Kernel and Ensemble Methods: Classification

Isabelle Kirby, Bridgette Bryant

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

I utilized SVM linear, polynomial, and radial kernels to classify the League of Legends team won or lost based on the total team stats for kills, deaths, gold, and vision scores. Our dataset has all these values in two datasets lose_team_stats.csv and win_team_stats.csv. Each game also has a gameId which is unique to that specific game. This ID is identical in each dataset. The win_team_stats.csv file contains the winning teams kills, deaths, gold earned, damage dealt, crowd control, and vision scores for each player. The lose_team_stats is the same but for the losing teams. These datasets are great because they contain nearly 100k games and have no NA/Null values.

Cleaning Data

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

```
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)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
win_stats_df <- win_stats_df %>%
  mutate(won="True")

lose_stats_df <- lose_stats_df %>%
  mutate(won ="False")

#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)]

# Make our won column a factor for classification
full_stats_df$won <- as.factor(full_stats_df$won)

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                          : Factor w/ 2 levels "False","True": 2 2 2 2 2 2 2 2 2 2 ...
```

```
i <- sample(1:nrow(full_stats_df), .2*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)))

lolDataless <- lolDataless %>% rowwise() %>% mutate(TotalVision = sum(c_across(visionScore1:visionScore

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
##  False:18058   Min.   : 0.00   Min.   : 0.00   Min.   :     0
##  True :18142   1st Qu.:14.00   1st Qu.:14.00   1st Qu.: 39532
##                Median :22.00   Median :22.00   Median : 61428
##                Mean   :22.59   Mean   :22.64   Mean   : 65046
##                3rd Qu.:30.00   3rd Qu.:30.00   3rd Qu.: 85649
##                Max.   :77.00   Max.   :85.00   Max.   :257629
##    TotalGold       TotalVision    TotalCrowdControl
##  Min.   :  3356   Min.   :  0.0   Min.   :   0.0
##  1st Qu.: 35412   1st Qu.: 82.0   1st Qu.: 575.0
##  Median : 46600   Median :126.0   Median : 797.0
##  Mean   : 46451   Mean   :130.1   Mean   : 852.1
##  3rd Qu.: 57435   3rd Qu.:172.0   3rd Qu.:1070.0
##  Max.   :108542   Max.   :483.0   Max.   :3781.0
```

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

```
set.seed(1010)
i <- sample(1:nrow(lolDataless), 0.75*nrow(lolDataless), replace=FALSE)
full_stats_train <- lolDataless[i,]
full_stats_test <- lolDataless[-i,]

summary(full_stats_train)
```

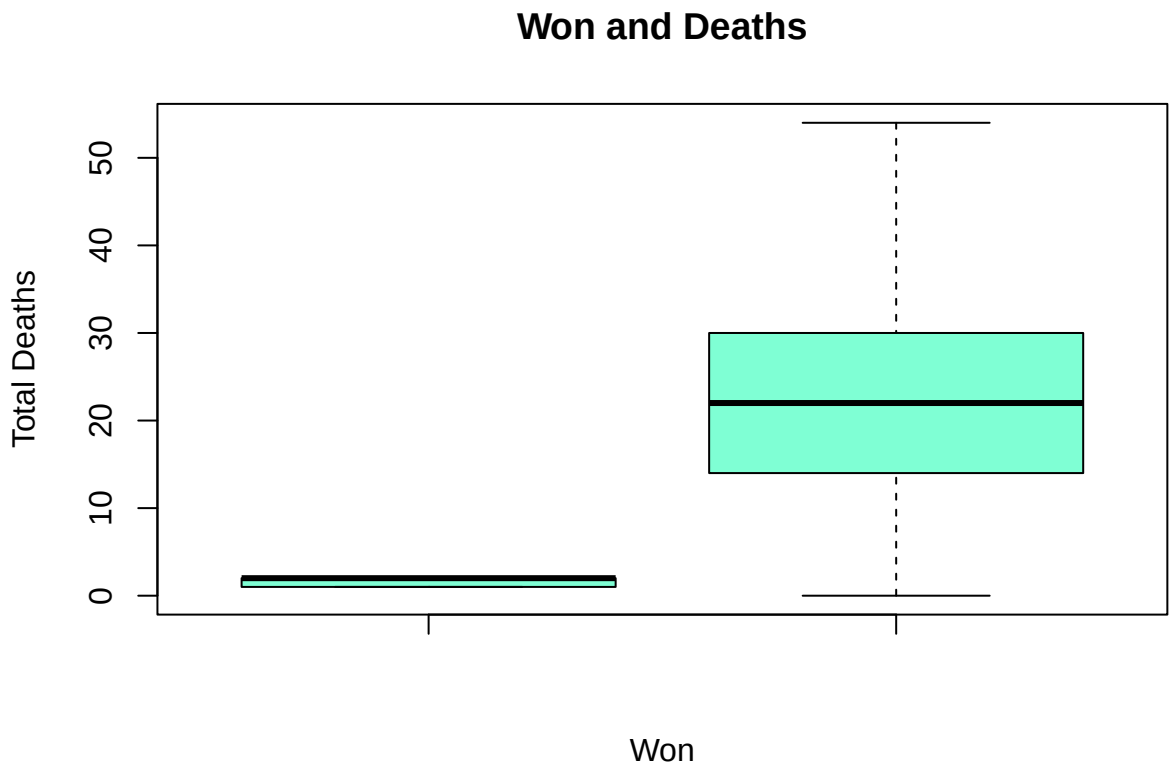
```
##     won          TotalKill       TotalDeath     TotalDamage
##  False:13526   Min.   : 0.00   Min.   : 0.00   Min.   :     0
##  True :13624   1st Qu.:14.00   1st Qu.:14.00   1st Qu.: 39702
##                Median :22.00   Median :22.00   Median : 61314
##                Mean   :22.62   Mean   :22.65   Mean   : 65046
##                3rd Qu.:30.00   3rd Qu.:30.00   3rd Qu.: 85644
##                Max.   :77.00   Max.   :85.00   Max.   :257629
##    TotalGold       TotalVision    TotalCrowdControl
##  Min.   :  3356   Min.   :  0.0   Min.   :   0.0
##  1st Qu.: 35445   1st Qu.: 82.0   1st Qu.: 576.0
```

```
##  Median : 46633   Median :126.0   Median : 797.0
##  Mean   : 46460   Mean   :130.2   Mean   : 852.4
##  3rd Qu.: 57468   3rd Qu.:172.0   3rd Qu.:1068.8
##  Max.   :108542   Max.   :483.0   Max.   :3781.0
```

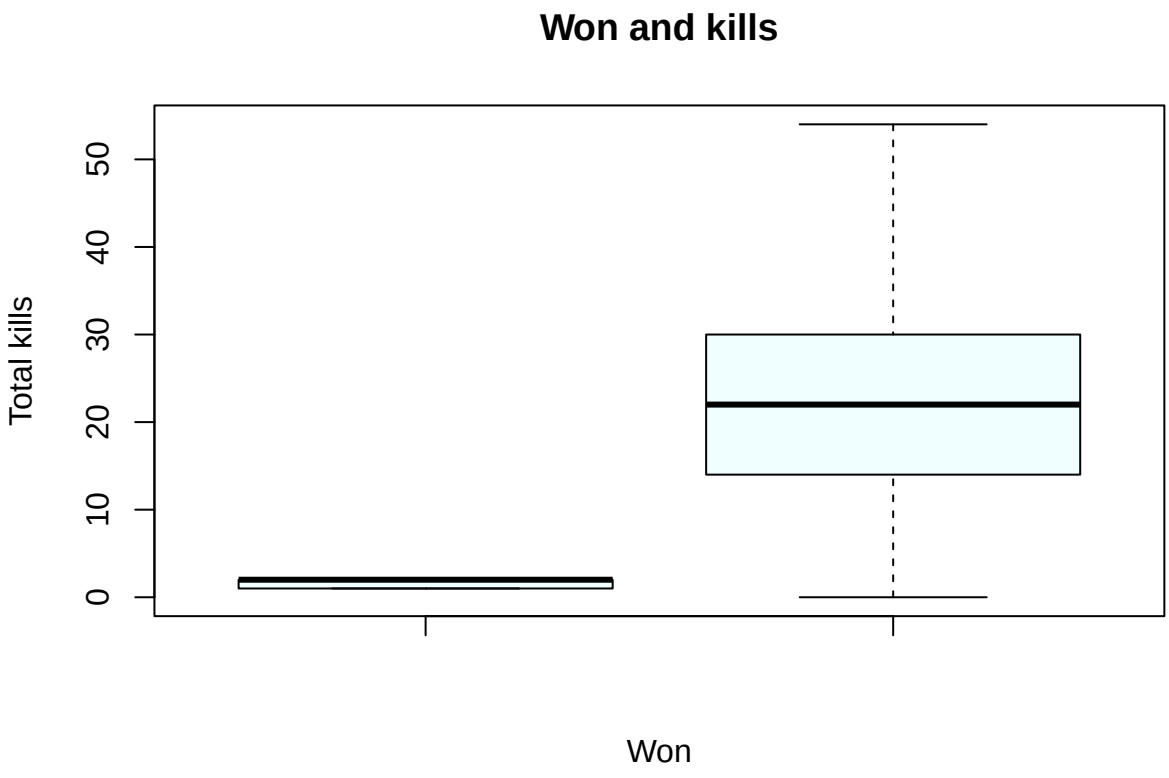
Data Exploration

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

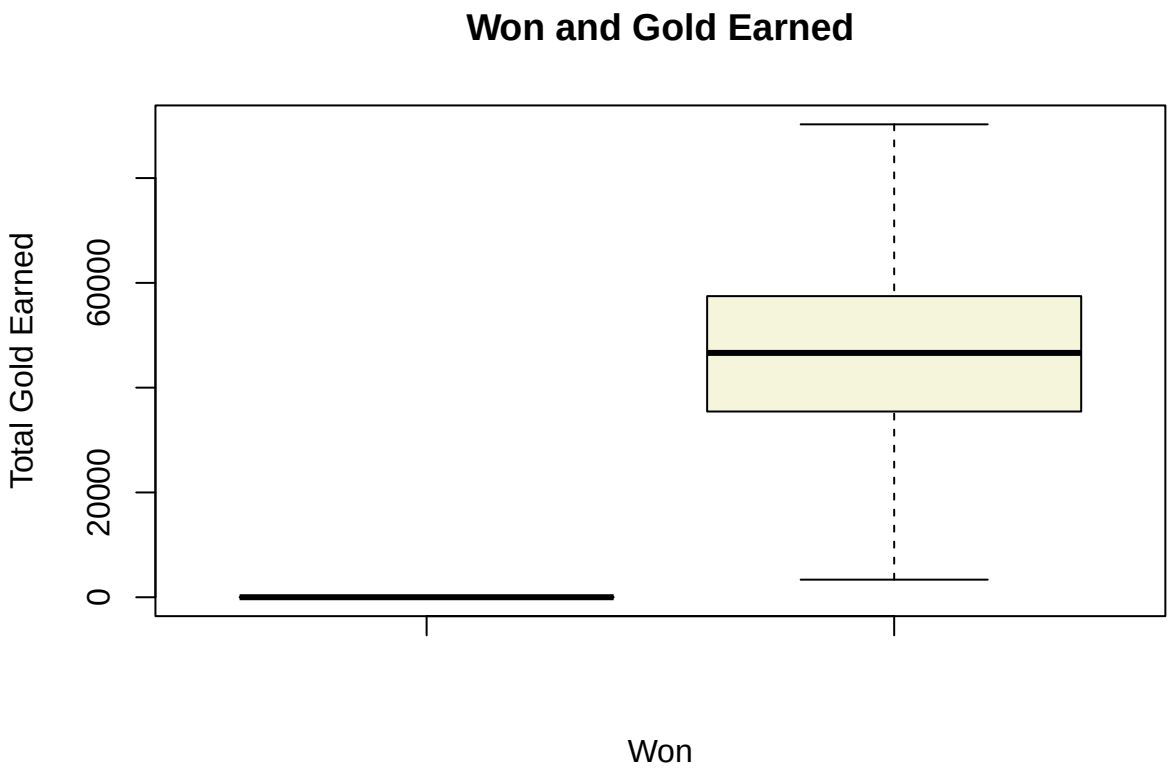
```
boxplot(full_stats_train$won, full_stats_train$TotalDeath, main="Won and Deaths", xlab="Won", ylab="Tota
```



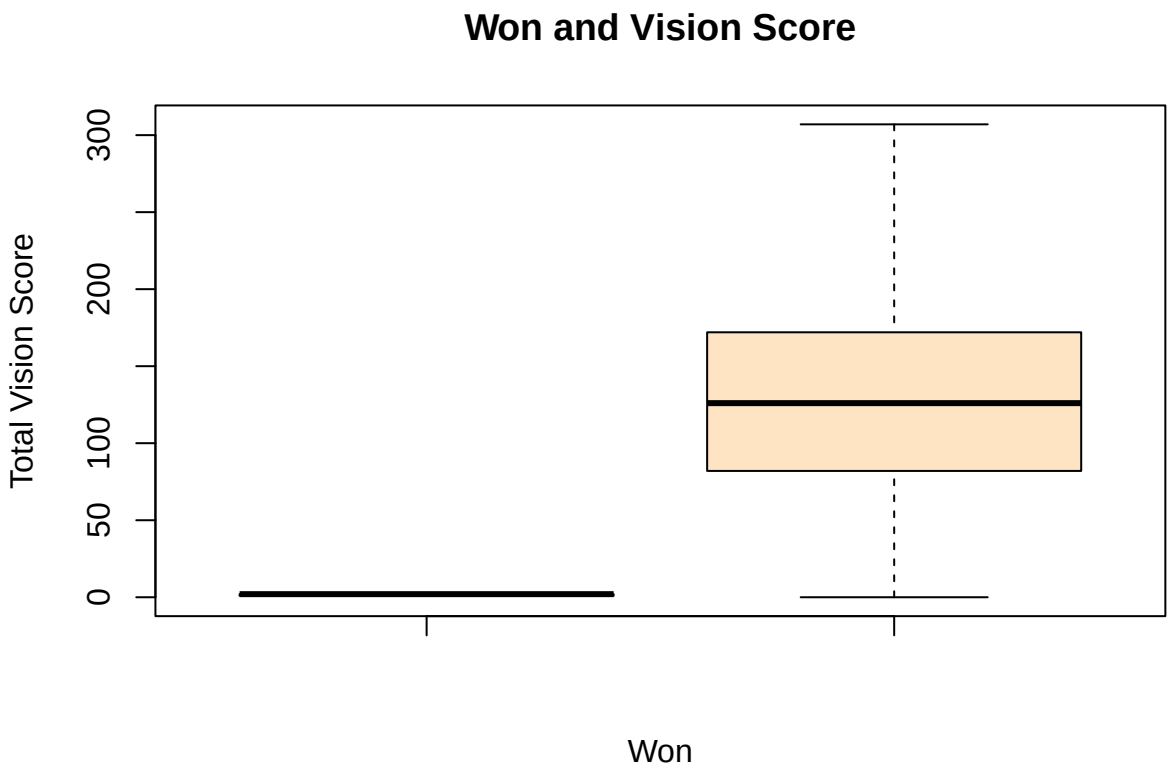

```
boxplot(full_stats_train$won, full_stats_train$TotalKill, main="Won and kills", xlab="Won", ylab="Total
```



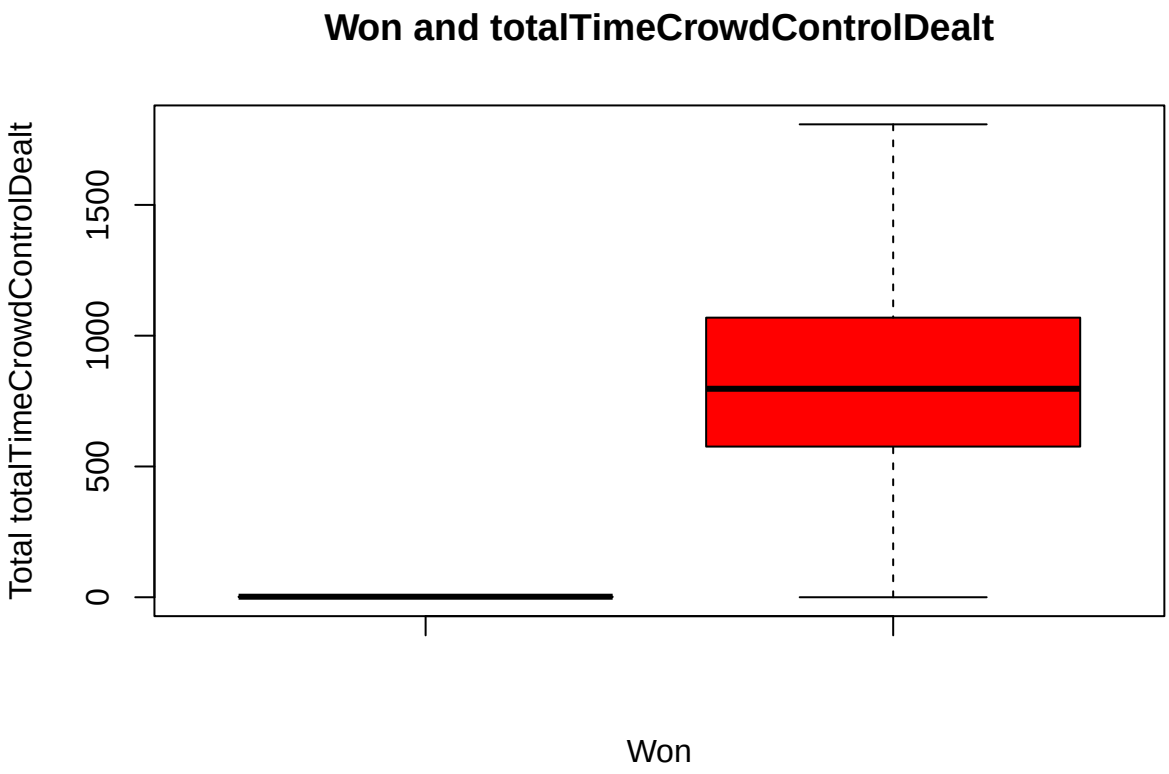

```
boxplot(full_stats_train$won, full_stats_train$TotalGold, main="Won and Gold Earned", xlab="Won", ylab=
```



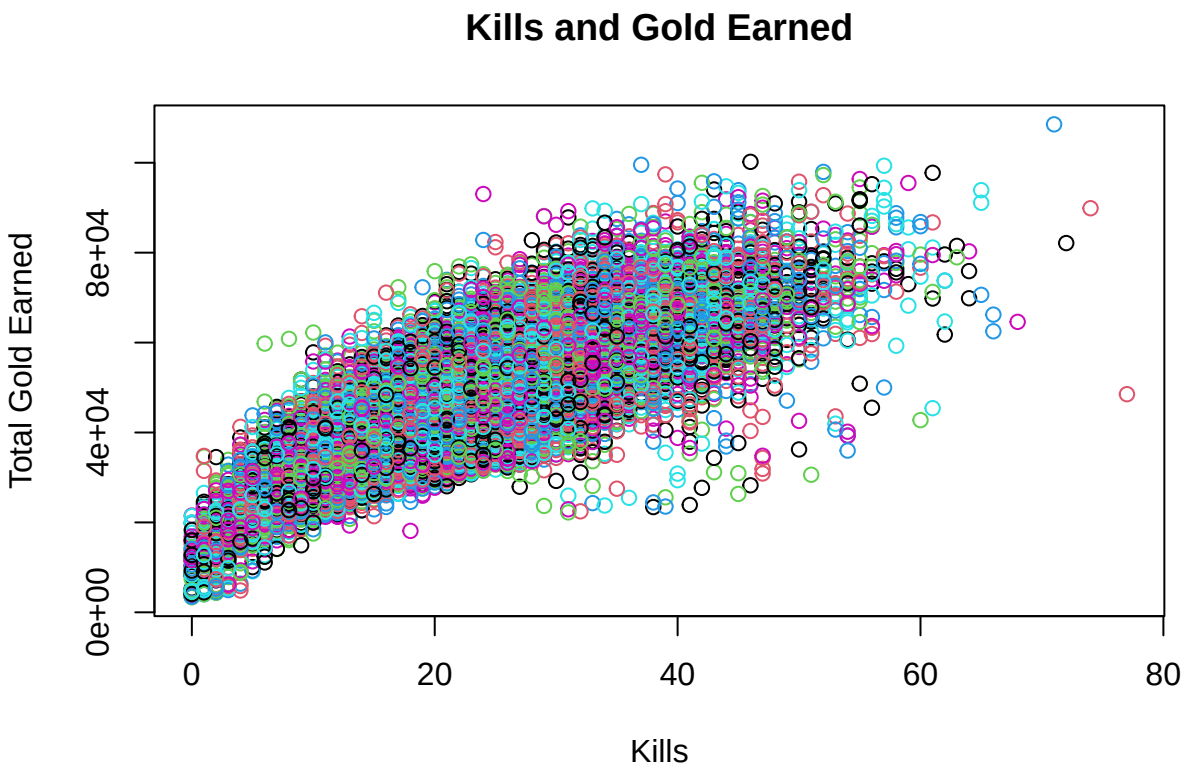

```
boxplot(full_stats_train$won, full_stats_train$TotalVision, main="Won and Vision Score", xlab="Won", yla
```



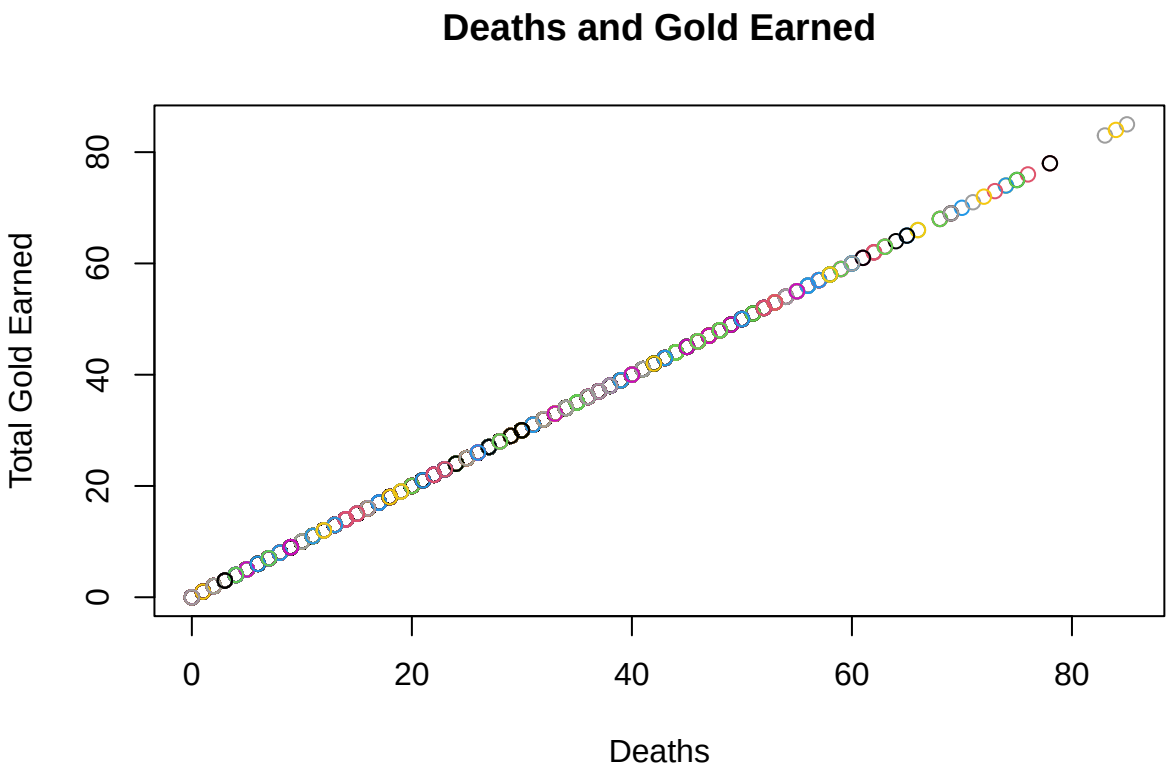

```
boxplot(full_stats_train$won, full_stats_train$TotalCrowdControl, main="Won and totalTimeCrowdControlDea
```



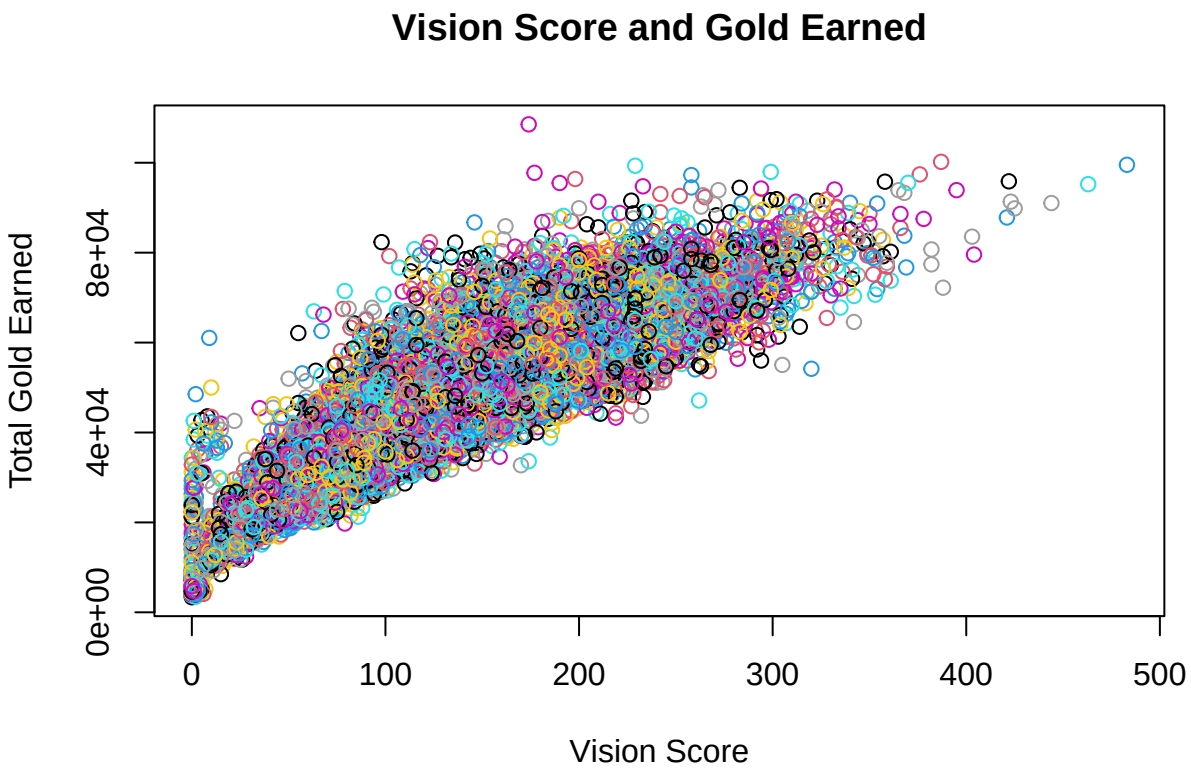

```
plot(full_stats_train$TotalKill, full_stats_train$TotalGold, main="Kills and Gold Earned", xlab="Kills"
```



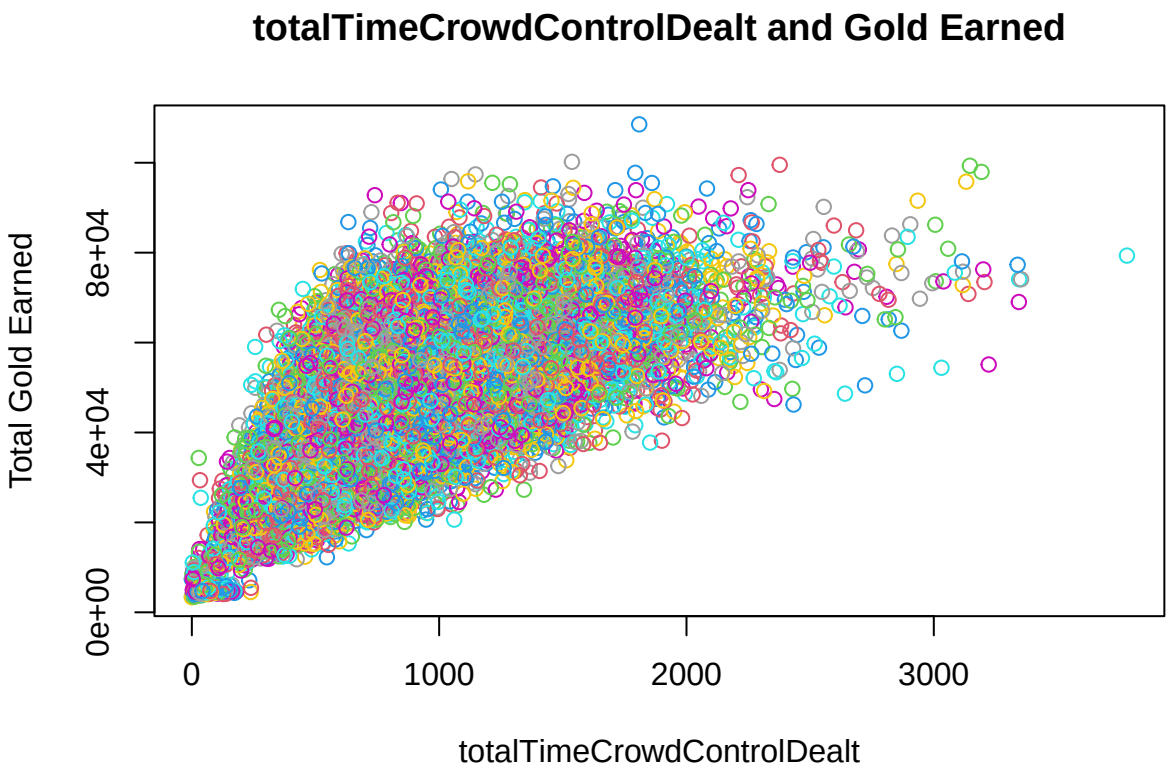

```
plot(full_stats_train$TotalDeath, full_stats_train$TotalDeath, main="Deaths and Gold Earned", xlab="Dea
```




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




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



SVM Classification

Linear

Training Now we will build our logistic regression model

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

```
##
## Call:
## svm(formula = won ~ ., data = full_stats_train, kernel = "linear",
##     cost = 10, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  3141
##
##  ( 1571 1570 )
##
##
## Number of Classes:  2
```

```
##
## Levels:
##  False True
```

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

```
library(caret)
```

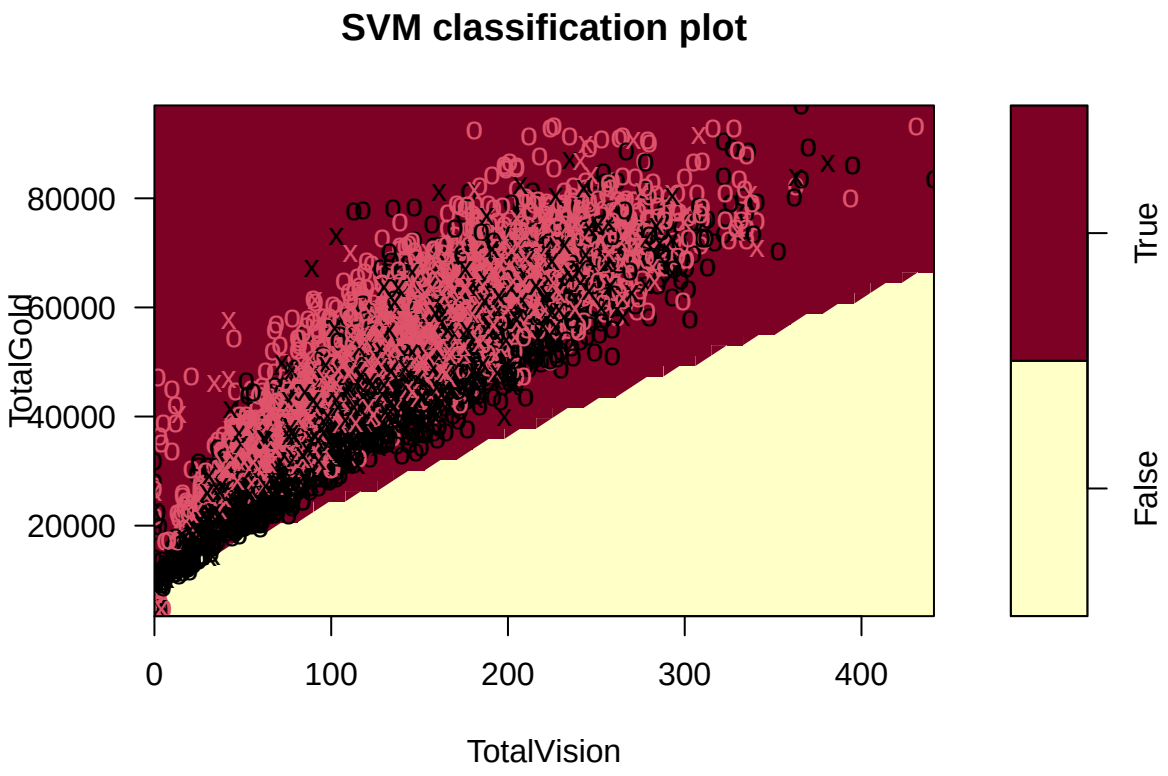
```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

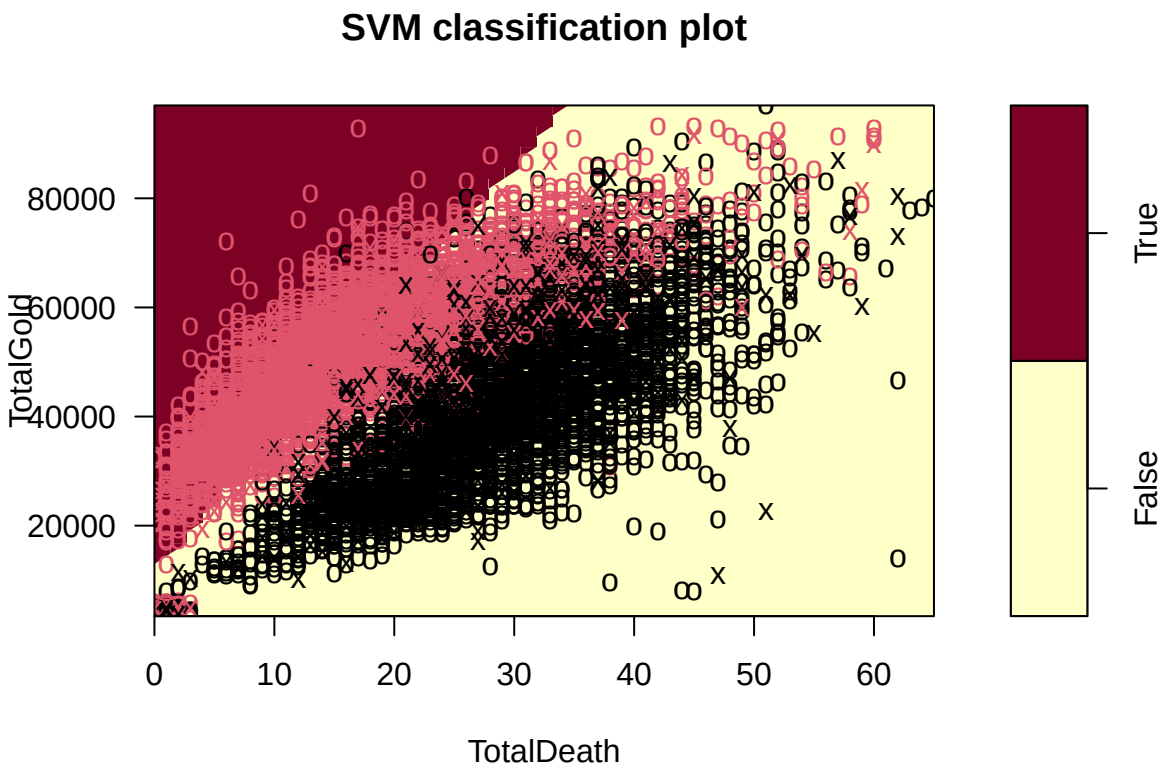
```
svm_probs_lin <- predict(svm_lin, newdata = full_stats_test)

confusionMatrix(svm_probs_lin, reference = full_stats_test$won)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  4353  254
##      True    179 4264
##
##                Accuracy : 0.9522
##                  95% CI : (0.9476, 0.9565)
##     No Information Rate : 0.5008
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9043
##
##  Mcnemar's Test P-Value : 0.0003762
##
##             Sensitivity : 0.9605
##             Specificity : 0.9438
##          Pos Pred Value : 0.9449
##          Neg Pred Value : 0.9597
##              Prevalence : 0.5008
##          Detection Rate : 0.4810
##    Detection Prevalence : 0.5091
##       Balanced Accuracy : 0.9521
##
##        'Positive' Class : False
##
```

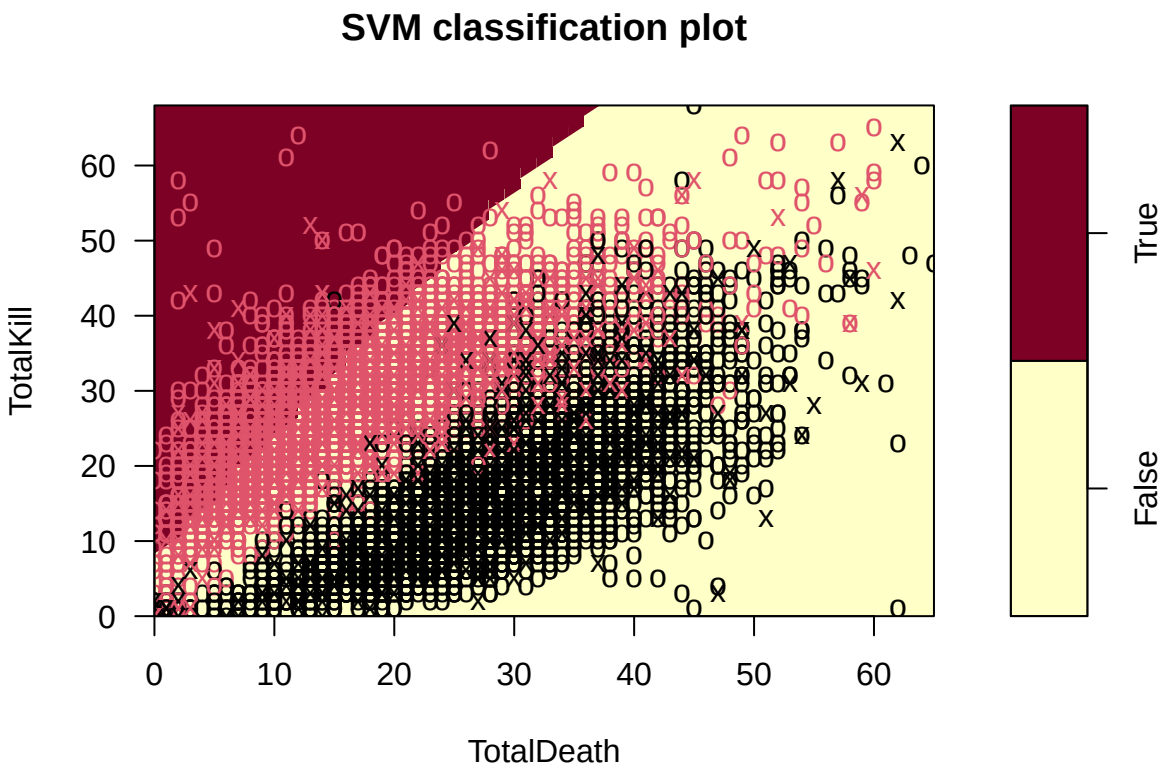
```
plot(svm_lin, full_stats_test, TotalGold ~ TotalVision)
```



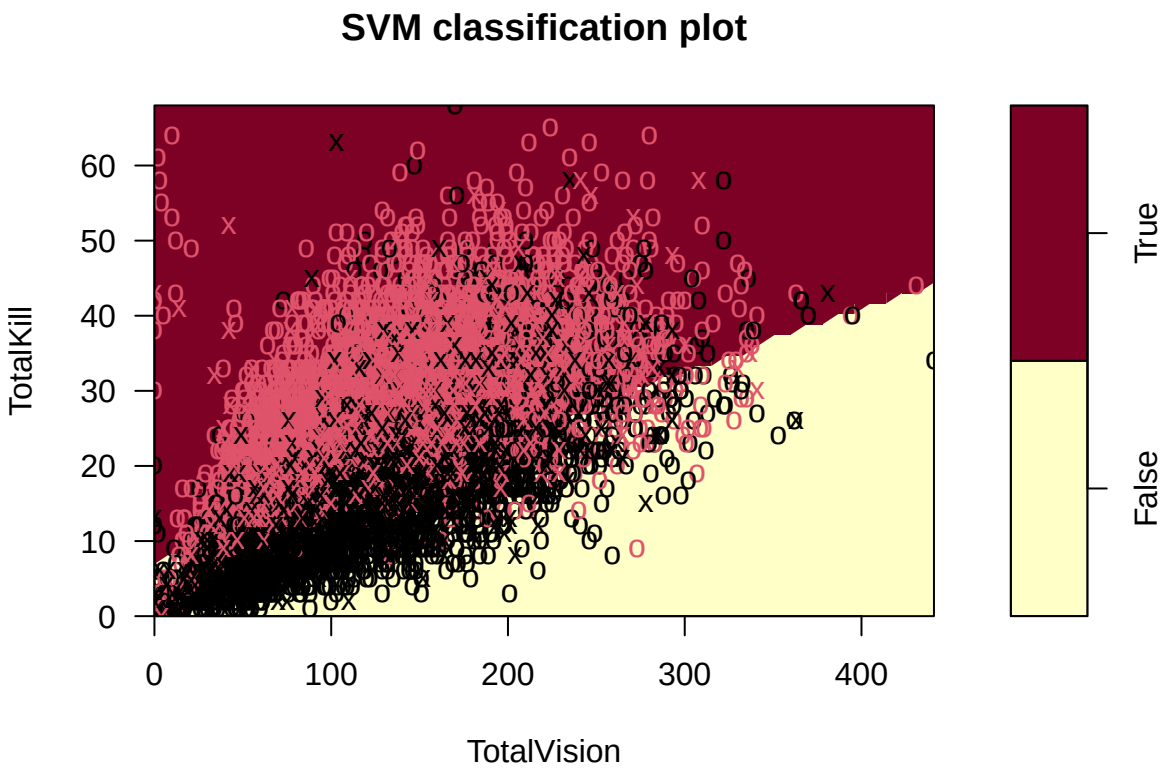

```
plot(svm_lin, full_stats_test, TotalGold ~ TotalDeath)
```



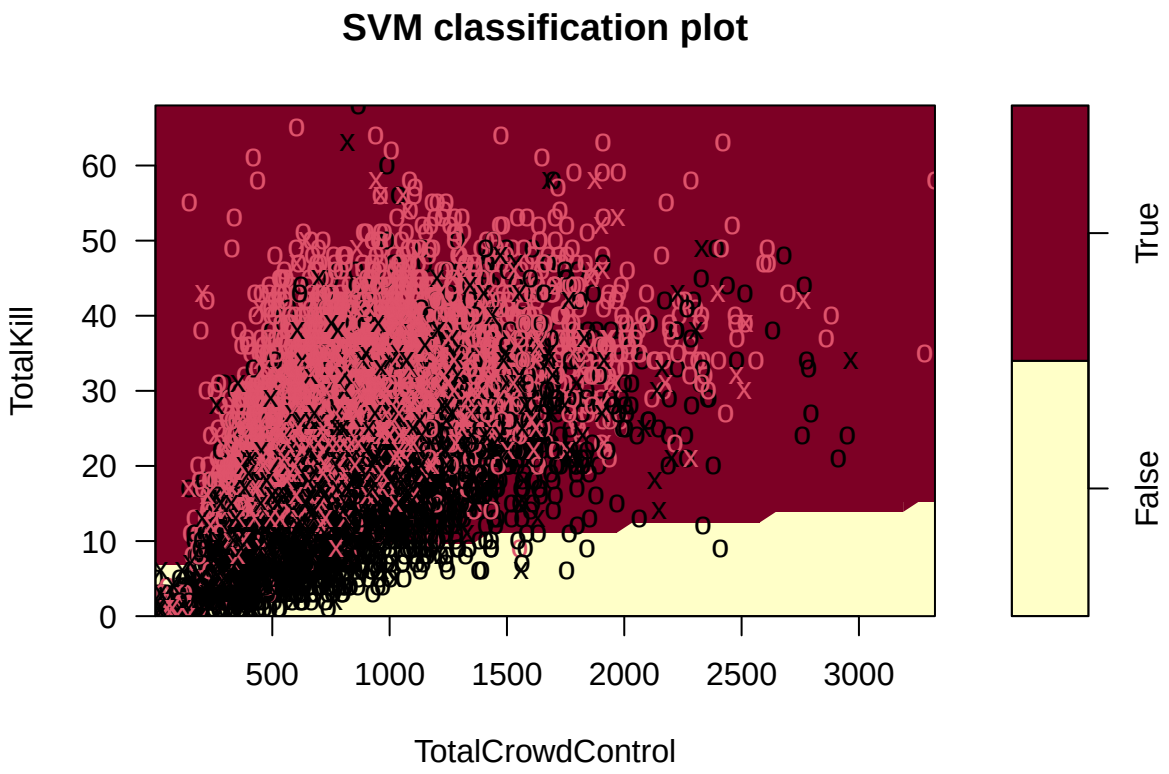

```
plot(svm_lin, full_stats_test, TotalKill ~ TotalDeath)
```



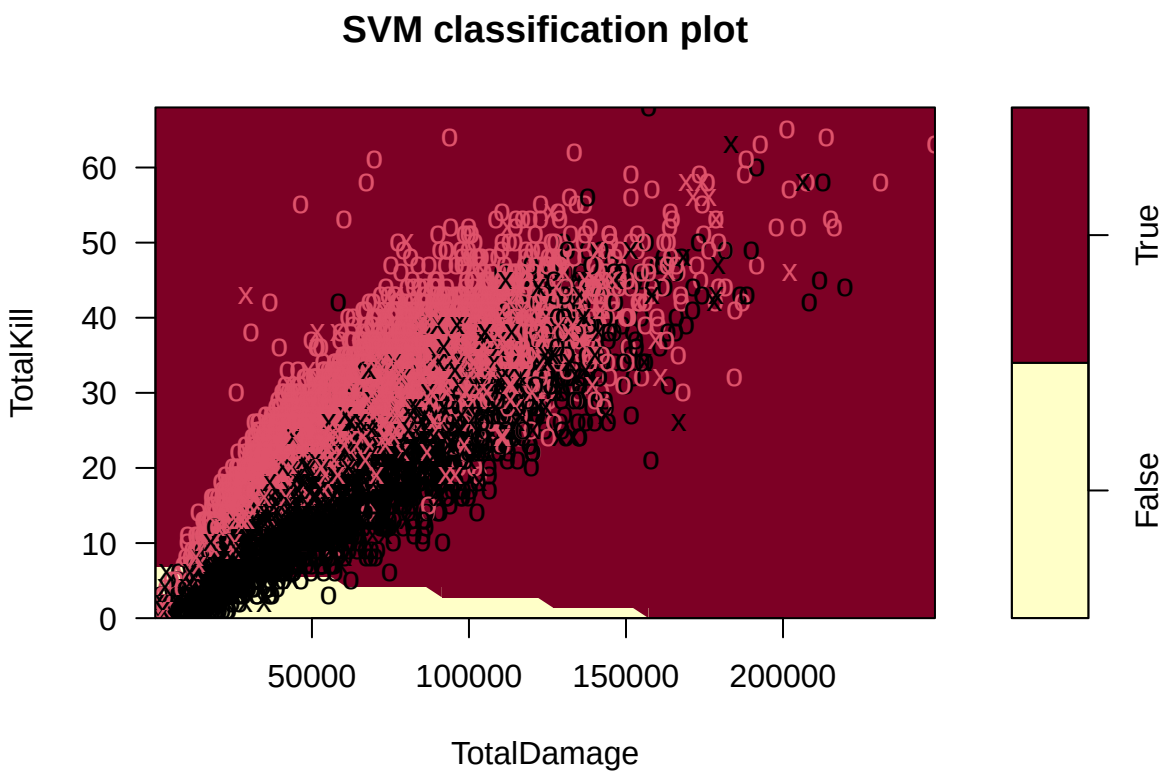

```
plot(svm_lin, full_stats_test, TotalKill ~ TotalVision)
```




```
plot(svm_lin, full_stats_test, TotalKill ~ TotalCrowdControl)
```




```
plot(svm_lin, full_stats_test, TotalKill ~ TotalDamage)
```



Polynomial

Training Now we will build our logistic regression model

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

```
##
## Call:
## svm(formula = won ~ ., data = full_stats_train, kernel = "polynomial",
##     cost = 10, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  3
##      coef.0:  0
##
## Number of Support Vectors:  5240
##
##  ( 2616 2624 )
##
##
## Number of Classes:  2
```

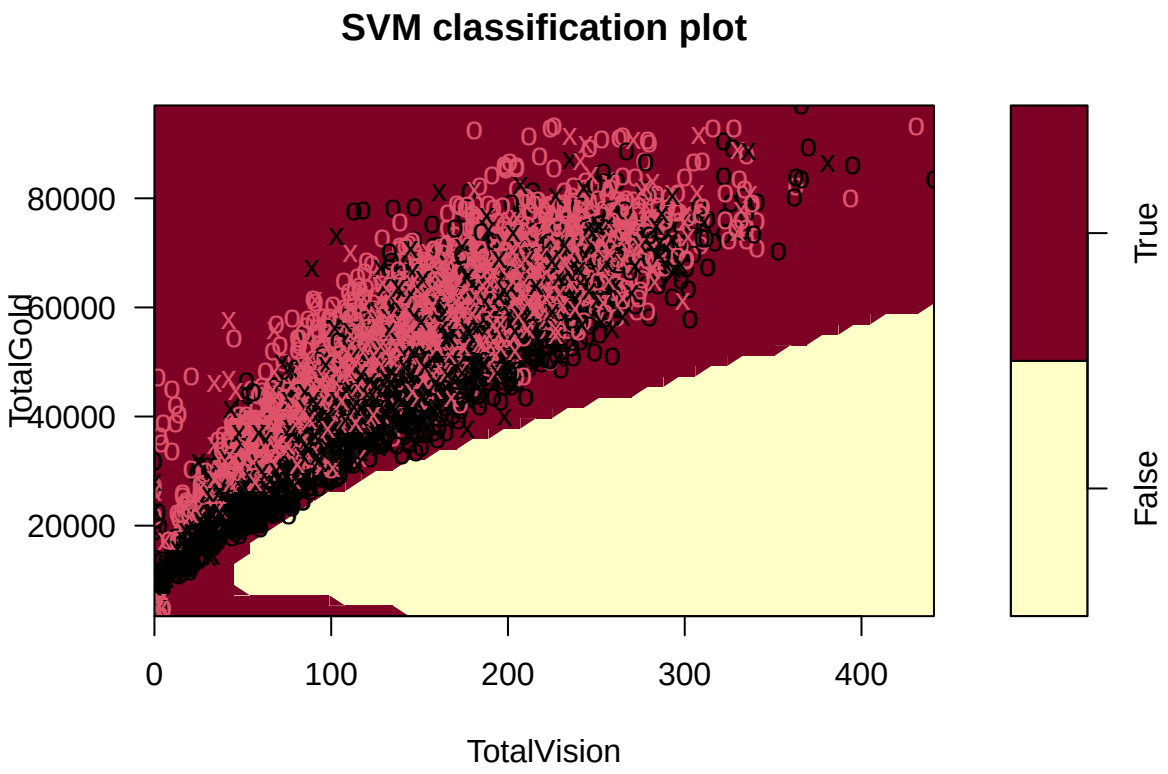
```
##
## Levels:
##  False True
```

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

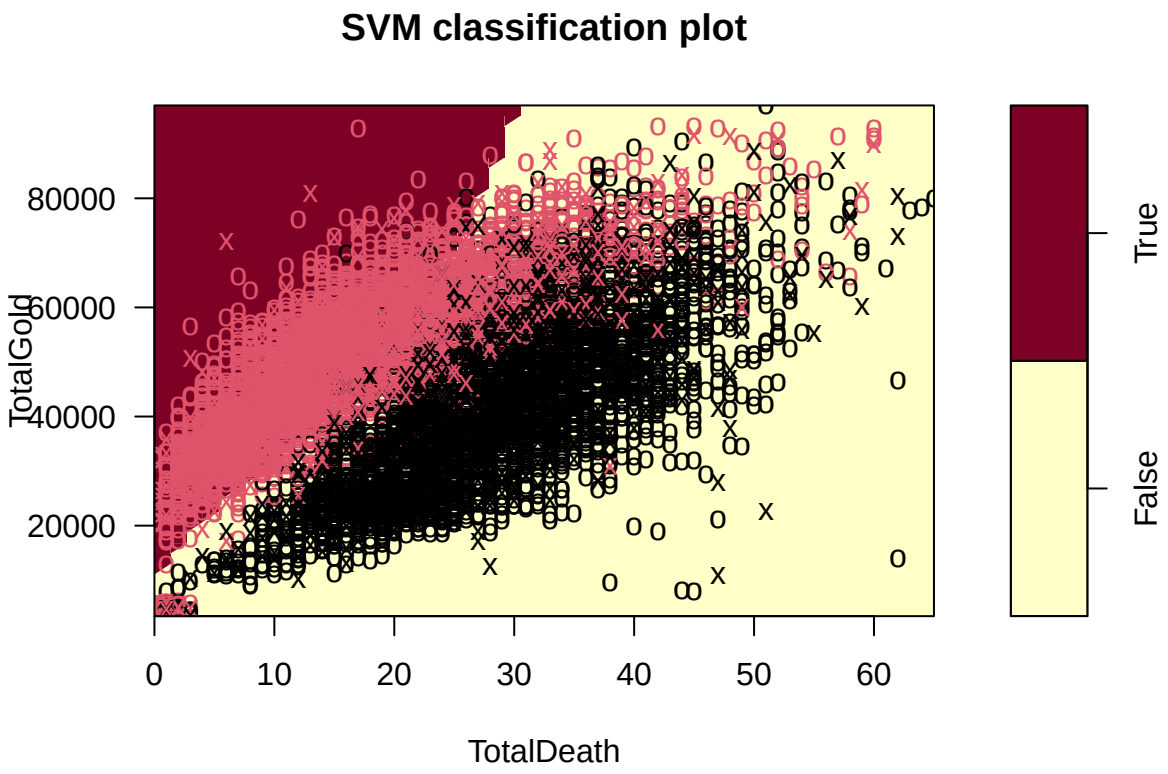
```
library(caret)
svm_probs_poly <- predict(svm_poly, newdata = full_stats_test)

confusionMatrix(svm_probs_poly, reference = full_stats_test$won)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  4326  222
##      True    206 4296
##
##                Accuracy : 0.9527
##                  95% CI : (0.9481, 0.957)
##     No Information Rate : 0.5008
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9054
##
##  Mcnemar's Test P-Value : 0.4684
##
##             Sensitivity : 0.9545
##             Specificity : 0.9509
##          Pos Pred Value : 0.9512
##          Neg Pred Value : 0.9542
##              Prevalence : 0.5008
##          Detection Rate : 0.4780
##    Detection Prevalence : 0.5025
##       Balanced Accuracy : 0.9527
##
##        'Positive' Class : False
##
```

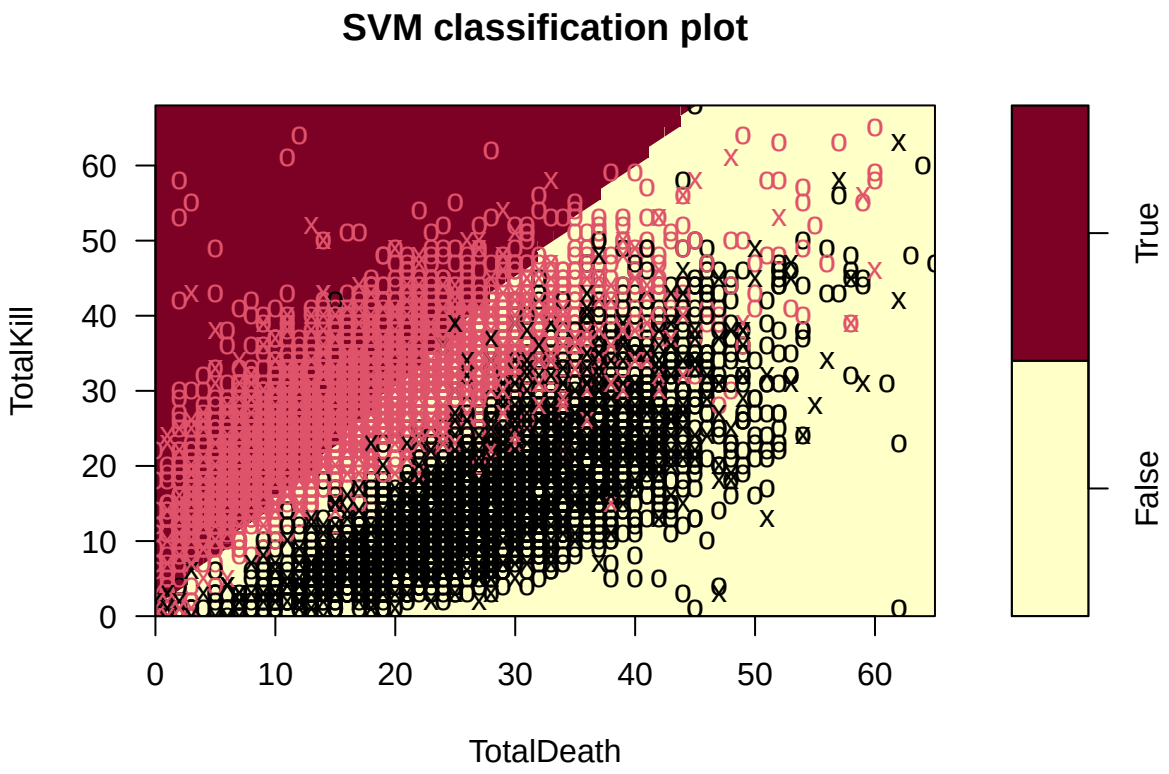
```
plot(svm_poly, full_stats_test, TotalGold ~ TotalVision)
```



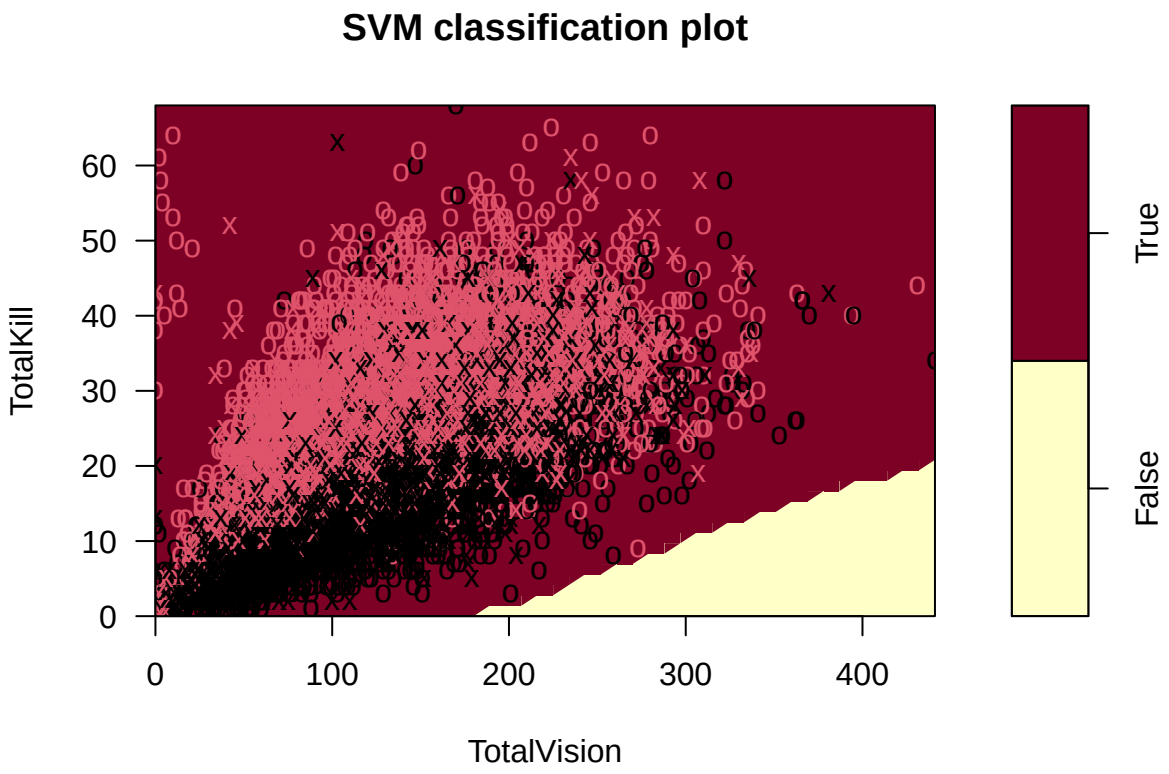

```
plot(svm_poly, full_stats_test, TotalGold ~ TotalDeath)
```



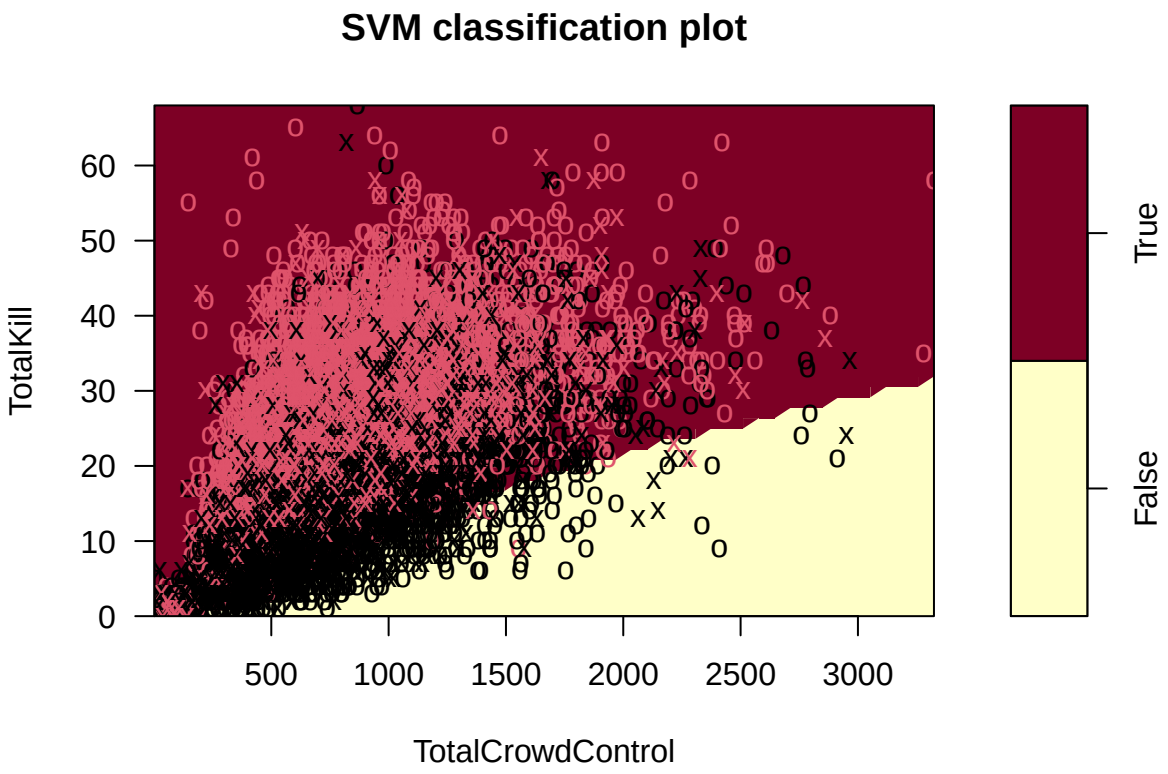

```
plot(svm_poly, full_stats_test, TotalKill ~ TotalDeath)
```



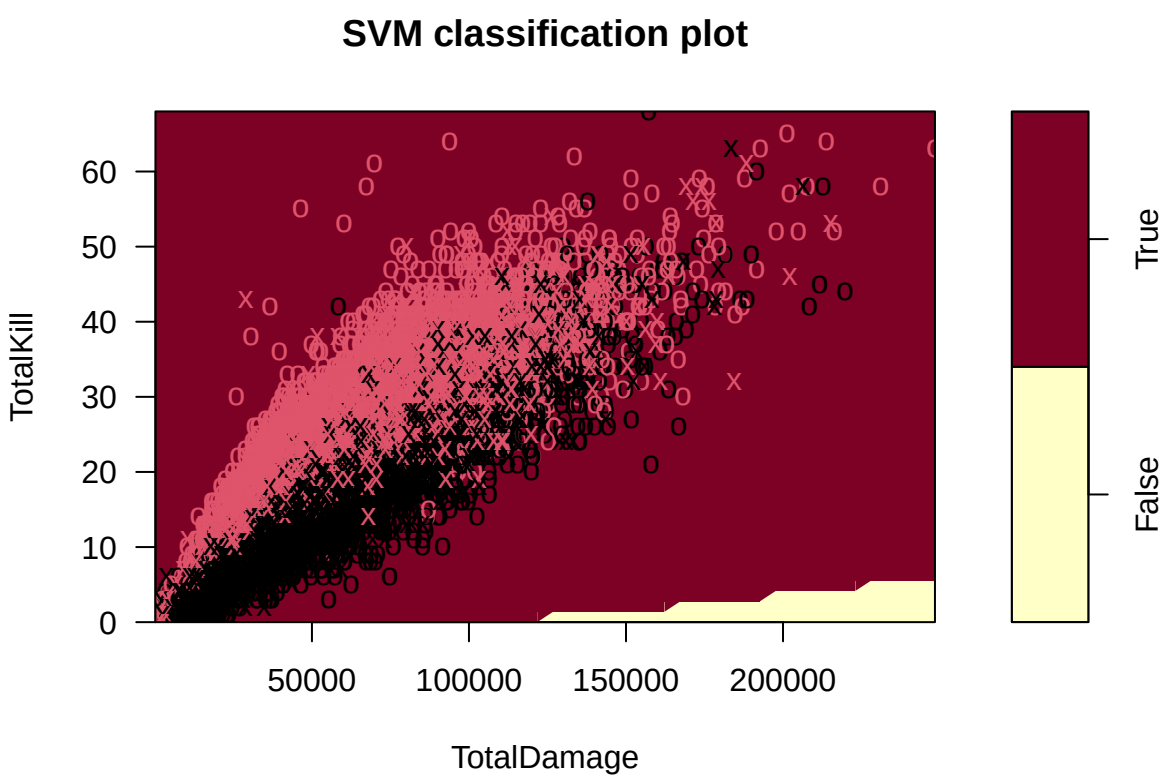

```
plot(svm_poly, full_stats_test, TotalKill ~ TotalVision)
```




```
plot(svm_poly, full_stats_test, TotalKill ~ TotalCrowdControl)
```




```
plot(svm_poly, full_stats_test, TotalKill ~ TotalDamage)
```



Radial

Training Now we will build our logistic regression model

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

```
##
## Call:
## svm(formula = won ~ ., data = full_stats_train, kernel = "radial",
##     cost = 10, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##
## Number of Support Vectors:  2957
##
##  ( 1461 1496 )
##
##
## Number of Classes:  2
##
## Levels:
```

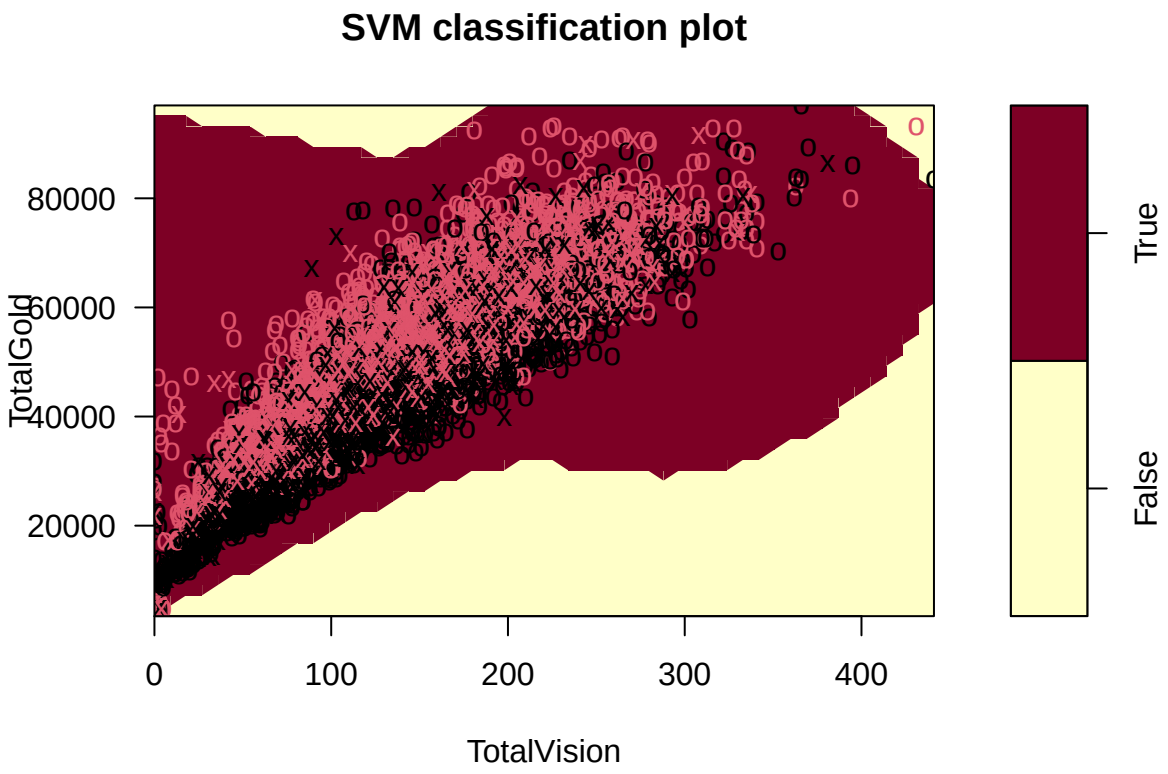
```
##  False True
```

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

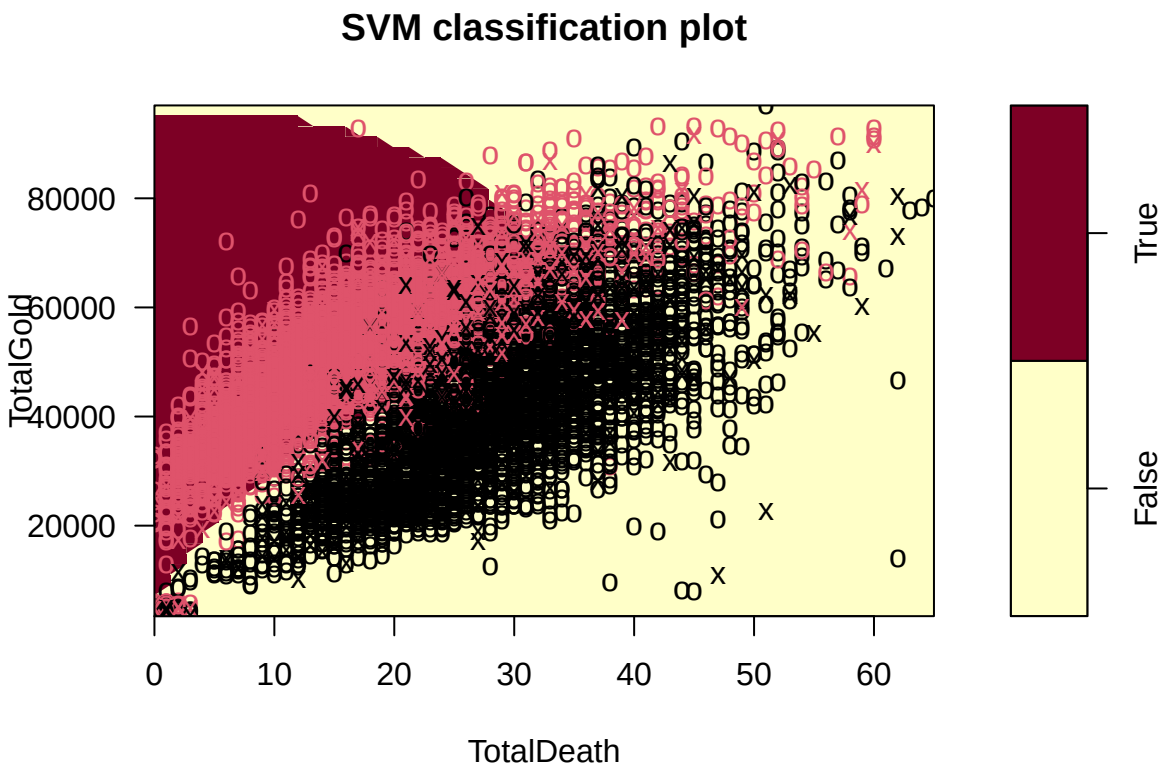
```
library(caret)
svm_probs_rad <- predict(svm_rad, newdata = full_stats_test)

confusionMatrix(svm_probs_rad, reference = full_stats_test$won)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  4320  186
##      True    212 4332
##
##                Accuracy : 0.956
##                  95% CI : (0.9516, 0.9602)
##     No Information Rate : 0.5008
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.912
##
##  Mcnemar's Test P-Value : 0.2102
##
##             Sensitivity : 0.9532
##             Specificity : 0.9588
##          Pos Pred Value : 0.9587
##          Neg Pred Value : 0.9533
##              Prevalence : 0.5008
##          Detection Rate : 0.4773
##    Detection Prevalence : 0.4979
##       Balanced Accuracy : 0.9560
##
##        'Positive' Class : False
##
```

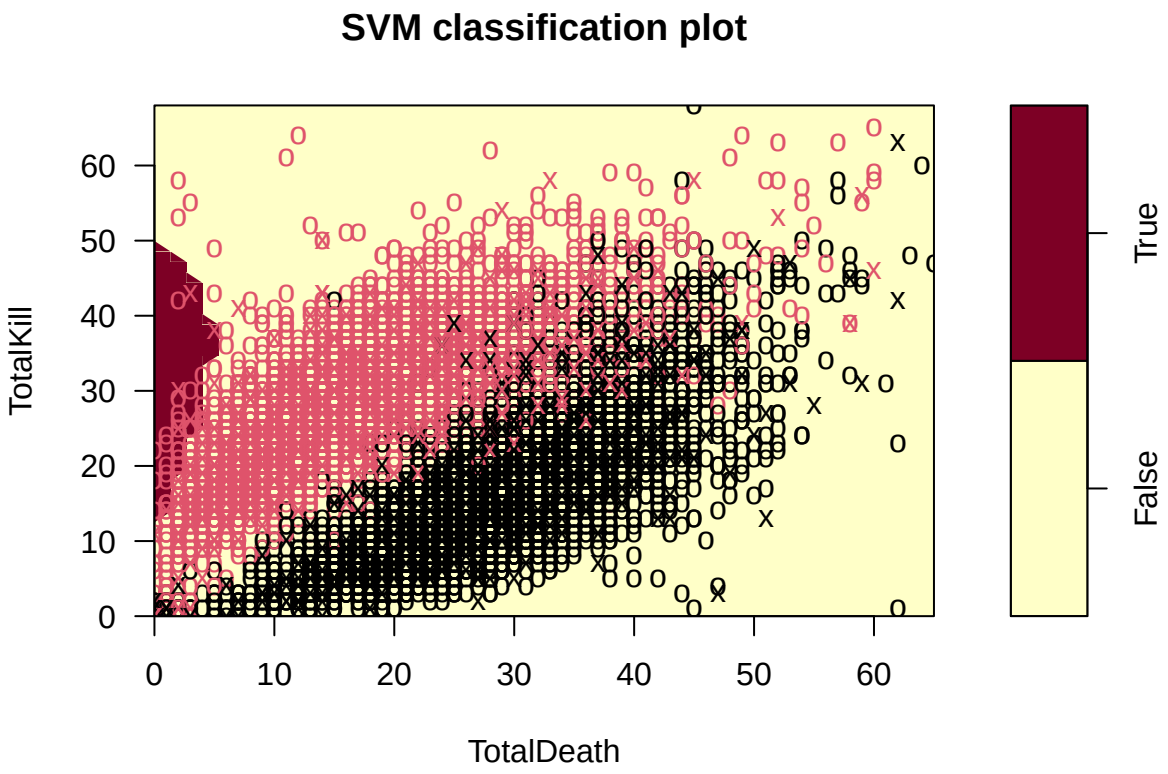
```
plot(svm_rad, full_stats_test, TotalGold ~ TotalVision)
```



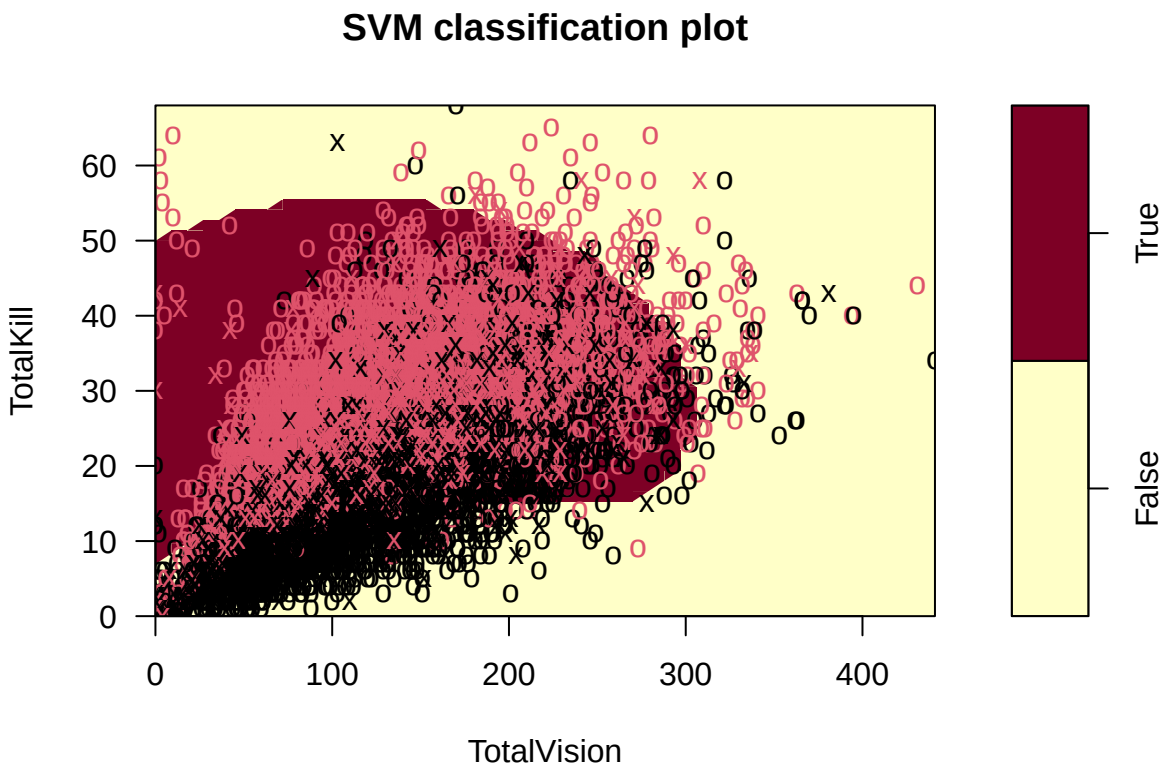

```
plot(svm_rad, full_stats_test, TotalGold ~ TotalDeath)
```



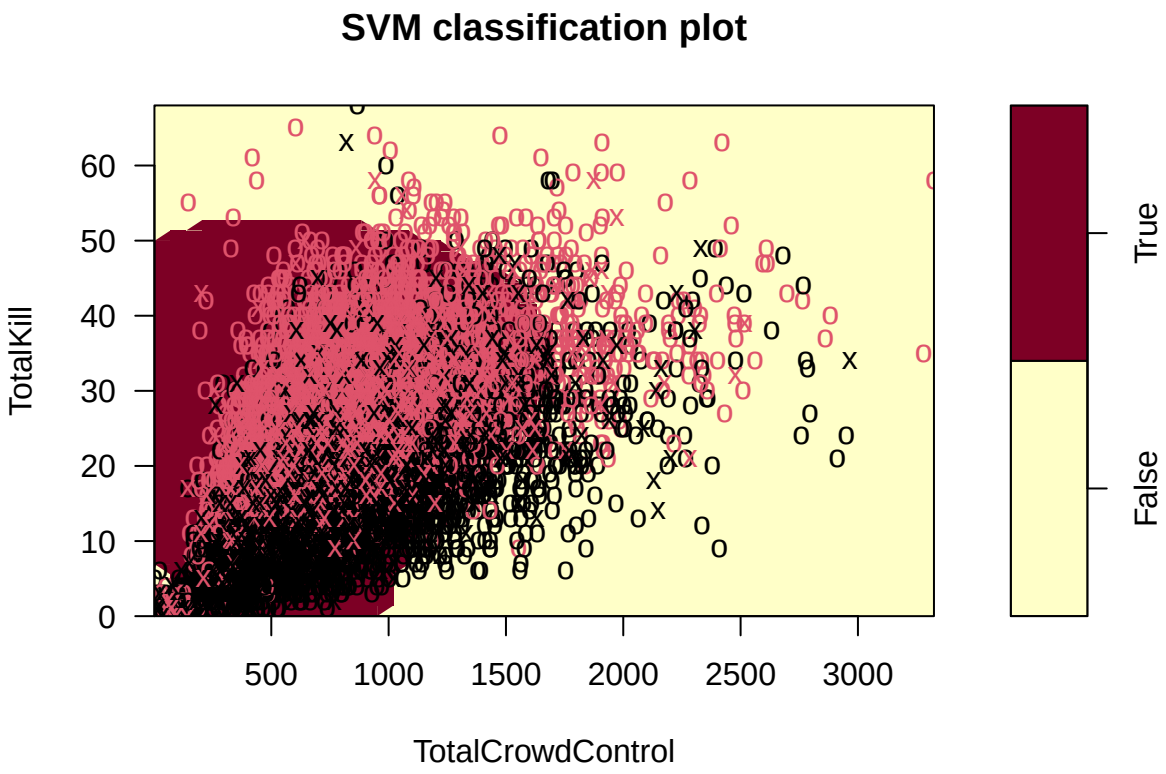

```
plot(svm_rad, full_stats_test, TotalKill ~ TotalDeath)
```



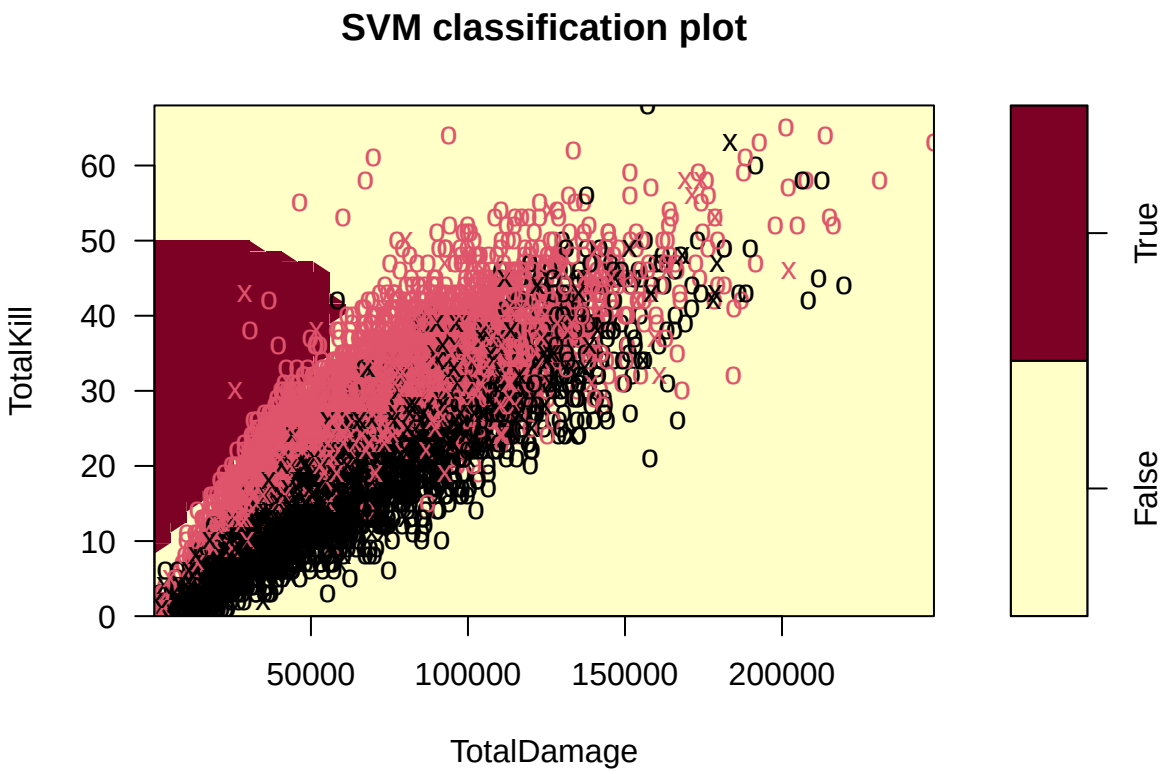

```
plot(svm_rad, full_stats_test, TotalKill ~ TotalVision)
```




```
plot(svm_rad, full_stats_test, TotalKill ~ TotalCrowdControl)
```




```
plot(svm_rad, full_stats_test, TotalKill ~ TotalDamage)
```



SVM Linear vs Polynomial vs Radial

Analyzing the results of each model based on the algorithms.

SVM Linear

Our SVM Linear model utilized 3128 support vectors along the margin. It had an accuracy of 95.48%, the balanced accuracy is the same! This is a fairly high accuracy, which isn't surprising to me because our data as seen in the graphs is very linear and could be split easily using a linear function. The sensitivity is about .96 and the specificity is about .95 which are both close to 1 and very good. The p-value is also very low which shows this a good model. The Kappas is about .91, which shows a excellent positive agreement. Overall, this model has very good metrics and is a very good model for our data.

SVM Polynomial

Our SVM Linear model utilized 5197 support vectors along the margin. It had an accuracy of 95.54%, the balanced accuracy is the same! This is a fairly high accuracy, which isn't surprising to me because our data as seen in the graphs is very linear and could be split easily using a polynomial function. The sensitivity is about .94 and the specificity is about .96 which are both close to 1 and very good. The p-value is also very low which shows this a good model. The Kappas is about .91, which shows a excellent positive agreement. Overall, this model has very good metrics and is a very good model for our data.

SVM Radial

Our SVM Linear model utilized 2914 support vectors along the margin. It had an accuracy of 95.54%, the balanced accuracy is 95.53%. This is a fairly high accuracy, which isn't surprising to me because our data as seen in the graphs is very linear and could be split easily using a polynomial function. The sensitivity is

about .94 and the specificity is about .96 which are both close to 1 and very good. The p-value is also very low which shows this a good model. The Kappas is about .91, which shows a excellent positive agreement. Overall, this model has very good metrics and is a very good model for our data.

Summary

The SVM Polynomial had the most support vectors and highest accuracy of all the models. Therefore, I believe it is the best model out of the three. The SVM Radial model has the second highest (balanced) accuracy but the least support vectors. Which I found interesting as I didn't think the data could be represented easily by a radial model, however since there is a fair amount of clusters in the data as seen in the graphs it makes some sense. The SVM Linear model had the second most support vectors and the lowest accuracy, however this model still does a good job of representing the data. Overall, all of the models are very good and represent our data very well.