar run time as the vanilla regression with a small
# improvement (it is close but within 1 SD so I cannot say which model is truly better)

# Boosted Tree
gbmGrid <- expand.grid(n.trees=c(100,200,500),interaction.depth=1:4,
                       shrinkage=c(.01,.001),n.minobsinnode=c(5,10))

cluster <- makeCluster(detectCores() - 1) #Line 1 for parallelization
registerDoParallel(cluster) #Line 2 for parallelization
set.seed(23); GBM <- train(runs_on_play~.,data=TRAIN, method='gbm',tuneGrid=gbmGrid,verbose=FALSE,
                    trControl=fitControl, preProc = c("center", "scale"))
```

```
## Warning in train.default(x, y, weights = w, ...): The metric "Accuracy" was not
## in the result set. ROC will be used instead.
```

```
stopCluster(cluster) #Line 3 for parallelization
registerDoSEQ()

# GBM # for clairty of pdf
plot(GBM)
```




```
GBM$bestTune
```

```
##    n.trees interaction.depth shrinkage n.minobsinnode
## 45     500                 4      0.01              5
```

```
GBM$results[rownames(GBM$bestTune),] # ROC: 0.9470632 ; SD: 0.01116904
```

```
##    shrinkage interaction.depth n.minobsinnode n.trees       ROC      Sens
## 45      0.01                 4              5     500 0.9470632 0.7274998
##         Spec      ROCSD     SensSD     SpecSD
## 45 0.9410297 0.01116904 0.02337229 0.02150602
```

```
varImp(GBM)
```

```
## gbm variable importance
##
##   only 20 most important variables shown (out of 31)
##
##                          Overall
## ball_apex_y              100.000
## ball_vy_0                 36.008
## fielder_distance_y        29.663
## landing_location_x        23.056
## ball_apex_x               11.703
## launch_speed               9.951
## runner_season_top_velo     8.202
## launch_angle               7.272
## throw                      5.959
## throw_dist                 3.382
## ball_vx_0                  3.123
## ball_vz_0                  2.907
```

```
## ball_apex_z                2.845
## exchange                   2.536
## landing_location_y         2.162
## batting_run_differential   2.059
## time_to_catch              1.460
## launch_direction           1.446
## ball_v_0                   1.137
## ball_ay_0                  1.076
```

```
postResample(predict(GBM,newdata=HOLDOUT,n.trees=500),HOLDOUT$runs_on_play)  # Accuracy: 0.8983297
```

```
##  Accuracy     Kappa
## 0.8983297 0.7086111
```

```
roc(HOLDOUT$runs_on_play,predict(GBM,newdata=HOLDOUT,type="prob",n.trees=500)[,2]) # AUC: 0.951
```

```
## Setting levels: control = no_run, case = run
## Setting direction: controls < cases
```

```
##
## Call:
## roc.default(response = HOLDOUT$runs_on_play, predictor = predict(GBM,     newdata = HOLDOUT, type =
##
## Data: predict(GBM, newdata = HOLDOUT, type = "prob", n.trees = 500)[, 2] in 315 controls (HOLDOUT$ru
## Area under the curve: 0.951
```

```
#### Boosted Tree does well and has considerably longer run time but still I have not had
# a model separate from any of the others in terms of performance
```

Step 4: Choosing the best model of those considered

Model Over Fitting Evaluation: We see Vanilla Logistic Regression, Random Forest, and Boosted Tree all perform very similar and all of the training ROC values are withing 1 standard deviation of the best performing model [0.9358942, 0.9582322]. I want to test if the models overfit the data, where a larger value could point to more over fitting and no gap means under fitting.

```
# checking between models to see if we have any significant overfitting between training and validating
# (holdout - training)

# Vanilla Regression
0.9465 - 0.9387461 # 0.0077539
```

```
## [1] 0.0077539
```

```
# Random Forest
0.9514 - 0.945811 # 0.005589
```

```
## [1] 0.005589
```

```
# Boosted Tree
0.951 - 0.9470632 # 0.0039368
```

```
## [1] 0.0039368
```

None of these models appear to overfit the data. I will use the boosted tree model, since it had the best performance during training and does not appear to overfit the data.

Final Model

```
FINAL <- train(runs_on_play~., data, method = "gbm",
               tuneGrid = expand.grid(n.trees=500, interaction.depth=4,
                                      shrinkage=0.01, n.minobsinnode=5),
               verbose = FALSE,
               trControl = trainControl(method = "none"),
               preProc = c("center", "scale"))
predictions <- predict(FINAL, HOLDOUT)
confusionMatrix(predictions, HOLDOUT$runs_on_play)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction no_run  run
##     no_run    246   45
##     run        69 1017
##
##                Accuracy : 0.9172
##                  95% CI : (0.9014, 0.9312)
##     No Information Rate : 0.7712
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.7589
##
##  Mcnemar's Test P-Value : 0.03123
##
##             Sensitivity : 0.7810
##             Specificity : 0.9576
##          Pos Pred Value : 0.8454
##          Neg Pred Value : 0.9365
##              Prevalence : 0.2288
##          Detection Rate : 0.1786
##    Detection Prevalence : 0.2113
##       Balanced Accuracy : 0.8693
##
##        'Positive' Class : no_run
##
```

```
saveRDS(FINAL, "model.rds") # save model to working directory
saveRDS(FINAL, "Go_NoGo/model.rds") # save model to shiny app folder
```

The boosted tree model does extremely well in its predictions, having an accuracy of 91.72%.

```
probs <- predict(FINAL, type = "prob")
quantile(probs$run, 0.25) # potential run scoring prob threshold for red/yellow
```

```
##       25%
## 0.6206143
```

```
quantile(probs$run, 0.75) # potential run scoring prob threshold for yellow/green
```

```
##       75%
## 0.9784665
```

```
summary(probs$run); sd(probs$run) # information to make 20/80 scale for tools grade
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.04668 0.62061 0.95455 0.77143 0.97847 0.98066
```

```
## [1] 0.3012723
```

---

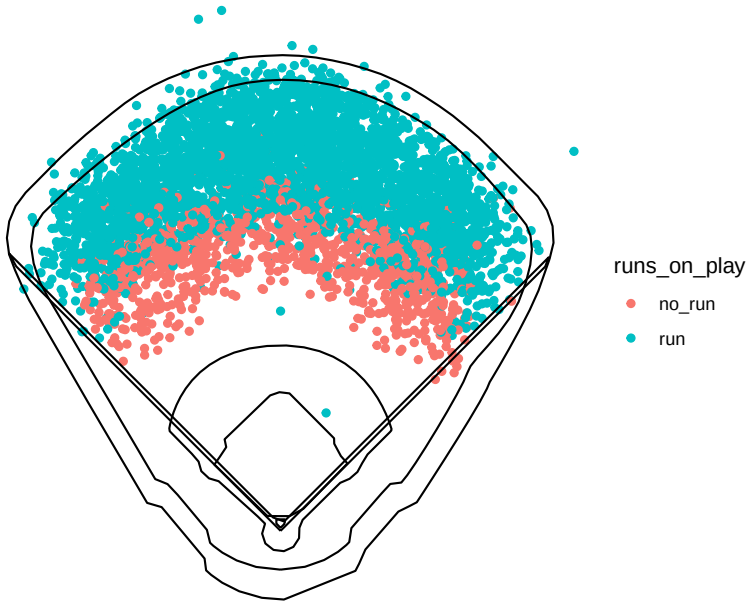
## Goal 2 Additional Code for UI

---

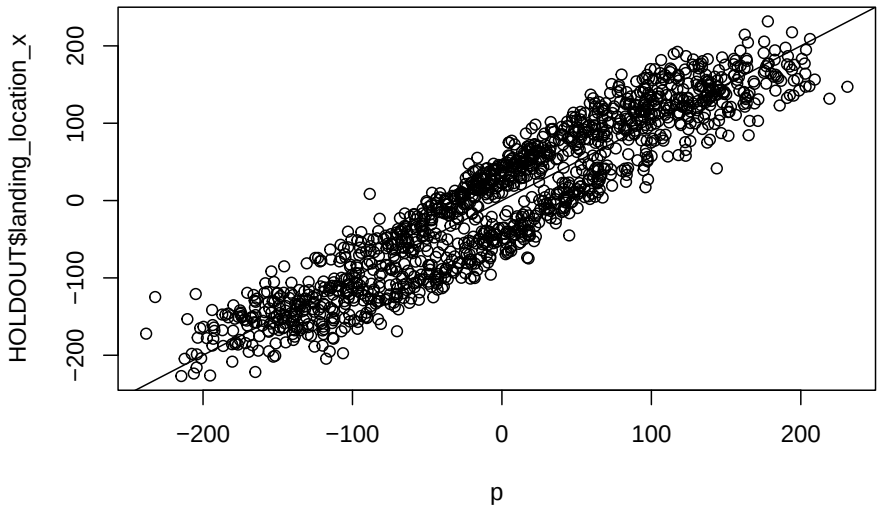
A user guide to the UI is provided: *Once the app has been launched, on the Red Rover tab the user will provide inputs for the provided dropdown menus and numeric/slider inputs. The defaults are already selected. After the desired inputs have been selected the user will hit the "update" button and will see a loading icon and the output of the model predicting a Go or No Go situation.* On the second tab (Stop Light), the user will be able to select the desired outfielder based on their ID. Once selected the the UI will update to show a plot of all the fly balls fielded by that outfielder with a color designation based on a run scoring or not overlaid on Kauffmann Stadium. The user also sees a text output reporting the stop light tier (explained in more detail below) and 20/80 scale grade (also in detail below) for the chosen outfielder.

The remaining code in this section is used in support of the UI.

```
batted_ball_result <- tags[,c("outs_on_play","runs_on_play","landing_location_x",
                              "landing_location_y","runner_id","fielder_id")]
fig <- batted_ball_result %>%
        ggplot(aes(x=landing_location_x, y=landing_location_y, color=runs_on_play)) +
        geom_spraychart(stadium_ids = "royals",
                        stadium_transform_coords = TRUE,
                        stadium_segments = "all") +
        theme_void() +
        coord_fixed()
fig
```



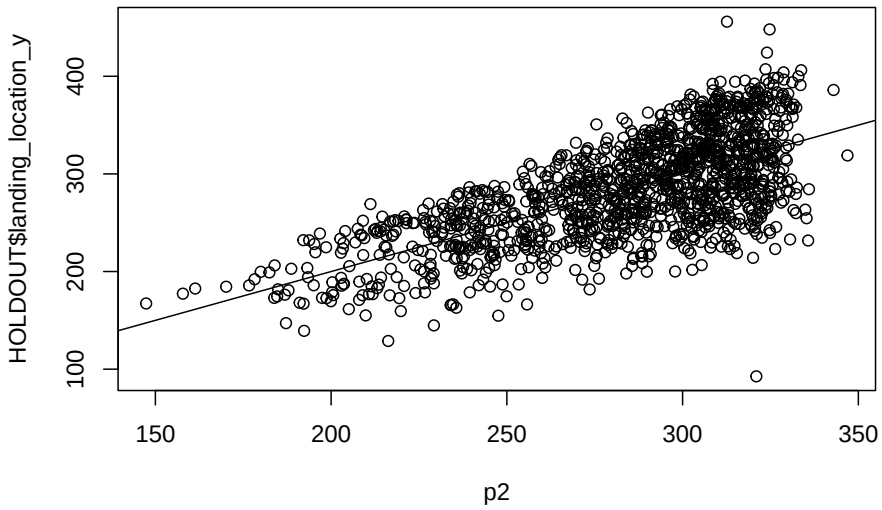

```
landing_model_x <- lm(landing_location_x ~ launch_direction + launch_speed + launch_angle, data=data)
landing_model_y <- lm(landing_location_y ~ (launch_direction + launch_speed + launch_angle)^2, data=data
# summary(landing_model_x)
# summary(landing_model_y)
p <- predict(landing_model_x, HOLDOUT)
p2 <- predict(landing_model_y, HOLDOUT)
plot(y=HOLDOUT$landing_location_x, x=p); abline(0,1) # looks good to control for user inputs
```




```
sqrt(mean((p-HOLDOUT$landing_location_x)^2)) # RMSE: 38.34
```

```
## [1] 38.34361
```

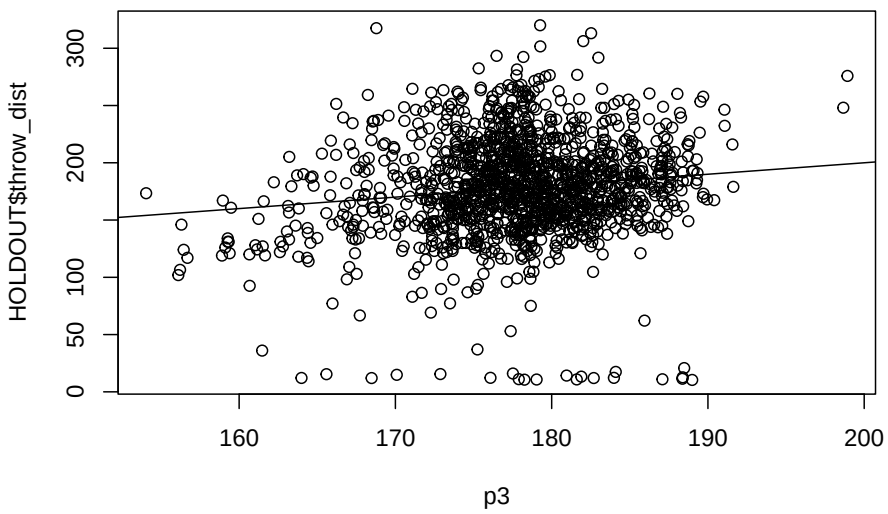
```
saveRDS(landing_model_x, "landing_x.rds")
saveRDS(landing_model_x, "Go_NoGo/landing_x.rds")

plot(y=HOLDOUT$landing_location_y, x=p2); abline(0,1) # does a decent job to control for user inputs
```




```
sqrt(mean((p2-HOLDOUT$landing_location_y)^2)) # RMSE: 39.37
```

```
## [1] 39.37383
```

```
saveRDS(landing_model_y, "landing_y.rds")
saveRDS(landing_model_y, "Go_NoGo/landing_y.rds")

throw_dist_model <- lm(throw_dist ~ (landing_location_x + landing_location_y)^2, data=data)
p3 <- predict(throw_dist_model, HOLDOUT)
plot(y=HOLDOUT$throw_dist, x=p3); abline(0,1)
```




```
# summary(throw_dist_model)
# not a perfect model by any means but should be sufficient for this UI currently
saveRDS(throw_dist_model, "Go_NoGo/throw_dist.rds")

time_to_catch_model <- lm(time_to_catch ~ launch_direction + launch_speed + launch_angle, data=data)
# summary(time_to_catch_model)
saveRDS(time_to_catch_model, "Go_NoGo/time_to_catch.rds")

ball_vx_model <- lm(ball_vx_0 ~ launch_direction + launch_speed + launch_angle, data=data)
# summary(ball_vx_model)
saveRDS(ball_vx_model, "Go_NoGo/ball_vx.rds")

ball_vy_model <- lm(ball_vy_0 ~ launch_direction + launch_speed + launch_angle, data=data)
# summary(ball_vy_model)
saveRDS(ball_vy_model, "Go_NoGo/ball_vy.rds")

ball_vz_model <- lm(ball_vz_0 ~ launch_direction + launch_speed + launch_angle, data=data)
# summary(ball_vz_model)
saveRDS(ball_vz_model, "Go_NoGo/ball_vz.rds")

ball_v_model <- lm(ball_v_0 ~ launch_direction + launch_speed + launch_angle, data=data)
# summary(ball_v_model)
saveRDS(ball_v_model, "Go_NoGo/ball_v.rds")

ball_ax_model <- lm(ball_ax_0 ~ launch_direction + launch_speed + launch_angle, data=data)
# summary(ball_ax_model)
saveRDS(ball_ax_model, "Go_NoGo/ball_ax.rds")

ball_ay_model <- lm(ball_ay_0 ~ launch_direction + launch_speed + launch_angle, data=data)
# summary(ball_ay_model)
saveRDS(ball_ay_model, "Go_NoGo/ball_ay.rds")
```

```r
ball_az_model <- lm(ball_az_0 ~ launch_direction + launch_speed + launch_angle, data=data)
# summary(ball_az_model)
saveRDS(ball_az_model, "Go_NoGo/ball_az.rds")

ball_apex_x_model <- lm(ball_apex_x ~ launch_direction + launch_speed + launch_angle, data=data)
# summary(ball_apex_x_model)
saveRDS(ball_apex_x_model, "Go_NoGo/ball_apex_x.rds")

ball_apex_y_model <- lm(ball_apex_y ~ launch_direction + launch_speed + launch_angle, data=data)
# summary(ball_apex_y_model)
saveRDS(ball_apex_y_model, "Go_NoGo/ball_apex_y.rds")

ball_apex_z_model <- lm(ball_apex_z ~ launch_direction + launch_speed + launch_angle, data=data)
# summary(ball_apex_z_model)
saveRDS(ball_apex_z_model, "Go_NoGo/ball_apex_z.rds")
```

---

## Goal 3 Code/Process (Also used in UI)

---

```r
probs <- predict(FINAL, type="prob")
preds <- predict(FINAL,tags)
grades <- cbind(tags, preds)
grades$correct <-  ifelse(grades$runs_on_play == grades$preds, "C", "NC")
grades$allowT <- ifelse(grades$runs_on_play=="run",1,0)
grades$allowP <- ifelse(grades$preds=="run",1,0)
fielderT <- aggregate(allowT~fielder_id, data=grades, FUN=mean)
fielderP <- aggregate(allowP~fielder_id, data=grades, FUN=mean)

red_yellow <- quantile(probs$run, 0.25)
yellow_green <- quantile(probs$run, 0.75)

fielders <- merge(fielderT,fielderP, by="fielder_id")
fielders$grade <- ifelse(fielders$allowT < red_yellow, "red", ifelse(fielders$allowT > yellow_green, "g

average_run <- mean(probs$run)
sd_run <- sd(probs$run)

fielders$scale<-ifelse(fielders$allowT >= average_run+3*sd_run, "20",
                       ifelse(fielders$allowT >= average_run+2.5*sd_run, "25",
                              ifelse(fielders$allowT >= average_run+2*sd_run, "30",
                                     ifelse(fielders$allowT >= average_run+1.5*sd_run,"35",
                                            ifelse(fielders$allowT >= average_run+1*sd_run, "40",
                                                   ifelse(fielders$allowT >= average_run+0.5*sd_run,"45
                                                          ifelse(fielders$allowT >= average_run, "50",
                                                                 ifelse(fielders$allowT >= average_run-
                                                                        ifelse(fielders$allowT >= averag
                                                                               ifelse(fielders$allowT >=
                                                                                      ifelse(fielders$a
                                                                                             ifelse(fiel
```

```
))))))))))
```

```
table(fielders$grade)
```

```
##
##  green    red yellow
##    169     90    244
```

```
table(fielders$scale)
```

```
##
##  45  50  55  60  65  70  80
## 178 109 126  50  10   1  29
```

Utilizing the 25th percentile for allowing runners to score for the threshold of red to yellow and 75th percentile for the threshold of yellow to green, I was able to generate three levels to evaluate outfielders in this dataset. The three levels are similar to a stoplight and would be used to help our coaching staff and runners to make decisions in games. The resulting thresholds grouped 169 outfielders as green (should send runners), 244 as yellow, and 90 as red (difficult to send in most situations).

Also, using the mean and standard deviation for the probability of a run scoring I was able to grade the outfielders in this dataset on the typical 20/80 scouting scale. The scale allows us to see where the outfielders in this dataset would fall if we scouted their ability to prevent runners from scoring. Since the probability of a run scoring is not perfectly normal the approach above could be refined more in the future.