Kernel and Ensemble Methods: Regression

Isabelle Kirby, Bridgette Bryant

Regression for Korea's top high elo teams in League of Legends

Cleaning Data

First we have to unzip the archive (they are large data sets nearly 100k games). Then we will save the

```r
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6      v purrr   0.3.4
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
win_stats_df <- read.csv((unz("League_Data/league_korea_high_elo_team_stats.zip", "win_team_stats.csv"))
lose_stats_df <- read.csv((unz("League_Data/league_korea_high_elo_team_stats.zip", "lose_team_stats.csv

str(win_stats_df)
```

```
## 'data.frame':    90500 obs. of  31 variables:
##  $ win_kills1                       : num  10 3 7 11 0 7 9 8 18 4 ...
##  $ win_kills2                       : num  4 7 3 4 4 1 10 10 5 2 ...
##  $ win_kills3                       : num  4 0 5 7 2 11 9 5 2 2 ...
##  $ win_kills4                       : num  6 7 9 2 11 10 9 10 2 6 ...
##  $ win_kills5                       : num  7 2 4 3 8 8 2 0 11 4 ...
##  $ win_deaths1                      : num  4 5 5 2 1 3 2 5 2 3 ...
##  $ win_deaths2                      : num  1 0 1 2 4 0 2 5 6 0 ...
##  $ win_deaths3                      : num  4 0 2 2 4 3 3 4 8 1 ...
##  $ win_deaths4                      : num  4 0 1 3 3 6 1 7 4 0 ...
##  $ win_deaths5                      : num  2 3 2 4 4 5 5 6 8 1 ...
##  $ win_totalDamageDealtToChampions1: num  17898 16662 16241 12111 10900 ...
##  $ win_totalDamageDealtToChampions2: num  15800 11674 7572 5510 11362 ...
##  $ win_totalDamageDealtToChampions3: num  10786 7498 10024 8440 14494 ...
##  $ win_totalDamageDealtToChampions4: num  16964 13016 15115 5373 27391 ...
##  $ win_totalDamageDealtToChampions5: num  11568 11393 12395 11038 22399 ...
##  $ win_goldEarned1                  : num  9802 8452 9029 10175 7217 ...
##  $ win_goldEarned2                  : num  9203 9069 6921 5552 10497 ...
##  $ win_goldEarned3                  : num  11127 6023 8331 7439 10323 ...
##  $ win_goldEarned4                  : num  9286 9868 11860 5873 13499 ...
##  $ win_goldEarned5                  : num  10414 7660 8589 7033 12720 ...
##  $ win_visionScore1                 : num  28 14 11 17 42 41 19 23 72 9 ...
##  $ win_visionScore2                 : num  16 27 35 25 38 41 46 55 50 21 ...
```

```
##  $ win_visionScore3                : num  23 46 25 17 30 21 25 47 29 13 ...
##  $ win_visionScore4                : num  17 16 15 9 18 39 31 25 80 19 ...
##  $ win_visionScore5                : num  36 22 21 19 26 19 86 70 22 10 ...
##  $ win_totalTimeCrowdControlDealt1 : num  183 33 178 134 69 310 168 279 365 15 ...
##  $ win_totalTimeCrowdControlDealt2 : num  92 291 82 61 503 45 291 493 119 62 ...
##  $ win_totalTimeCrowdControlDealt3 : num  231 31 371 332 562 133 102 287 456 126 ...
##  $ win_totalTimeCrowdControlDealt4 : num  54 235 140 274 79 78 444 501 215 209 ...
##  $ win_totalTimeCrowdControlDealt5 : num  281 407 122 163 69 73 92 193 300 168 ...
##  $ gameId                          : num  4.25e+09 4.25e+09 4.26e+09 4.26e+09 4.26e+09 ...
```

```
str(lose_stats_df)
```

```
## 'data.frame':    90500 obs. of  31 variables:
##  $ lose_kills1                      : num  3 0 3 1 4 7 0 11 3 0 ...
##  $ lose_kills2                      : num  0 2 5 6 2 1 3 5 10 1 ...
##  $ lose_kills3                      : num  4 3 1 2 4 4 4 3 7 2 ...
##  $ lose_kills4                      : num  4 3 1 1 1 5 2 7 2 2 ...
##  $ lose_kills5                      : num  4 0 1 3 5 0 4 1 6 0 ...
##  $ lose_deaths1                     : num  6 4 7 6 7 7 10 10 7 5 ...
##  $ lose_deaths2                     : num  6 6 4 3 6 9 5 4 6 2 ...
##  $ lose_deaths3                     : num  5 3 7 7 1 11 8 6 10 2 ...
##  $ lose_deaths4                     : num  7 4 5 4 7 4 9 7 10 3 ...
##  $ lose_deaths5                     : num  7 2 5 7 4 6 7 6 5 6 ...
##  $ lose_totalDamageDealtToChampions1: num  10844 4618 7096 9492 9686 ...
##  $ lose_totalDamageDealtToChampions2: num  7095 14837 17030 8557 14045 ...
##  $ lose_totalDamageDealtToChampions3: num  13458 9197 8735 6679 24086 ...
##  $ lose_totalDamageDealtToChampions4: num  9670 10035 7849 4058 2959 ...
##  $ lose_totalDamageDealtToChampions5: num  14972 5531 5815 6912 16719 ...
##  $ lose_goldEarned1                 : num  6844 4524 6551 5442 7928 ...
##  $ lose_goldEarned2                 : num  5205 8823 7562 7846 8042 ...
##  $ lose_goldEarned3                 : num  8226 7788 5346 5092 12502 ...
##  $ lose_goldEarned4                 : num  7911 9008 5004 3931 6185 ...
##  $ lose_goldEarned5                 : num  8815 6993 5931 5071 10729 ...
##  $ lose_visionScore1                : num  14 30 14 1 25 11 17 46 30 13 ...
##  $ lose_visionScore2                : num  34 16 13 27 10 48 30 34 25 7 ...
##  $ lose_visionScore3                : num  8 27 40 9 29 14 38 19 39 13 ...
##  $ lose_visionScore4                : num  21 28 15 26 65 15 81 44 84 24 ...
##  $ lose_visionScore5                : num  14 10 9 5 28 13 13 68 45 8 ...
##  $ lose_totalTimeCrowdControlDealt1 : num  19 80 133 71 422 188 97 71 246 20 ...
##  $ lose_totalTimeCrowdControlDealt2 : num  38 67 249 476 151 77 36 327 98 102 ...
##  $ lose_totalTimeCrowdControlDealt3 : num  173 491 135 13 161 109 347 433 36 169 ...
##  $ lose_totalTimeCrowdControlDealt4 : num  305 50 125 52 63 204 208 44 95 47 ...
##  $ lose_totalTimeCrowdControlDealt5 : num  237 0 226 88 97 101 81 242 447 182 ...
##  $ gameId                           : num  4.25e+09 4.25e+09 4.26e+09 4.26e+09 4.26e+09 ...
```

Now we will merge the two data sets together based on their matching gameId column. This way our model can catagorize our data by team 0 or team 1 winning based on both teams contrasting stats.

```
# Replacing column names for rbind
colnames(win_stats_df) <- c('kill1', 'kill2', 'kill3', 'kill4', 'kill5', 'death1', 'death2', 'death3',
colnames(lose_stats_df) <- c('kill1', 'kill2', 'kill3', 'kill4', 'kill5', 'death1', 'death2', 'death3',

# Adding column based on dataset it is in
library(dplyr)

win_stats_df <- win_stats_df %>%
```

```r
  mutate(won=1)

lose_stats_df <- lose_stats_df %>%
  mutate(won =0)

#full_stats_df <- merge(win_stats_df, lose_stats_df, by = "gameId")
full_stats_df <- rbind(win_stats_df, lose_stats_df)
drop <- c("gameId")
full_stats_df <- full_stats_df[,!(names(full_stats_df) %in% drop)]
str(full_stats_df)
```

```
## 'data.frame':    181000 obs. of  31 variables:
##  $ kill1                        : num  10 3 7 11 0 7 9 8 18 4 ...
##  $ kill2                        : num  4 7 3 4 4 1 10 10 5 2 ...
##  $ kill3                        : num  4 0 5 7 2 11 9 5 2 2 ...
##  $ kill4                        : num  6 7 9 2 11 10 9 10 2 6 ...
##  $ kill5                        : num  7 2 4 3 8 8 2 0 11 4 ...
##  $ death1                       : num  4 5 5 2 1 3 2 5 2 3 ...
##  $ death2                       : num  1 0 1 2 4 0 2 5 6 0 ...
##  $ death3                       : num  4 0 2 2 4 3 3 4 8 1 ...
##  $ death4                       : num  4 0 1 3 3 6 1 7 4 0 ...
##  $ death5                       : num  2 3 2 4 4 5 5 6 8 1 ...
##  $ totalDamageDealtToChampions1: num  17898 16662 16241 12111 10900 ...
##  $ totalDamageDealtToChampions2: num  15800 11674 7572 5510 11362 ...
##  $ totalDamageDealtToChampions3: num  10786 7498 10024 8440 14494 ...
##  $ totalDamageDealtToChampions4: num  16964 13016 15115 5373 27391 ...
##  $ totalDamageDealtToChampions5: num  11568 11393 12395 11038 22399 ...
##  $ goldEarned1                  : num  9802 8452 9029 10175 7217 ...
##  $ goldEarned2                  : num  9203 9069 6921 5552 10497 ...
##  $ goldEarned3                  : num  11127 6023 8331 7439 10323 ...
##  $ goldEarned4                  : num  9286 9868 11860 5873 13499 ...
##  $ goldEarned5                  : num  10414 7660 8589 7033 12720 ...
##  $ visionScore1                 : num  28 14 11 17 42 41 19 23 72 9 ...
##  $ visionScore2                 : num  16 27 35 25 38 41 46 55 50 21 ...
##  $ visionScore3                 : num  23 46 25 17 30 21 25 47 29 13 ...
##  $ visionScore4                 : num  17 16 15 9 18 39 31 25 80 19 ...
##  $ visionScore5                 : num  36 22 21 19 26 19 86 70 22 10 ...
##  $ totalTimeCrowdControlDealt1 : num  183 33 178 134 69 310 168 279 365 15 ...
##  $ totalTimeCrowdControlDealt2 : num  92 291 82 61 503 45 291 493 119 62 ...
##  $ totalTimeCrowdControlDealt3 : num  231 31 371 332 562 133 102 287 456 126 ...
##  $ totalTimeCrowdControlDealt4 : num  54 235 140 274 79 78 444 501 215 209 ...
##  $ totalTimeCrowdControlDealt5 : num  281 407 122 163 69 73 92 193 300 168 ...
##  $ won                          : num  1 1 1 1 1 1 1 1 1 1 ...
```

```r
i <- sample(1:nrow(full_stats_df), .1*nrow(full_stats_df), replace=FALSE)
full_stats_smol <- full_stats_df[i,]

lolDataless <- full_stats_smol %>% rowwise() %>% mutate(TotalKill = sum(c_across(kill1:kill5)))

lolDataless <- lolDataless %>% rowwise() %>% mutate(TotalDeath = sum(c_across(death1:death5)))

lolDataless <- lolDataless %>% rowwise() %>% mutate(TotalDamage = sum(c_across(totalDamageDealtToChampi

lolDataless <- lolDataless %>% rowwise() %>% mutate(TotalGold = sum(c_across(goldEarned1:goldEarned5)))
```

```r
lolDataless <- lolDataless %>% rowwise() %>% mutate(TotalVision = sum(c_across(visionScore1:visionScore5

lolDataless <- lolDataless %>% rowwise() %>% mutate(TotalCrowdControl = sum(c_across(totalTimeCrowdCont

drop <- c('kill1', 'kill2', 'kill3', 'kill4', 'kill5', 'death1', 'death2', 'death3', 'death4', 'death5'

lolDataless = lolDataless[, !(names(lolDataless) %in% drop)]
summary(lolDataless)
```

```
##       won           TotalKill       TotalDeath     TotalDamage
##  Min.   :0.0000   Min.   : 0.00   Min.   : 0.00   Min.   :     0
##  1st Qu.:0.0000   1st Qu.:14.00   1st Qu.:14.00   1st Qu.: 39638
##  Median :1.0000   Median :22.00   Median :22.00   Median : 61184
##  Mean   :0.5022   Mean   :22.52   Mean   :22.59   Mean   : 65133
##  3rd Qu.:1.0000   3rd Qu.:30.00   3rd Qu.:30.00   3rd Qu.: 86398
##  Max.   :1.0000   Max.   :71.00   Max.   :85.00   Max.   :296232
##    TotalGold       TotalVision    TotalCrowdControl
##  Min.   :  3458   Min.   :  0.0   Min.   :   0.0
##  1st Qu.: 35423   1st Qu.: 83.0   1st Qu.: 571.0
##  Median : 46551   Median :127.0   Median : 797.0
##  Mean   : 46355   Mean   :130.2   Mean   : 852.4
##  3rd Qu.: 57587   3rd Qu.:172.0   3rd Qu.:1071.0
##  Max.   :101407   Max.   :447.0   Max.   :3339.0
```

Next let's randomly divide the data into train, test, and validate:

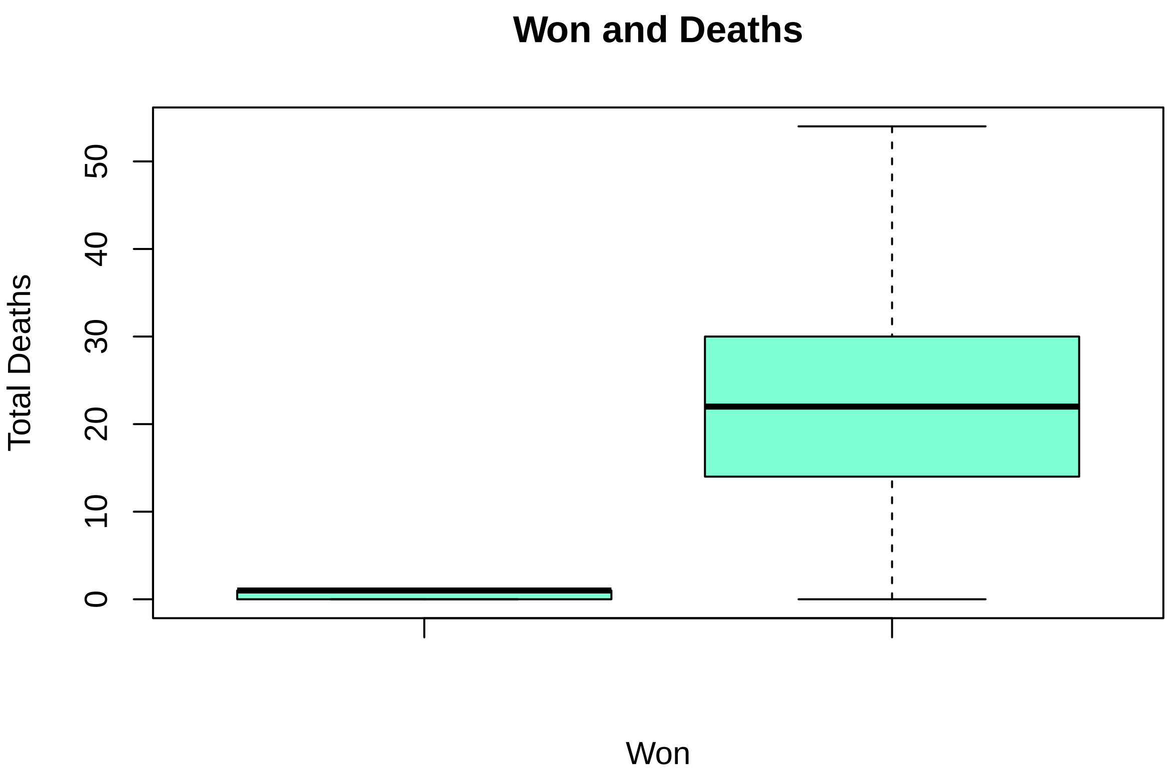
```r
set.seed(1010)
#j <- sample(1:nrow(full_stats_smol), 0.75*nrow(full_stats_smol), replace=FALSE)
#full_stats_df <- full_stats_df[j,]
spec <-c(train=.6, test=.2, validate=.2)
i <- sample(cut(1:nrow(lolDataless), nrow(lolDataless)*cumsum(c(0,spec)), labels=names(spec)))
full_stats_train <- lolDataless[i=="train",]
full_stats_test <- lolDataless[i=="test",]
full_stats_validate <- lolDataless[i=="validate",]
summary(full_stats_train)
```

```
##       won          TotalKill       TotalDeath     TotalDamage
##  Min.   :0.000   Min.   : 0.00   Min.   : 0.00   Min.   :     0
##  1st Qu.:0.000   1st Qu.:14.00   1st Qu.:14.00   1st Qu.: 39777
##  Median :1.000   Median :22.00   Median :22.00   Median : 61406
##  Mean   :0.506   Mean   :22.61   Mean   :22.59   Mean   : 65341
##  3rd Qu.:1.000   3rd Qu.:30.00   3rd Qu.:30.00   3rd Qu.: 86605
##  Max.   :1.000   Max.   :71.00   Max.   :80.00   Max.   :296232
##    TotalGold       TotalVision    TotalCrowdControl
##  Min.   :  3728   Min.   :  0.0   Min.   :   0.0
##  1st Qu.: 35474   1st Qu.: 82.0   1st Qu.: 572.0
##  Median : 46579   Median :127.0   Median : 798.0
##  Mean   : 46354   Mean   :129.8   Mean   : 853.8
##  3rd Qu.: 57499   3rd Qu.:171.0   3rd Qu.:1075.0
##  Max.   :101407   Max.   :447.0   Max.   :3324.0
```

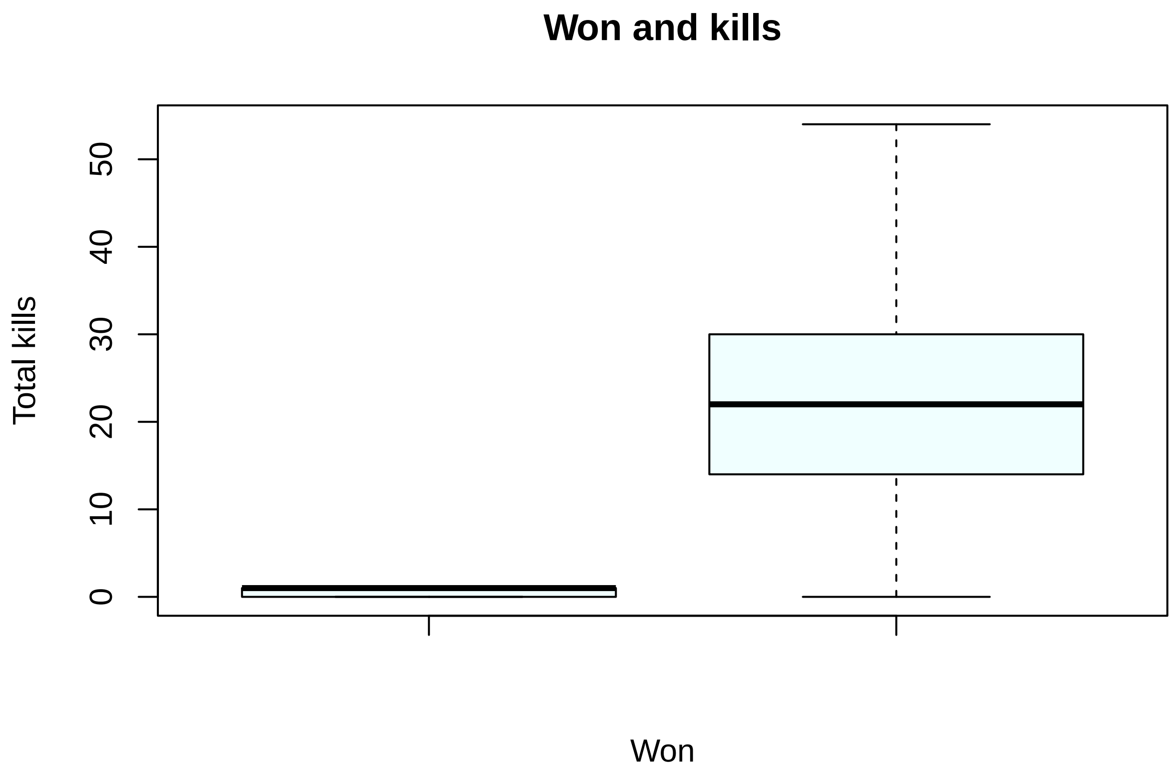
Data Exploration

Next we will plot some of our data to see possible differences/correlations. Time to do some data exploration.

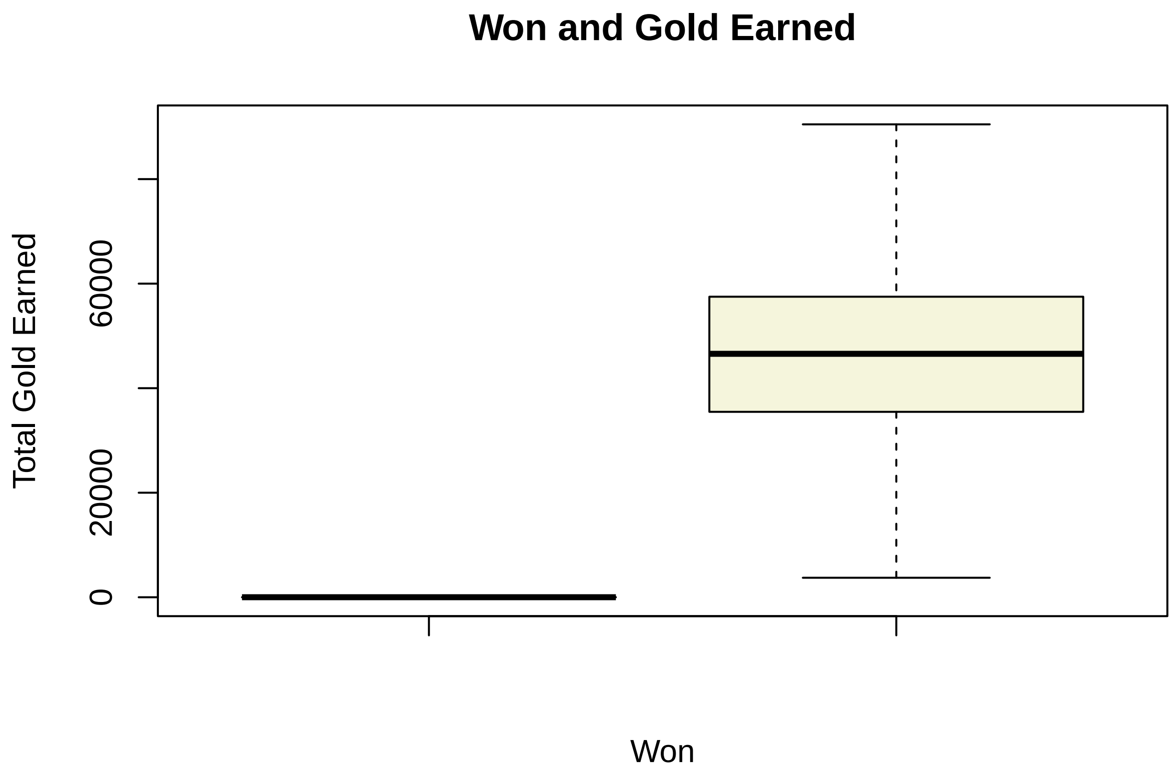
```
boxplot(full_stats_train$won, full_stats_train$TotalDeath, main="Won and Deaths", xlab="Won", ylab="Tota
```



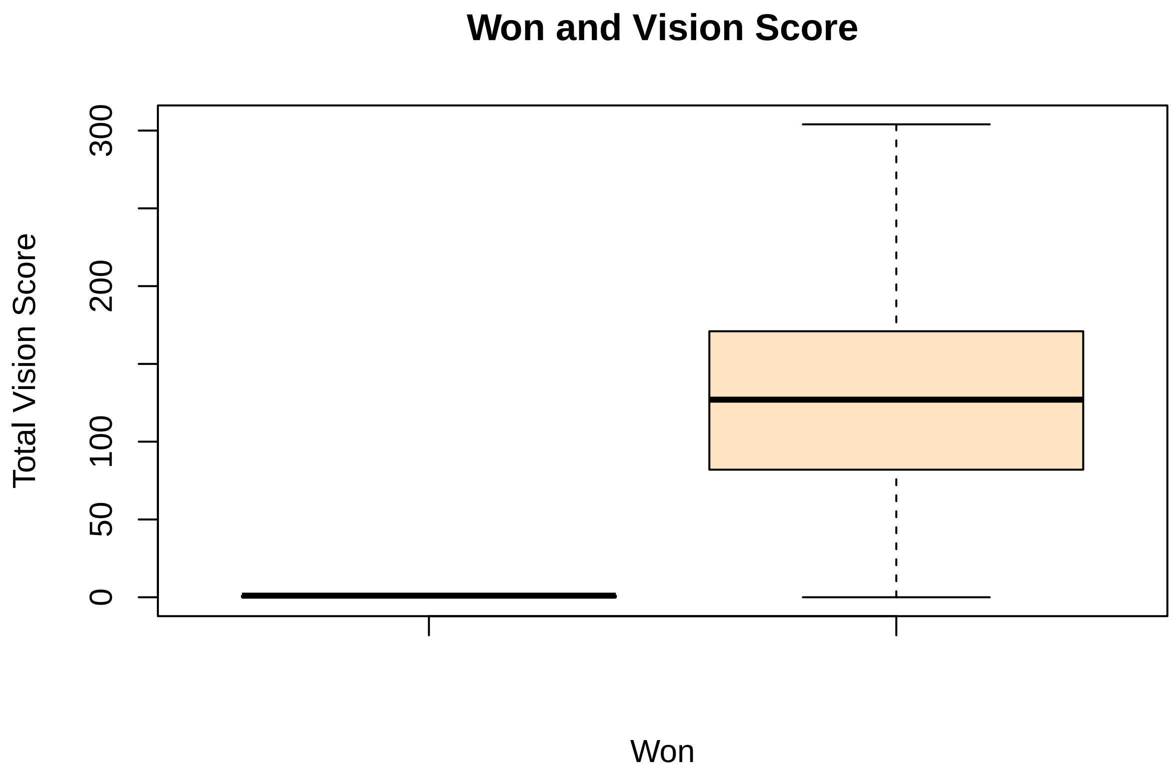

```
boxplot(full_stats_train$won, full_stats_train$TotalKill, main="Won and kills", xlab="Won", ylab="Total
```



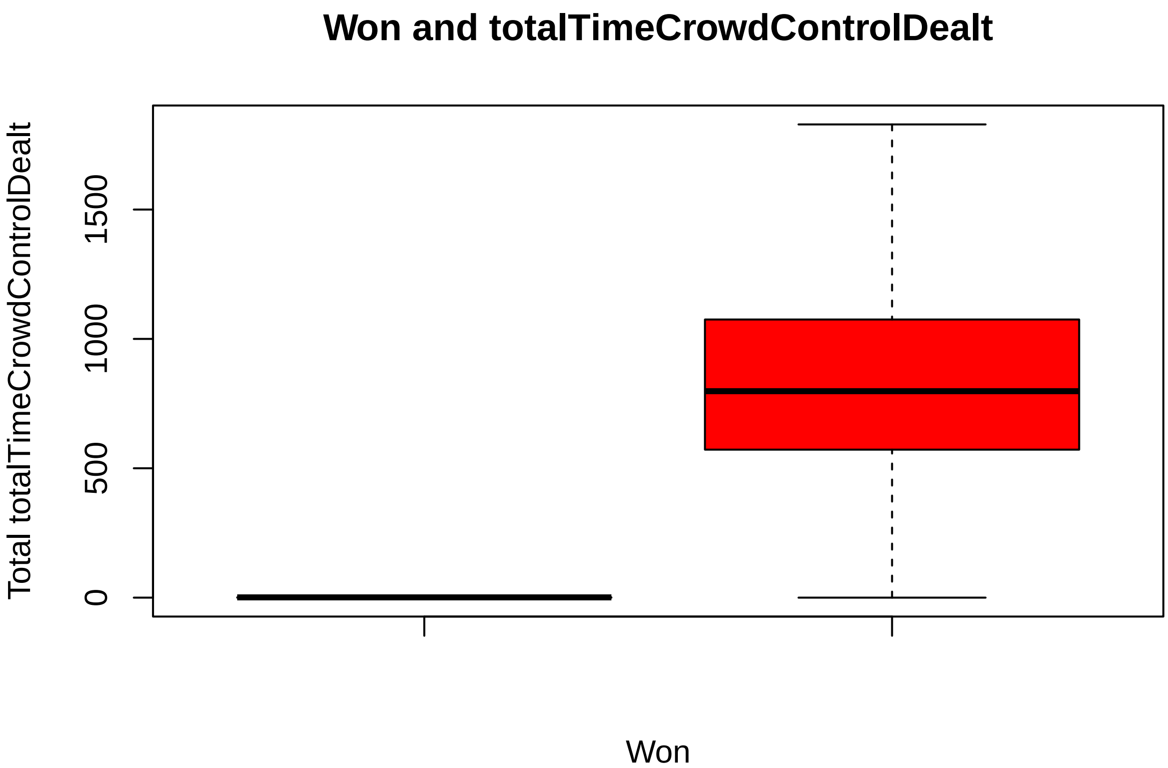

```
boxplot(full_stats_train$won, full_stats_train$TotalGold, main="Won and Gold Earned", xlab="Won", ylab=
```




```
boxplot(full_stats_train$won, full_stats_train$TotalVision, main="Won and Vision Score", xlab="Won", yla
```

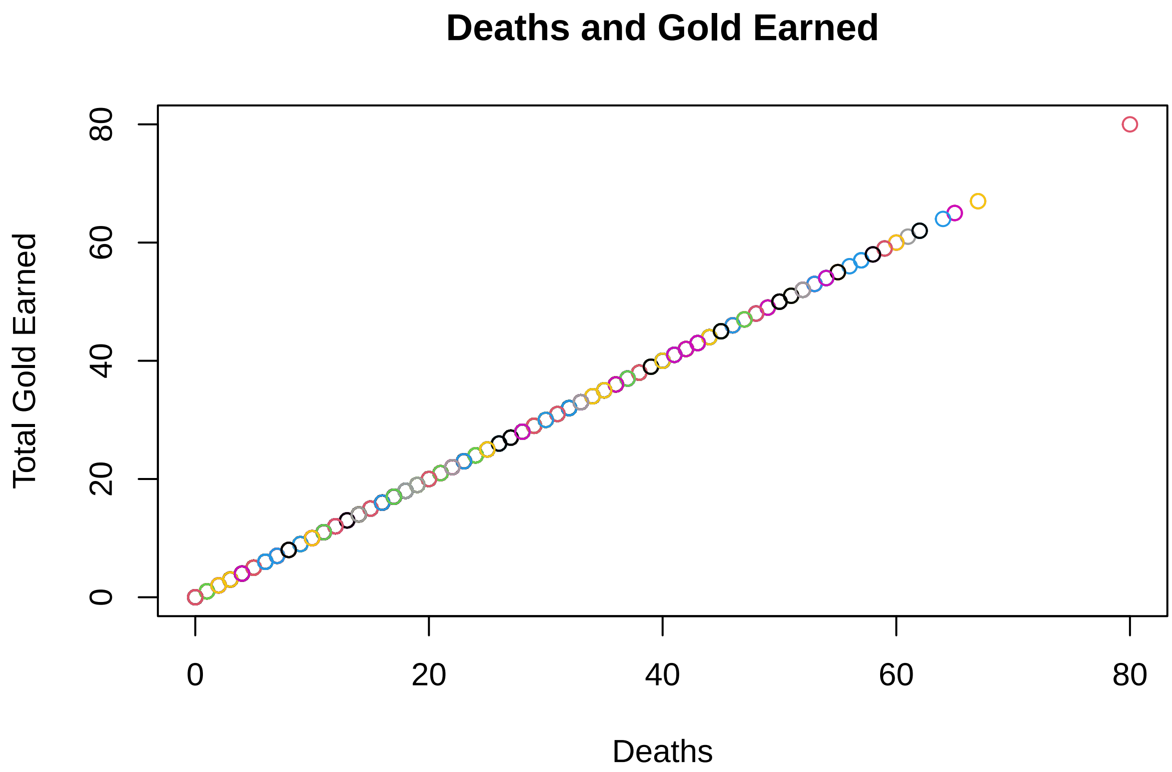



```
boxplot(full_stats_train$won, full_stats_train$TotalCrowdControl, main="Won and totalTimeCrowdControlDea
```

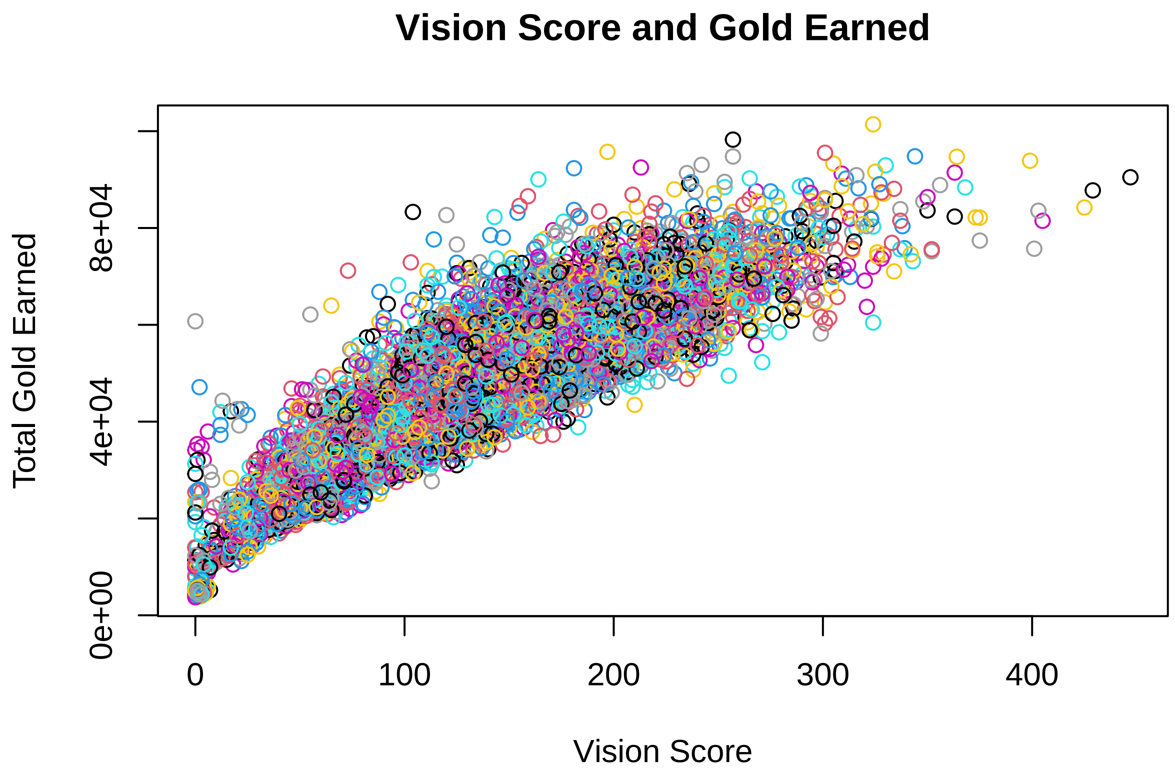



```
plot(full_stats_train$TotalKill, full_stats_train$TotalGold, main="Kills and Gold Earned", xlab="Kills"
```

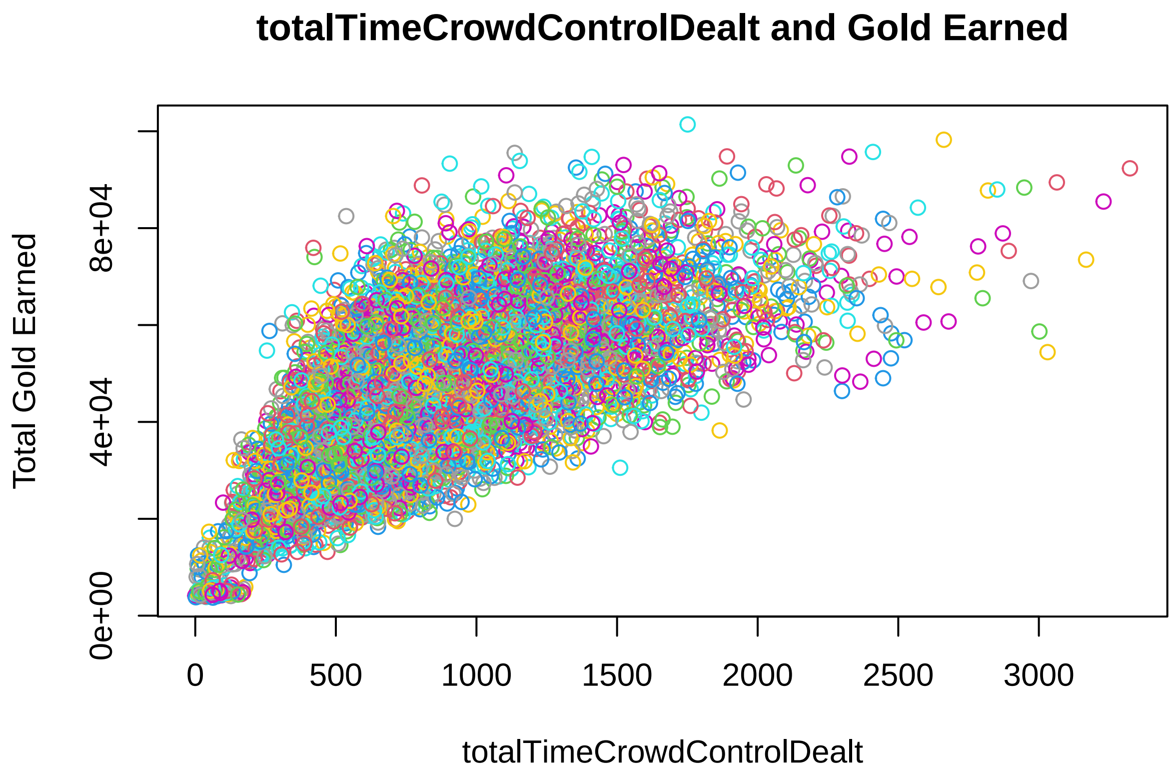

```
plot(full_stats_train$TotalDeath, full_stats_train$TotalDeath, main="Deaths and Gold Earned", xlab="Dea
```




```
plot(full_stats_train$TotalVision, full_stats_train$TotalGold, main="Vision Score and Gold Earned", xlal
```




```
plot(full_stats_train$TotalCrowdControl, full_stats_train$TotalGold, main="totalTimeCrowdControlDealt an
```




```
mean(full_stats_train$TotalKill)
```

```
## [1] 22.60792
```

```
mean(full_stats_test$TotalKill)
```

```
## [1] 22.53315
```

```
mean(full_stats_validate$TotalKill)
```

```
## [1] 22.24586
```

SVM Regression

Linear Regression

Trying linear regression with the data set

```
linreg <- lm(won~., data=full_stats_train)
predLin <- predict(linreg, newdata=full_stats_test)
cor_lin <- cor(predLin, full_stats_test$won)
mseLin <- mean((predLin-full_stats_test$won)^2)
summary(linreg)
```

```
##
## Call:
## lm(formula = won ~ ., data = full_stats_train)
##
## Residuals:
```

```
##     Min      1Q  Median      3Q     Max
## -1.1459 -0.1631  0.0018  0.1613  1.5548
##
## Coefficients:
##                     Estimate Std. Error  t value Pr(>|t|)
## (Intercept)        3.233e-01  1.143e-02   28.280  < 2e-16 ***
## TotalKill          2.137e-02  5.201e-04   41.087  < 2e-16 ***
## TotalDeath        -3.383e-02  2.952e-04 -114.619  < 2e-16 ***
## TotalDamage       -3.044e-07  2.371e-07   -1.284    0.199
## TotalGold          1.632e-05  6.924e-07   23.573  < 2e-16 ***
## TotalVision       -1.730e-03  1.022e-04  -16.927  < 2e-16 ***
## TotalCrowdControl -5.638e-05  8.417e-06   -6.698 2.22e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2538 on 10853 degrees of freedom
## Multiple R-squared:  0.7425, Adjusted R-squared:  0.7423
## F-statistic:  5215 on 6 and 10853 DF,  p-value: < 2.2e-16
```

Training Now we will build our SVM Linear model

```r
library(e1071)
svm_won <- svm(won~., data=full_stats_train, kernel="linear", cost=10, scale=TRUE)
summary(svm_won)
```

```
##
## Call:
## svm(formula = won ~ ., data = full_stats_train, kernel = "linear",
##     cost = 10, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  linear
##        cost:  10
##       gamma:  0.1666667
##     epsilon:  0.1
##
##
## Number of Support Vectors:  9023
```

Testing & Evaluation Now we can evaluate on the test set:

```r
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```r
svm_probs <- predict(svm_won, newdata = full_stats_test)
svm_pred <- ifelse(svm_probs > 0.5, 1, 0)
svm_acc <- mean(svm_pred == full_stats_test$won)
```

```r
confusionMatrix(as.factor(svm_pred), reference = as.factor(full_stats_test$won))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1726   75
##          1   99 1720
##
##                Accuracy : 0.9519
##                  95% CI : (0.9445, 0.9587)
##     No Information Rate : 0.5041
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.9039
##
##  Mcnemar's Test P-Value : 0.08122
##
##             Sensitivity : 0.9458
##             Specificity : 0.9582
##          Pos Pred Value : 0.9584
##          Neg Pred Value : 0.9456
##              Prevalence : 0.5041
##          Detection Rate : 0.4768
##    Detection Prevalence : 0.4975
##       Balanced Accuracy : 0.9520
##
##        'Positive' Class : 0
##
```

```r
predLinSvm <- predict(svm_won, newdata=full_stats_test)
corLinSvm <- cor(predLinSvm, full_stats_test$won)
mseLinSvm <- mean((predLinSvm - full_stats_test$won)^2)

tuneLin <- tune(svm, won~., data=full_stats_validate, kernel="linear", ranges=list(cost=c(0.001, 0.01, 
summary(tuneLin)
```

Tuning

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.01
##
## - best performance: 0.06787431
##
## - Detailed performance results:
##    cost      error  dispersion
## 1 1e-03 0.07189388 0.008834838
```

```
## 2 1e-02 0.06787431 0.010554193
## 3 1e-01 0.06839286 0.011019576
## 4 1e+00 0.06848676 0.011057311
## 5 5e+00 0.06849656 0.011058492
## 6 1e+01 0.06849910 0.011062166
## 7 1e+02 0.06849931 0.011053950
```

```r
svm_poly <- svm(won~., data=full_stats_train, kernel="polynomial", cost=10, scale=TRUE)
summary(svm_poly)
```

Polynomial Kernel

```
##
## Call:
## svm(formula = won ~ ., data = full_stats_train, kernel = "polynomial",
##     cost = 10, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  3
##       gamma:  0.1666667
##      coef.0:  0
##     epsilon:  0.1
##
##
## Number of Support Vectors:  10085
```

Testing & Evaluation Now we can evaluate on the test set:

```r
predPolySvm <- predict(svm_poly, newdata=full_stats_test)
corPolySvm <- cor(predPolySvm, full_stats_test$won)
svm_poly <- ifelse(predPolySvm > 0.5, 1, 0)
msePolySvm <- mean((predPolySvm - full_stats_test$won)^2)

confusionMatrix(as.factor(svm_poly), reference = as.factor(full_stats_test$won))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1650   62
##          1  175 1733
##
##                Accuracy : 0.9345
##                  95% CI : (0.926, 0.9424)
##     No Information Rate : 0.5041
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8691
##
##  Mcnemar's Test P-Value : 3.46e-13
```

```
##
##             Sensitivity : 0.9041
##             Specificity : 0.9655
##          Pos Pred Value : 0.9638
##          Neg Pred Value : 0.9083
##              Prevalence : 0.5041
##          Detection Rate : 0.4558
##    Detection Prevalence : 0.4729
##       Balanced Accuracy : 0.9348
##
##        'Positive' Class : 0
##
```

```
svm_rad <- svm(won~., data=full_stats_train, kernel="radial", cost=10, scale=TRUE)
summary(svm_rad)
```

Radial Kernel

```
##
## Call:
## svm(formula = won ~ ., data = full_stats_train, kernel = "radial",
##     cost = 10, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  10
##       gamma:  0.1666667
##     epsilon:  0.1
##
##
## Number of Support Vectors:  4918
```

Testing & Evaluation Now we can evaluate on the test set:

```
predRadSvm <- predict(svm_rad, newdata=full_stats_test)
corRadSvm <- cor(predRadSvm, full_stats_test$won)
mseRadSvm <- mean((predRadSvm - full_stats_test$won)^2)
svm_rad <- ifelse(predRadSvm > 0.5, 1, 0)

confusionMatrix(as.factor(svm_poly), reference = as.factor(full_stats_test$won))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1650   62
##          1  175 1733
##
##                Accuracy : 0.9345
##                  95% CI : (0.926, 0.9424)
##     No Information Rate : 0.5041
##     P-Value [Acc > NIR] : < 2.2e-16
```

```
##
##                   Kappa : 0.8691
##
##  Mcnemar's Test P-Value : 3.46e-13
##
##             Sensitivity : 0.9041
##             Specificity : 0.9655
##          Pos Pred Value : 0.9638
##          Neg Pred Value : 0.9083
##              Prevalence : 0.5041
##          Detection Rate : 0.4558
##    Detection Prevalence : 0.4729
##       Balanced Accuracy : 0.9348
##
##        'Positive' Class : 0
##
```

```
set.seed(1234)

tuneRad <- tune(svm, won~., data=full_stats_validate, kernel="radial", ranges=list(cost=c(0.1,1,10,100,1
summary(tuneRad)
```

Tune Hyperparameters

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1   0.5
##
## - best performance: 0.04113543
##
## - Detailed performance results:
##     cost gamma      error  dispersion
## 1  1e-01   0.5 0.04319639 0.003989288
## 2  1e+00   0.5 0.04113543 0.005748297
## 3  1e+01   0.5 0.04313502 0.008724538
## 4  1e+02   0.5 0.06033883 0.011026754
## 5  1e+03   0.5 0.12605387 0.044667895
## 6  1e-01   1.0 0.04538058 0.004680495
## 7  1e+00   1.0 0.04232746 0.007830379
## 8  1e+01   1.0 0.04903680 0.010549905
## 9  1e+02   1.0 0.07435409 0.011027605
## 10 1e+03   1.0 0.15559003 0.027085261
## 11 1e-01   2.0 0.05526589 0.005178939
## 12 1e+00   2.0 0.04642575 0.008861734
## 13 1e+01   2.0 0.05686034 0.009832372
## 14 1e+02   2.0 0.07318121 0.011179788
## 15 1e+03   2.0 0.08245586 0.011488873
## 16 1e-01   3.0 0.06918593 0.005243135
## 17 1e+00   3.0 0.05219230 0.008461446
```

```
## 18 1e+01   3.0 0.06128007 0.008603292
## 19 1e+02   3.0 0.06426263 0.008620592
## 20 1e+03   3.0 0.07402782 0.013371524
## 21 1e-01   4.0 0.08501295 0.005129542
## 22 1e+00   4.0 0.05875865 0.007525090
## 23 1e+01   4.0 0.06369005 0.007620441
## 24 1e+02   4.0 0.06532149 0.006817052
## 25 1e+03   4.0 0.07844597 0.027057094
```

```r
svm_radTune <- svm(won~., data=full_stats_validate, kernel="radial", cost=100, gamma=0.5, scale=TRUE)
summary(svm_radTune)
```

```
##
## Call:
## svm(formula = won ~ ., data = full_stats_validate, kernel = "radial",
##     cost = 100, gamma = 0.5, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  100
##       gamma:  0.5
##     epsilon:  0.1
##
##
## Number of Support Vectors:  1688
```

```r
predRadSvm1 <- predict(svm_radTune, newdata=full_stats_test)
corRadSvm1 <- cor(predRadSvm1, full_stats_test$won)
mseRadSvm1 <- mean((predRadSvm1 - full_stats_test$won)^2)
```

SVM Linear vs Polynomial vs Radial Kernels

Analyzing the results of each model based on the algorithms.

SVM Linear

For this we are trying to predict whether or not a team has won a match using the stats they had at the end of a game. Linear SVM works by plotting the data in a high-dimensional feature space so that the points can be categorized. The data is then transformed in ways that allows a separator to be drawn as a hyper plane. Linear kernels are better for when the data can be linearly separated easily.

For this data set there was little difference between the accuracies in the different kernel types, which may mean this data was easy to split regardless of the kernel type. The linear kernel did perform marginally better than the others.

SVM Polynomial

Polynomial kernels behave similarly to the linear kernels. However, they put all the data in a feature space over polynomials of the original variables, meaning that they work better for data that isn't easily separated linearly.

This performed slightly worse than linear kernel, meaning this dataset was easily split using a linear hyperplane. However, it was still very accurate.

SVM Radial Kernels

Radial kernels are similar to polynomial kernels as they both work with data that cannot be linearly separated easily. This generates a non-linear decision boundary.

This performed slightly worse than linear kernel, meaning this dataset was easily split using a linear hyperplane. However, it was still very accurate.