R Coding Sample (Royals Project)

Nick Skiera

2023-11-14

Querying Prompts

Using the attached dataset of throws to first base, please answer the following questions. Please utilize SQL, R or Python to aggregate the data. Attach all code and visualizations that you used throughout your entire process.

```r
dat <- read.csv("~/Downloads/dataset_2024.csv")
head(dat)
```

```
##   throw_id team_id fielder_id fielder_position thrower_id thrower_position
## 1        3      11        400                6        400                6
## 2        6      11        228                5        390                4
## 3        7       8        415                4        415                4
## 4        8       8        308                1        308                1
## 5       10       1        314                4        300                6
## 6       11       2        312                4        312                4
##   receiver_id receiver_position exchange_time throw_pos_x throw_pos_y
## 1          63                 3         1.533  -60.116123   132.18728
## 2          63                 3         0.534   -0.562563   122.82194
## 3         143                 3         1.266    1.598751   123.03693
## 4         143                 3         1.800   25.403185    59.92459
## 5         514                 3         0.733    8.957441   126.16588
## 6         695                 3         1.200   15.075073   138.72246
##   throw_velo_x throw_velo_y throw_velo_z batter_pos_x_at_throw
## 1     60.04756   -33.596364     8.583470             26.91291
## 2     51.29392   -45.316519     8.831870             43.39641
## 3     50.04965   -41.717447     5.986785             25.30078
## 4     24.01967     3.289158    12.783662             19.69922
## 5     51.63067   -63.361730     7.811719             40.11830
## 6     41.26329   -62.250650     6.230779             39.70496
##   batter_pos_y_at_throw batter_velo_at_throw bounce_pos_x bounce_pos_y
## 1              26.55610             25.78753     38.06139     69.47463
## 2              44.29049             28.64788           NA           NA
## 3              29.97072             25.13043           NA           NA
## 4              15.24991             18.50923           NA           NA
## 5              41.43340             27.31851           NA           NA
## 6              41.80575             27.57804           NA           NA
##   bounce_velo_x bounce_velo_y bounce_velo_z receiver_pos_x receiver_pos_y
## 1      32.81451      -25.4525      7.330194       56.19932       60.18778
## 2            NA            NA            NA       59.01313       65.65794
## 3            NA            NA            NA       61.03749       64.42913
## 4            NA            NA            NA       63.14208       65.64872
## 5            NA            NA            NA       59.77040       64.04606
```

```
## 6            NA            NA            NA       59.62286       63.94086
##   receiver_dist_from_1b throw_deflected_by_receiver start_state end_state
## 1              8.202015                           0        ____1     1___1
## 2              5.047566                           0        1___1     1___2
## 3              2.719261                           0        123_1     ____3
## 4              2.069797                           0        ____1     ____2
## 5              3.890504                           0        123_1     ____3
## 6              4.028031                           0        ____0     ____1
##   runs_on_play batter_result
## 1            0         first
## 2            0         first
## 3            0           out
## 4            0           out
## 5            0           out
## 6            0           out
```

1. Which 5 infielders had the quickest exchange times on throws to first base?

```r
infielders <- c(2,4,5,6)
dat %>%
  filter(thrower_position %in% infielders) %>% #filtering out outfielders and pitchers
  filter(exchange_time > 0) %>%
  dplyr::select(thrower_id, thrower_position, exchange_time) %>%
  arrange(exchange_time) %>%
  top_n(-5)
```

```
## Selecting by exchange_time

##   thrower_id thrower_position exchange_time
## 1        592                6         0.034
## 2        396                5         0.067
## 3        112                4         0.167
## 4        159                4         0.167
## 5         85                4         0.200
## 6        687                5         0.200
## 7        267                4         0.200
## 8        190                2         0.200
```

The 5 infielders that are not pitchers with the quickest exchange times are throwers 592, 396, 112/159 and 85/687/267/190. I removed pitchers because they are not position players we typically evaluate when looking at fielding. I removed exchange times that are 0 because those are likely glove flips or barehanded players.

```r
infielders_w_pitchers <- c(1,2,4,5,6)
dat %>%
  filter(thrower_position %in% infielders_w_pitchers) %>% #filtering out only outfielders
  filter(exchange_time > 0) %>%
  dplyr::select(thrower_id, thrower_position, exchange_time) %>%
  arrange(exchange_time) %>%
  top_n(-5)
```

```
## Selecting by exchange_time

##   thrower_id thrower_position exchange_time
## 1          1                1         0.033
## 2        592                6         0.034
## 3        292                1         0.034
## 4        658                1         0.067
```

```
## 5        396                  5          0.067
```

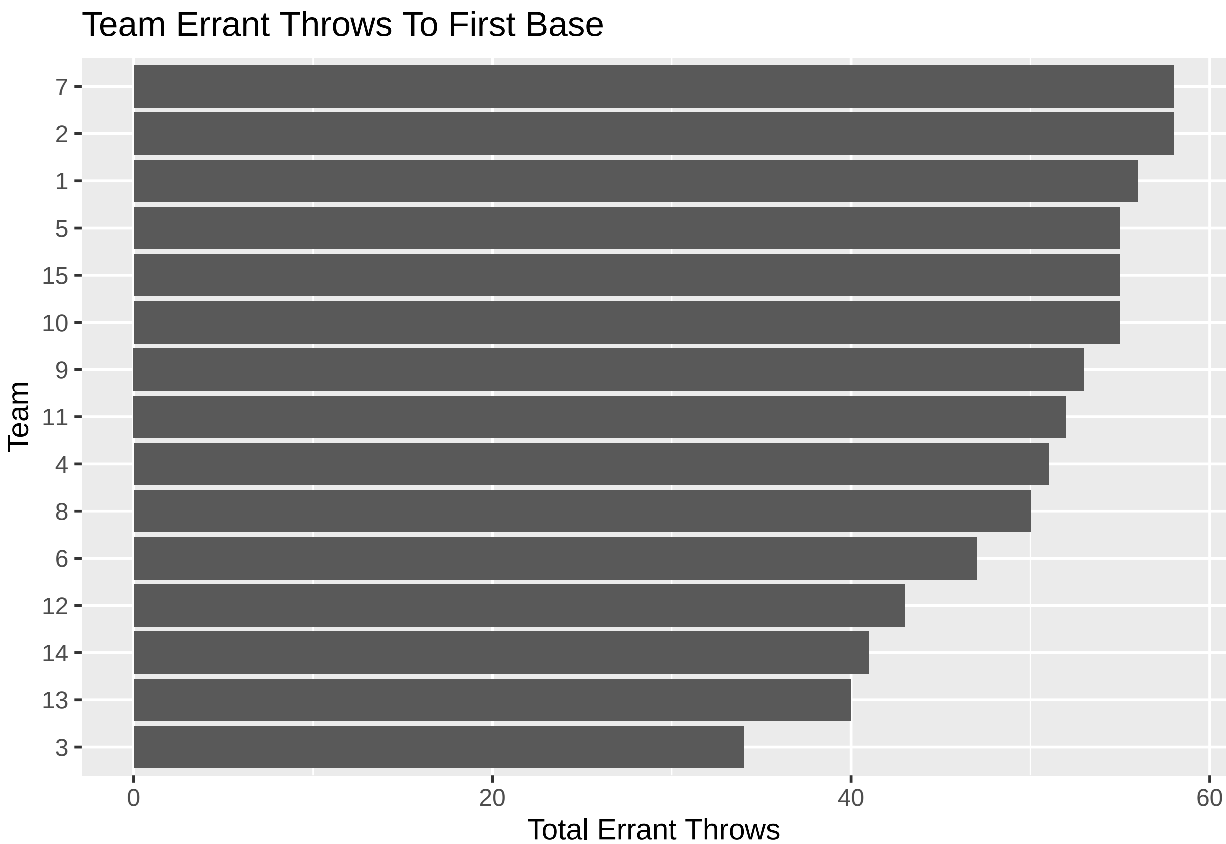
However, if we do include pitchers then the infielders with the quickest exchange times on throws to first are throwers 1, 592, 292, 658 and 396

2. The infield coach wants to see which teams made the most errant throws to first base. An errant throw is described as a throw that bounced and resulted in the runner being safe. Please create a basic visual that you would present to the infield coach to present your findings.

```
err_throw <- dat %>%
                filter(thrower_position %in% infielders_w_pitchers) %>%
                mutate(errant_throw = as.factor(ifelse(!is.na(bounce_pos_x) & batter_result != "out",1
                filter(errant_throw == 1) %>% #will remove all throws that don't bounce
                arrange(team_id)

team_err_throws <- c()
for(i in 1:15) {
  team_err_throws[i] <- sum(err_throw$team_id == i)
  }

head(df_err <- data.frame(
  team_id = as.character(unique(err_throw$team_id)),
  team_err_throws = team_err_throws) %>%
  mutate(team_order = fct_reorder(team_id, team_err_throws)))
```

```
##   team_id team_err_throws team_order
## 1       1              56          1
## 2       2              58          2
## 3       3              34          3
## 4       4              51          4
## 5       5              55          5
## 6       6              47          6
```

```
ggplot(df_err, aes(x = team_order, y = team_err_throws)) +
  geom_bar(stat = "identity") +
  labs(title = "Team Errant Throws To First Base") +
  xlab("Team") +
  ylab("Total Errant Throws") +
  coord_flip()
```



3. Looking at all infield throws to first base, given that the distance of the throw to first base was in the top 90th percentile, what team had the best average exchange time? Which team had the largest variation in exchange time on these throws?

```
dist_throw_1b <- dat %>%
  filter(thrower_position %in% infielders_w_pitchers) %>%
  filter(exchange_time > 0) %>%
  mutate(distance_of_throw = sqrt((receiver_pos_x - throw_pos_x)^2 + (receiver_pos_y - throw_pos_y)^2))
  mutate(percentile_rank = ntile(desc(distance_of_throw), 10)) %>%
  arrange(desc(distance_of_throw))

head(throws_90_pctile <- dist_throw_1b %>%
  filter(percentile_rank == 1))
```

```
##   throw_id team_id fielder_id fielder_position thrower_id thrower_position
## 1     1313       5        112                4        112                4
## 2     2778       5        446                4        446                4
## 3    18712       6        566                6        566                6
## 4    15975      15        426                5        426                5
## 5    27885      15        159                4        159                4
## 6    25832       7        673                4        673                4
##   receiver_id receiver_position exchange_time throw_pos_x throw_pos_y
## 1         766                 3         1.767  -104.49645    403.5053
## 2         766                 3         0.833    91.56045    403.6370
## 3          25                 3         1.734   132.01699    375.0706
## 4         568                 3         0.666   169.96265    329.2181
## 5         313                 3         0.866   214.40527    283.8712
## 6         615                 3         2.267   223.92484    272.7766
```

```
##   throw_velo_x throw_velo_y throw_velo_z batter_pos_x_at_throw
## 1    -24.44178    -25.40783     6.780555              34.75114
## 2    -23.14839     16.59581    11.931548              39.28094
## 3    -20.50401     25.14487     3.784122              33.14163
## 4    -24.38325     19.03692    -4.884555              19.41157
## 5    -23.56285     27.72982     6.858485              34.98759
## 6    -27.55181     34.07074     8.871604              31.61921
##   batter_pos_y_at_throw batter_velo_at_throw bounce_pos_x bounce_pos_y
## 1              34.31264             26.31457           NA           NA
## 2              38.94313             28.01125           NA           NA
## 3              34.08661             25.23992           NA           NA
## 4              20.03403             23.30695           NA           NA
## 5              33.14406             24.66136           NA           NA
## 6              27.91198             19.46385           NA           NA
##   bounce_velo_x bounce_velo_y bounce_velo_z receiver_pos_x receiver_pos_y
## 1            NA            NA            NA       60.32098       63.92279
## 2            NA            NA            NA       60.16601       64.60350
## 3            NA            NA            NA       60.44282       63.03786
## 4            NA            NA            NA       59.42645       63.54653
## 5            NA            NA            NA       61.28857       63.59752
## 6            NA            NA            NA       62.40411       64.56134
##   receiver_dist_from_1b throw_deflected_by_receiver start_state end_state
## 1              3.330693                           0       1___1     ____3
## 2              3.604850                           0       ____2     ____3
## 3              3.252932                           0       _2__2     ____3
## 4              4.214187                           0       ____0     ____1
## 5              2.351422                           0       ____0     ____1
## 6              1.541444                           0       ____0     ____1
##   runs_on_play batter_result distance_of_throw percentile_rank
## 1            0           out          377.4667               1
## 2            0           out          340.4839               1
## 3            0           out          320.1363               1
## 4            0           out          287.7492               1
## 5            0           out          268.2634               1
## 6            0           out          263.5195               1
```

```
avg_exch_time <- c()
for (i in 1:15) {
  avg_exch_time[i] <- mean(throws_90_pctile$exchange_time[which(throws_90_pctile$team_id == i)])
}
var_exch_time <- c()
for (i in 1:15) {
  var_exch_time[i] <- var(throws_90_pctile$exchange_time[which(throws_90_pctile$team_id == i)])
}

(df_throw_dist <- data.frame(
  team_id = as.character(unique(throws_90_pctile$team_id)),
  avg_exch_time = avg_exch_time,
  var_exch_time = var_exch_time) %>%
  arrange(avg_exch_time))
```

```
##    team_id avg_exch_time var_exch_time
## 1        2      1.132352     0.1388616
## 2       14      1.157395     0.1198779
## 3        6      1.161526     0.1090549
```

```
## 4         5      1.164255      0.1383404
## 5         4      1.172265      0.1290435
## 6        13      1.185328      0.1142765
## 7        10      1.193778      0.1687950
## 8         7      1.194088      0.1626194
## 9        12      1.202600      0.1338400
## 10        3      1.212613      0.1256264
## 11       15      1.215025      0.1348278
## 12        8      1.228241      0.1462788
## 13        9      1.241426      0.1387692
## 14       11      1.255300      0.1477178
## 15        1      1.322391      0.1051598
```

```
df_throw_dist$team_id[which(df_throw_dist$avg_exch_time == min(df_throw_dist$avg_exch_time))]
```

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

```
df_throw_dist$team_id[which(df_throw_dist$var_exch_time == max(df_throw_dist$var_exch_time))]
```

```
## [1] "10"
```

Team 2 had the lowest average exchange time at 1.132352 and Team 10 had the largest variation in exchange time at 0.1687950.

4. Given that a throw was made less than 100 feet from first base, is there a correlation between throw velocity and throw distance? Provide a basic visual alongside a brief explanation.

```
head(short_throw <- dist_throw_1b %>%
  filter(distance_of_throw < 100) %>%
  mutate(throw_velo = sqrt(throw_velo_x^2 + throw_velo_y^2 + throw_velo_z^2)))
```

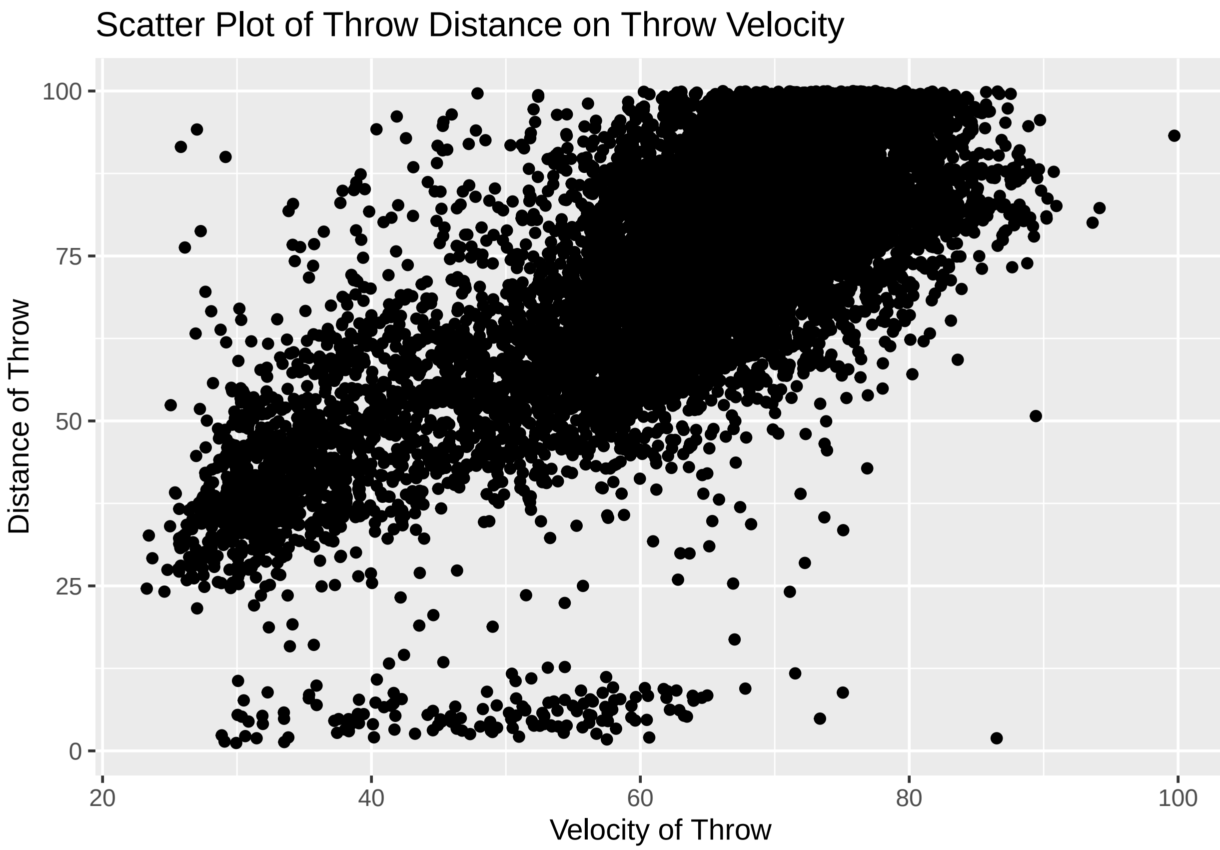
```
##   throw_id team_id fielder_id fielder_position thrower_id thrower_position
## 1    14638       8         26                5         26                5
## 2    24105       7        757                6        757                6
## 3    27977       4        379                6        379                6
## 4    18143       3        653                5        653                5
## 5    24302       7        757                6        757                6
## 6     9638       3        653                5        653                5
##   receiver_id receiver_position exchange_time throw_pos_x throw_pos_y
## 1         863                 3         0.933   -37.95813    73.84991
## 2         820                 3         0.900   -32.63508    97.67744
## 3         397                 3         0.633   -20.44641   120.65644
## 4         103                 3         1.133   -38.61237    75.88515
## 5         339                 3         0.634   -20.72630   121.82080
## 6         474                 3         1.300   -32.62485    99.70371
##   throw_velo_x throw_velo_y throw_velo_z batter_pos_x_at_throw
## 1     77.36064    -3.579660     2.471836              36.79104
## 2     72.93405   -18.626071     9.118424              34.14535
## 3     54.93388   -35.254666    10.689079              26.00501
## 4     79.23197    -5.978675     6.226513              34.69233
## 5     61.85791   -39.462358     9.019587              36.83248
## 6     67.02635   -22.415908     7.923176              33.04008
##   batter_pos_y_at_throw batter_velo_at_throw bounce_pos_x bounce_pos_y
## 1              38.84842             28.42306     43.76895     66.43157
## 2              36.28389             28.56228           NA           NA
## 3              27.19864             22.19235           NA           NA
## 4              35.19163             25.34513           NA           NA
```

```
## 5           37.74694          25.23409        NA         NA
## 6           33.88707          24.75661        NA         NA
##   bounce_velo_x bounce_velo_y bounce_velo_z receiver_pos_x receiver_pos_y
## 1      52.96815     -8.404934      6.527076       61.63916       64.89060
## 2            NA            NA            NA       62.49966       66.90812
## 3            NA            NA            NA       61.90322       63.96179
## 4            NA            NA            NA       60.53937       63.13401
## 5            NA            NA            NA       61.08303       64.39102
## 6            NA            NA            NA       61.72682       66.75446
##   receiver_dist_from_1b throw_deflected_by_receiver start_state end_state
## 1              2.359400                           0       ____2     ____3
## 2              3.461598                           0       ____0     ____1
## 3              1.766031                           0       ____2     ____3
## 4              3.141200                           0       ____2     ____3
## 5              2.664716                           0       ____1     ____2
## 6              3.655276                           0       _2__2     ____3
##   runs_on_play batter_result distance_of_throw percentile_rank throw_velo
## 1            0           out          99.99945               5   77.48285
## 2            0           out          99.98685               5   75.82514
## 3            0           out          99.97871               5   66.14286
## 4            0           out          99.96829               5   79.70081
## 5            0           out          99.95472               5   73.92585
## 6            0           out          99.93943               5   71.11808
```

```r
cor(short_throw$throw_velo, short_throw$distance_of_throw, use ="complete.obs")
```

```
## [1] 0.7380965
```

```r
ggplot(short_throw, aes(x = throw_velo, y = distance_of_throw)) +
  geom_point() +
  labs(title = "Scatter Plot of Throw Distance on Throw Velocity") +
  xlab("Velocity of Throw") +
  ylab("Distance of Throw")
```



We can see based on the plot that, outside of a few outliers, there is clearly a strong positive relationship between the distance of the throw and the velocity of the throw with a correlation of 0.7380965.

Modeling Project

While often routine, an infielder making a timely and accurate throw to first base is a skill that is critical to the outcome of a game. Arm strength, exchange time, velocity, and the first baseman's ability to receive an errant throw all determine whether or not an out is made on the play. We have attached a dataset of throws to first base for you to analyze. We would like to see you build some sort of model based on this data that evaluates the talent of a subset of infielders. We will be evaluating your submission on four components of your output:
1. Your modeling approach and creativity.
2. Your model evaluation process.
3. Your ability to perform a skill assessment of some group of players involved.
4. Your creation of a player-evaluation tool or other presentation layer that would be presented to a less technical audience.

If there is additional information that you think would clarify this problem or strategies you would implement if you had more time, please detail that as well.
This project should take approximately 8-12 hours to complete. Please include all code written (in either R or Python) for this analysis, as well as your final product.

```
new_dat <- dat %>%
  filter(thrower_position %in% infielders) %>% #taking out outfielders and pitchers
  subset(thrower_position == fielder_position) %>% #isolates data to just fielders who made throws to 1
  mutate(thrower_err_throw = as.factor(ifelse(!is.na(bounce_pos_x), 1, 0))) %>% #variable for thrower b
  mutate(first_base_save = as.factor(ifelse(!is.na(bounce_pos_x) & batter_result == "out", 1, 0))) %>% #
  mutate(throw_velo = sqrt(throw_velo_x^2 + throw_velo_y^2 + throw_velo_z^2)) %>% #create overall throw
```

```r
  mutate(num_outs = as.numeric(substr(start_state, 5, 5))) %>% #create variable for number of outs
  mutate(runner_on_third = as.factor(ifelse(substr(start_state, 3, 3) == "3", 1, 0))) %>% #create varia
  mutate(distance_of_throw = sqrt((receiver_pos_x - throw_pos_x)^2 + (receiver_pos_y - throw_pos_y)^2))
  mutate(batter_result = as.factor(ifelse(batter_result == "out", 1, 0))) %>% #changes batter_result to
  remove_rownames() %>% column_to_rownames( var = "throw_id") %>%
  dplyr::select(-c(bounce_pos_x, bounce_pos_y, bounce_velo_x, bounce_velo_y, bounce_velo_z, throw_velo_
new_dat <- na.omit(new_dat)

head(new_dat)
```

```
##    team_id thrower_id thrower_position receiver_id exchange_time
## 3       11        400                6          63         1.533
## 7        8        415                4         143         1.266
## 11       2        312                4         695         1.200
## 14       2        396                5         695         0.966
## 15       1        314                4         514         0.967
## 16       1        201                5         514         1.633
##    batter_pos_x_at_throw batter_pos_y_at_throw batter_velo_at_throw
## 3               26.91291              26.55610             25.78753
## 7               25.30078              29.97072             25.13043
## 11              39.70496              41.80575             27.57804
## 14              36.31824              39.76688             29.64266
## 15              38.21505              38.47144             26.57897
## 16              27.59490              29.82228             24.41569
##    receiver_dist_from_1b throw_deflected_by_receiver runs_on_play batter_result
## 3               8.202015                           0            0             0
## 7               2.719261                           0            0             1
## 11              4.028031                           0            0             1
## 14              4.419242                           0            0             1
## 15              2.929510                           0            0             1
## 16             11.914992                           0            1             0
##    thrower_err_throw first_base_save throw_velo num_outs runner_on_third
## 3                  1               0   69.34047        1               0
## 7                  0               0   65.43053        1               1
## 11                 0               0   74.94414        0               0
## 14                 0               0   70.72622        0               0
## 15                 0               0   69.82297        0               0
## 16                 1               0   75.34457        1               1
##    distance_of_throw
## 3          136.79624
## 7           83.47357
## 11          87.04478
## 14          70.42454
## 15          66.81671
## 16         140.20326
```

```r
dim(new_dat)
```

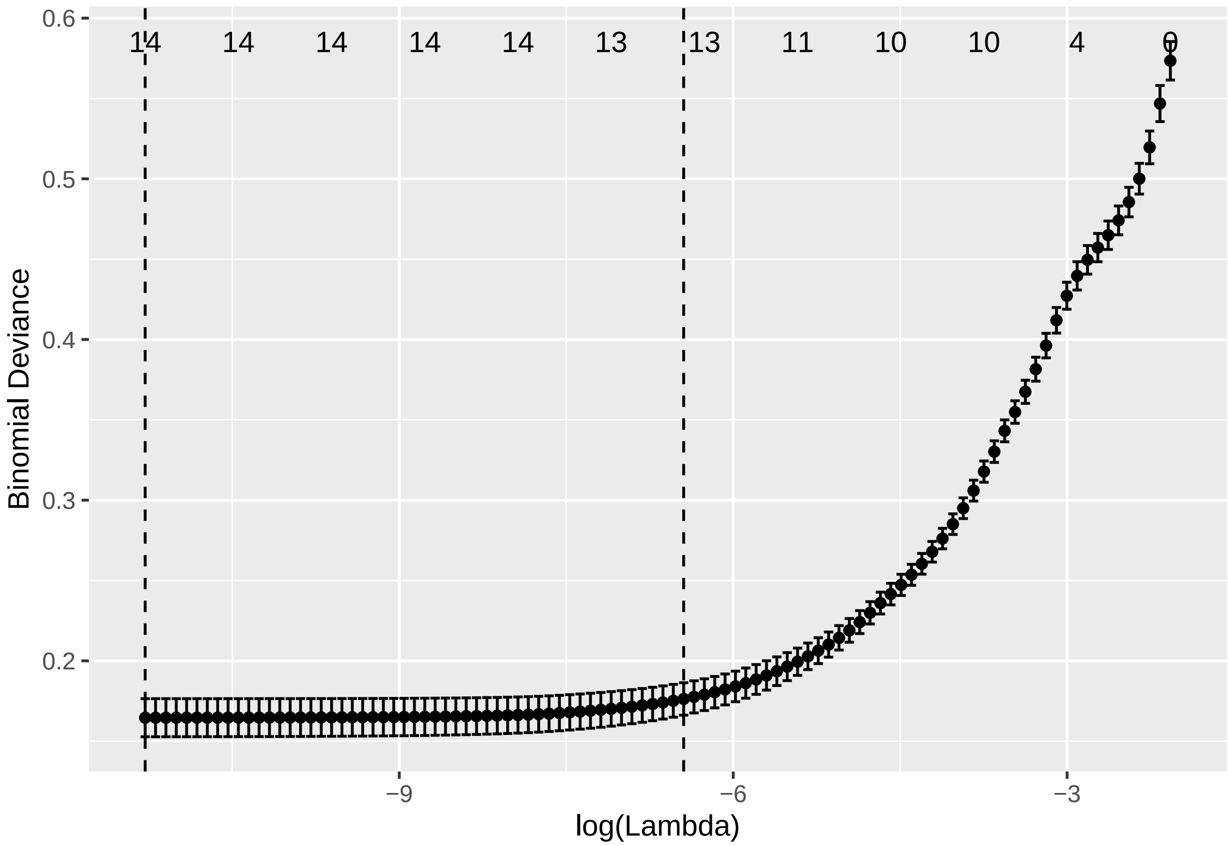
```
## [1] 15385    18
```

I removed outfielders and pitchers because we are evaluating infielders and the main job of pitchers is not fielding baseballs and making good throws to first base. I also removed plays where the thrower didn't field the ball so we can eliminate abnormal plays such as ricochets and double plays involving multiple infielders other than the receiver. I created the variables thrower errant throw, first base save, throw velocity, number of outs, runner on third and distance of throw because I believe that his will add a complexity while also reducing the amount of variables in the data set. I created thrower errant throw because I wanted to eliminate the bounce coordinate variables because they contain NAs. First base save shows us if the first basemen saved the errant throw. I used the velocity coordinates on the throw to create an overall throwing velocity variable which will reduce the dimensionality while maintaining the value of the variables. I wanted to remove the variables start_state and end_state because they are character variables so I extracted the data I deemed important which was number of outs and if there was a runner on third. The number of outs and having a runner on third can add pressure to make a good throw depending on the situation. I also wanted to remove the thrower and receiver coordinates but to maintain their importance I created the distance of throw variable. Finally, I mutated the variable batter_result to be a response factor of whether the batter was thrown out/success (1) or reached base/failure (0).

```
set.seed(100)

#splitting data into training and test data for cross validation
train_inx <- sample(seq_len(nrow(new_dat)), size = 0.7*nrow(new_dat))
dat_train <- new_dat[train_inx, -c(1, 2, 4)] #removing player ids manually
dat_test <- new_dat[-train_inx,-c(1, 2, 4)]
```

I took out all identification variables (team_id, thrower_id and receiver_id) because these are not predictors. We have 14 predictors. I split the data set into training and test data to use in cross validation and to compare models in-sample and out-of-sample classification accuracy. I hope to find a model with a high classification accuracy in both to have a good fit. A model with a high in-sample classification accuracy and low out-of-sample classification accuracy is over fit because it conforms to the data it was built upon well but not to new data. The opposite scenario means that the model would be under fit because it is able to conform to any data but not the data it was built upon. This is similar to bias-variance trade-off.

```
set.seed(100)

cv_lasso <- cv.glmnet(y = dat_train$batter_result, x = as.matrix(dat_train[,-9]), alpha = 1, nfolds = 1
autoplot(cv_lasso)
```




```r
lasso_1se <- glmnet(y = dat_train$batter_result, x = as.matrix(dat_train[,-9]), alpha = 1, lambda = cv_l
coef(lasso_1se)
```

```
## 15 x 1 sparse Matrix of class "dgCMatrix"
##                                   s0
## (Intercept)                27.29658281
## thrower_position           -0.08079468
## exchange_time               0.75377407
## batter_pos_x_at_throw      -0.18328416
## batter_pos_y_at_throw      -0.11338428
## batter_velo_at_throw       -0.43075000
## receiver_dist_from_1b      -0.07178555
## throw_deflected_by_receiver -6.70331836
## runs_on_play               -2.13608634
## thrower_err_throw          -5.39103958
## first_base_save             9.15897327
## throw_velo                  0.07547682
## num_outs                    .
## runner_on_third             0.77420350
## distance_of_throw          -0.07449516
```

```r
pred_lasso <- predict(lasso_1se, newx = as.matrix(dat_train[,-9]), type = "class")
mean(pred_lasso == dat_train$batter_result)
```

```
## [1] 0.9700994
```

I used cross validation in lasso regression with the desire for dimension reduction and see that it actually eliminated the variable num_outs from the model. I will take num_outs out of the data frame for the rest of the tested models. We now have 13 predictors

```r
dat_train1 <- dat_train[,-13]
dat_test1 <- dat_test[,-13]
```

```r
mod_logit <- glm(batter_result ~ ., data = dat_train1, family = "binomial")
summary(mod_logit)
```

```
##
## Call:
## glm(formula = batter_result ~ ., family = "binomial", data = dat_train1)
##
## Coefficients:
##                              Estimate Std. Error z value Pr(>|z|)
## (Intercept)                 35.509032   1.614840  21.989  < 2e-16 ***
## thrower_position            -0.175949   0.079499  -2.213 0.026882 *
## exchange_time                0.909231   0.157887   5.759 8.47e-09 ***
## batter_pos_x_at_throw       -0.239710   0.019398 -12.357  < 2e-16 ***
## batter_pos_y_at_throw       -0.150832   0.019687  -7.661 1.84e-14 ***
## batter_velo_at_throw        -0.569881   0.038582 -14.771  < 2e-16 ***
## receiver_dist_from_1b       -0.089965   0.019775  -4.549 5.38e-06 ***
## throw_deflected_by_receiver  -9.584015   0.877126 -10.927  < 2e-16 ***
## runs_on_play                -3.743544   0.623854  -6.001 1.97e-09 ***
## thrower_err_throw1         -20.946685 382.564552  -0.055 0.956335
## first_base_save1            44.595619 574.973767   0.078 0.938177
## throw_velo                   0.110000   0.009185  11.976  < 2e-16 ***
## runner_on_third1             2.211648   0.596827   3.706 0.000211 ***
## distance_of_throw           -0.102763   0.005325 -19.300  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6185.5  on 10768  degrees of freedom
## Residual deviance: 1799.7  on 10755  degrees of freedom
## AIC: 1827.7
##
## Number of Fisher Scoring iterations: 18
```

```r
pred_logit <- predict(mod_logit, type = "response") > 0.5
mean(pred_logit == as.numeric(dat_train1$batter_result))
```

```
## [1] 0.0218219
```

```r
out_pred_logit <- predict(mod_logit, newdata = dat_test1, type = "response") > 0.5
mean(out_pred_logit == as.numeric(dat_test1$batter_result))
```

```
## [1] 0.02058059
```

We can see that a logistic model is not great with an in-sample accuracy of 0.0218219 and out-of-sample accuracy of 0.02058059. This makes sense as this model is not cross validated. We will move forward to a K-Nearest-Neighbors model using cross validation.

```r
set.seed(100)

trctrl <- trainControl(method = "cv", number = 10)
knn_fit <- train(batter_result ~ ., data = dat_train1, method = "knn", trControl = trctrl, tuneGrid = e
```

```
knn_fit$bestTune
```

```
##   k
## 7 7
```

```
mod_knn <- knn(train = dat_train1[,-9], cl = dat_train1$batter_result, test = dat_train1[,-9], k = knn_
mean(mod_knn == dat_train1$batter_result)
```

```
## [1] 0.9605349
```

```
mean(mod_knn == dat_test1$batter_result)
```

```
## [1] 0.8704615
```

This prediction is a lot better with an in-sample accuracy of 0.9605349 and out-of-sample accuracy of 0.8704615. This model will work but we will test an LDA model to see if we can still improve.

```
mod_lda <- lda(batter_result ~., data = dat_train1, prior = rep(1, 2)/2)
pred_lda <- predict(mod_lda, dat_train1)

mean(pred_lda$class == dat_train1$batter_result)
```

```
## [1] 0.9594206
```

```
mean(pred_lda$class == dat_test1$batter_result)
```

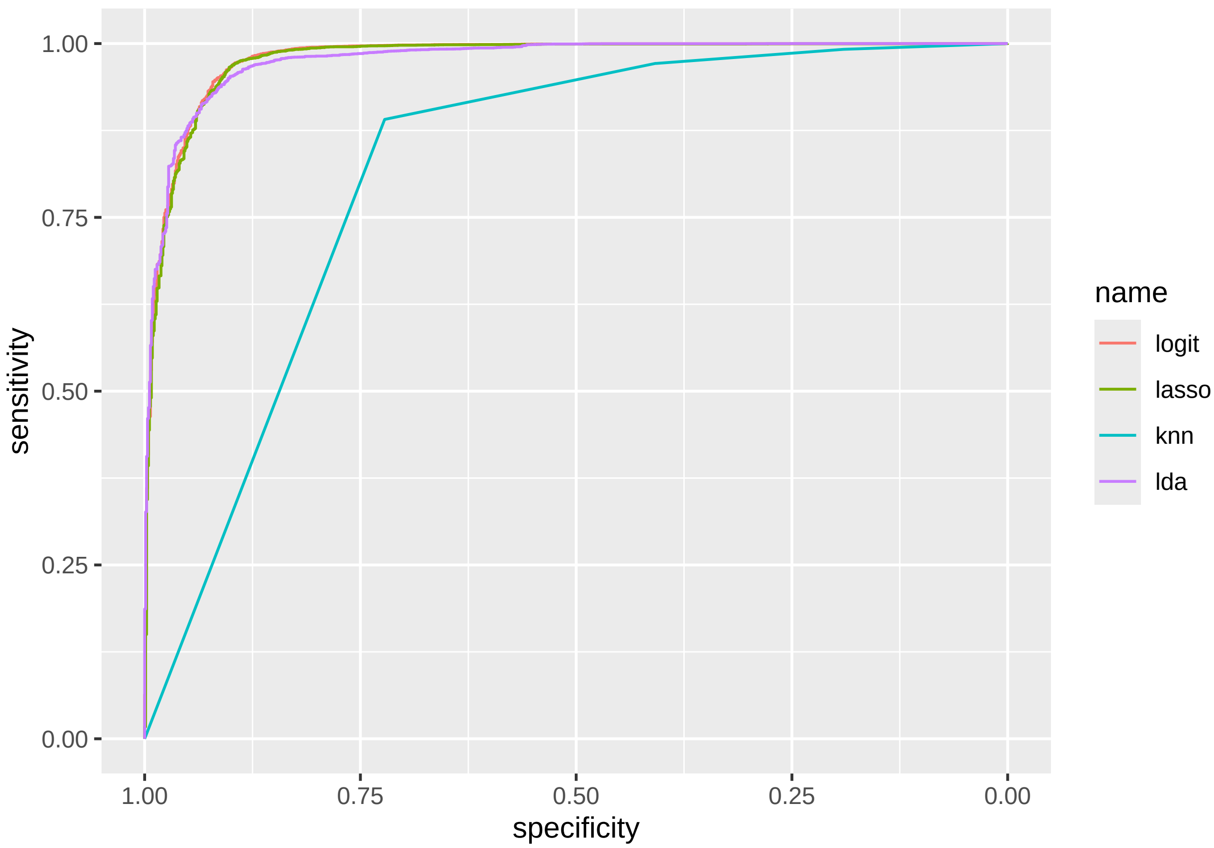
```
## [1] 0.8699044
```

LDA predicts worse than KNN but still great with an in-sample accuracy of 0.9594206 and out-of-sample accuracy of 0.8699044. Now I will combine all of these models on an in-sample ROC plot and calculate the Areas Under the Curves.

```
set.seed(100)

prob_logit <- predict(mod_logit, type = "response")
prob_lass <- predict(lasso_1se, newx = as.matrix(dat_train[,-9]), type = "response")
prob_knn <- 1 - attributes(knn(train = dat_train1[,-9], cl = dat_train1$batter_result, test = dat_train
prob_lda <- predict(mod_lda, dat_train1)$posterior[,2]
df_roc <- data.frame(logit = prob_logit,
                     lasso = prob_lass,
                     knn = prob_knn,
                     lda = prob_lda,
                     batter_result = dat_train$batter_result)
names(df_roc) <- c("logit", "lasso", "knn", "lda", "batter_result" )
rocobj <- roc(batter_result ~ logit + lasso + knn + lda, data = df_roc)
```

```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls > cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```

```r
ggroc(rocobj)
```




```r
df_auc <- data.frame(logit = auc(rocobj$logit),
                     lasso = auc(rocobj$lasso),
                     knn = auc(rocobj$knn),
                     lda = auc(rocobj$lda))
df_auc
```

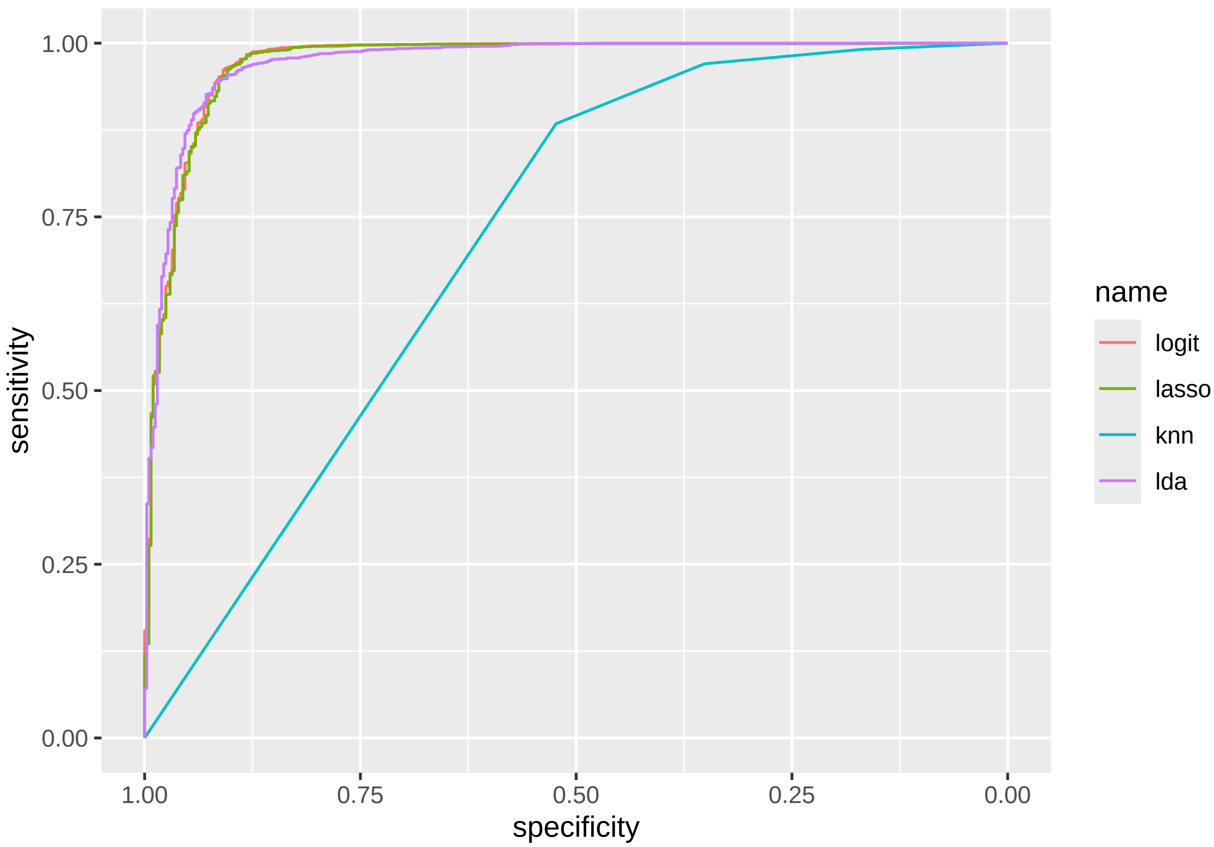
```
##       logit     lasso       knn       lda
## 1 0.9792219 0.9776247 0.8193673 0.9768522
```

The logistic model actually has the highest AUC with a 0.9792219. However, due to the low accuracies, I will avoid this model. This might be due to the nature of ROC and how it assigns scores. However, LDA has a great AUC with a 0.9768522 which pairs well with its classification accuracy.

```r
prob_logit_test <- predict(mod_logit, newdata = dat_test1, type = "response")
prob_lass_test <- predict(lasso_1se, newx = as.matrix(dat_test[,-9]), type = "response")
prob_knn_test <- 1 - attributes(knn(train = dat_train1[,-9], cl = dat_train1$batter_result, test = dat_
prob_lda_test <- predict(mod_lda, dat_test1)$posterior[,2]

df_roc_test <- data.frame(logit = prob_logit_test,
                          lasso = prob_lass_test,
                          knn = prob_knn_test,
                          lda = prob_lda_test,
                          Test = dat_test$batter_result)
names(df_roc_test) <- c("logit", "lasso", "knn", "lda", "batter_result" )
rocobj_test <- roc(batter_result ~ logit + lasso + knn + lda, data = df_roc_test)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
## Setting levels: control = 0, case = 1
## Setting direction: controls > cases
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
ggroc(rocobj_test)
```




```
df_auc_test <- data.frame(logit = auc(rocobj_test$logit),
                          lasso = auc(rocobj_test$lasso),
                          knn = auc(rocobj_test$knn),
                          lda = auc(rocobj_test$lda))
df_auc_test
```

```
##       logit     lasso       knn      lda
## 1 0.9738865 0.9723764 0.7172281 0.973346
```

We can also see that LDA performs great with an AUC of 0.973346 that pairs well with its out-of-sample classification accuracy

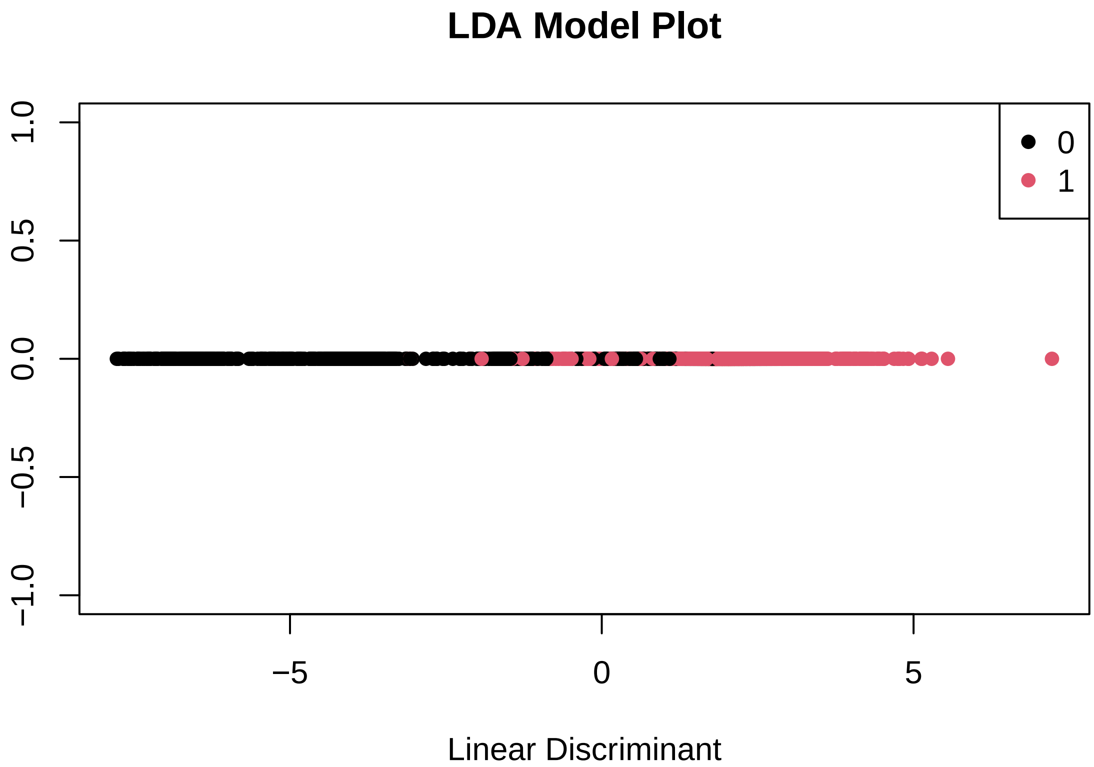
```
mod_lda
```

```
## Call:
```

```
## lda(batter_result ~ ., data = dat_train1, prior = rep(1, 2)/2)
##
## Prior probabilities of groups:
##   0   1
## 0.5 0.5
##
## Group means:
##   thrower_position exchange_time batter_pos_x_at_throw batter_pos_y_at_throw
## 0         5.228031     0.9713026              36.74825              37.93253
## 1         4.978419     1.1905809              32.89720              33.40297
##   batter_velo_at_throw receiver_dist_from_1b throw_deflected_by_receiver
## 0             27.01667              7.666191                0.2280311457
## 1             24.75818              3.292706                0.0005065856
##   runs_on_play thrower_err_throw1 first_base_save1 throw_velo runner_on_third1
## 0   0.11234705         0.47163515       0.00000000   70.96213       0.08120133
## 1   0.01550152         0.04620061       0.04620061   70.34564       0.07588652
##   distance_of_throw
## 0         114.22989
## 1          96.50439
##
## Coefficients of linear discriminants:
##                                      LD1
## thrower_position            -0.003170845
## exchange_time                0.343448596
## batter_pos_x_at_throw       -0.040479914
## batter_pos_y_at_throw       -0.033499128
## batter_velo_at_throw        -0.021273202
## receiver_dist_from_1b       -0.020249905
## throw_deflected_by_receiver -3.112544926
## runs_on_play                -0.918889843
## thrower_err_throw1          -5.027873054
## first_base_save1             5.577849448
## throw_velo                   0.027130156
## runner_on_third1             0.233984265
## distance_of_throw           -0.018695688
```

We can see based on the coefficients of the model that many of the variables actually have a negative coefficient in predicting the result of the batter with a first basement save, velocity of throw, exchange time and having a runner on third having a positive coefficient.

```
plot(pred_lda$x, rep(0, length(pred_lda$x)), col = dat_train1$batter_result, pch = 16, main ="LDA Model
legend("topright", legend = levels(dat_train1$batter_result), col = 1:2, pch = 16)
```



We can see in this plot that there is some overlap between the two results but is overall separated pretty well. This means that the model isn't perfect at predicting the result of a throw to first base which makes sense as baseball is an obscure game that can result in plays like this that should result in an out but don't. Plays that don't result in an out classify more with the negative linear discriminant values and plays that do result in an out correspond more with the positive linear discriminant values.