Valorant Win Predictions - EXST 7152

Dina Dinh

2024-03-24

Contents

Required Packages

```
library(easypackages)
libraries("readxl", "tidyverse", "ggplot2", "gridExtra", "viridis", "MASS", "AUC", "caret",
          "car", "glmnet", "pls", "mda", "randomForest", "psych")
```

Introduction

Valorant, a competitive first-person shooter game, was developed and released by Riot Games in June 2020. It is characterized as a team-based tactical shooter that prioritizes precise aiming, strategic coordination, and the effective employment of distinctive agent abilities. The game presents an array of agents, each endowed with unique capabilities and an assortment of weapons, enabling players to adopt strategic maneuvers to secure an advantage over the opposing team. Participants are split into two factions—attackers and defenders—and partake in a succession of rounds aiming to either plant or defuse a bomb or to eliminate the rival team entirely. Usually, the game is over once one of the two factions wins 13 rounds in a match.

The intricacies of shooting mechanics are fundamental to the gameplay of Valorant, demanding from players not only proficiency in aiming but also in controlling recoil. Furthermore, the game accentuates the importance of strategic positioning, dominance over the map, and the judicious application of utilities. Effective communication and coordination within a team are imperative, requiring players to make immediate decisions and adapt to the dynamically evolving circumstances of each round. Following each match, players receive an exhaustive report detailing their performance, thereby raising a pertinent question: Is it feasible for the statistical data of a single player to predict the outcome of a match? Consequently, this leads to an examination of which specific statistics are crucial in forecasting whether a match will culminate in a victory or defeat.

Data Source

The dataset includes player statistics from individual matches, such as Average Combat Score (ACS), the agent used, the map played, and the kill/death ratio. This information was retrieved from the player's competitive history on tracker.gg. The goal is to analyze these statistics to determine which factors might predict the outcome of a match, focusing on identifying key performance indicators that influence whether a game is won or lost. The dataset is as of February 16, 2023 and consists of 20 variables:

1. Date
2. Agent: 21 agents to choose from.
3. Map: 9 maps randomly assigned to each match.
4. Placement: shows the placement of target player compared to the other 9 players in the match based on ACS.
5. Match score: shows the final score of the match (removed)
6. Duration: total length of the match in minutes.
7. Average Rank of Match: average of all the players' ranks in each match.
8. Average Combat Score (ACS): calculated based on kills, multi-kills, and damage done to the enemy team.
9. A: amount of times a teammate killed an enemy the target player done at least 50 damage to. 10.K: amount of kill in the match.
10. D: amount of times died in the match.
11. K_D: ratio of kills per death.
12. Average Damage Per Round (ADR): average damage the target player did per round.
13. HS: headshot percentage.
14. KAST: percent of rounds where you got a kill, assist, survived, or traded.
15. First Kill: amount of rounds to be the first person on the team to get a kill in a round.
16. First Death: amount of rounds to be the first to die in a round.
17. MK (Multikill): amount of rounds getting more than 1 kill during a round.
18. Econ: how well the target player managed their in-game currency during the match.
19. Result: outcome of the match.

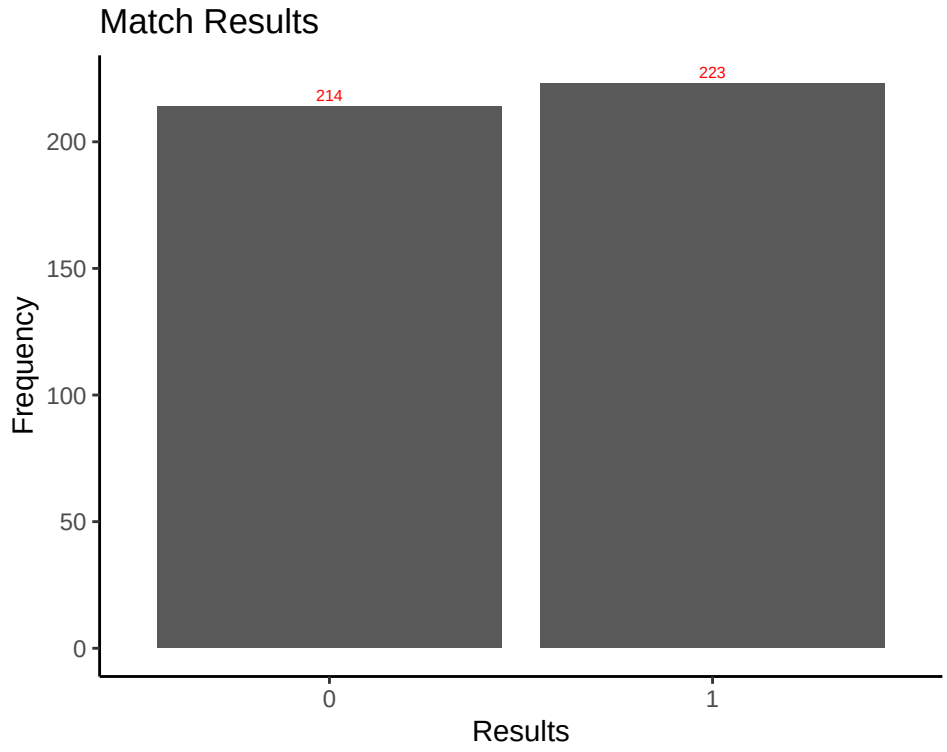
Data Cleaning

In the revised analysis, the variable "Match Score" has been removed from consideration. Categorical variables have been converted into factor types. Additionally, the response variable "Results" has been converted into a binary format. Notably, the outcome "D," representing draws, has been excluded from the analysis due to its limited occurrence in only three matches (observations).

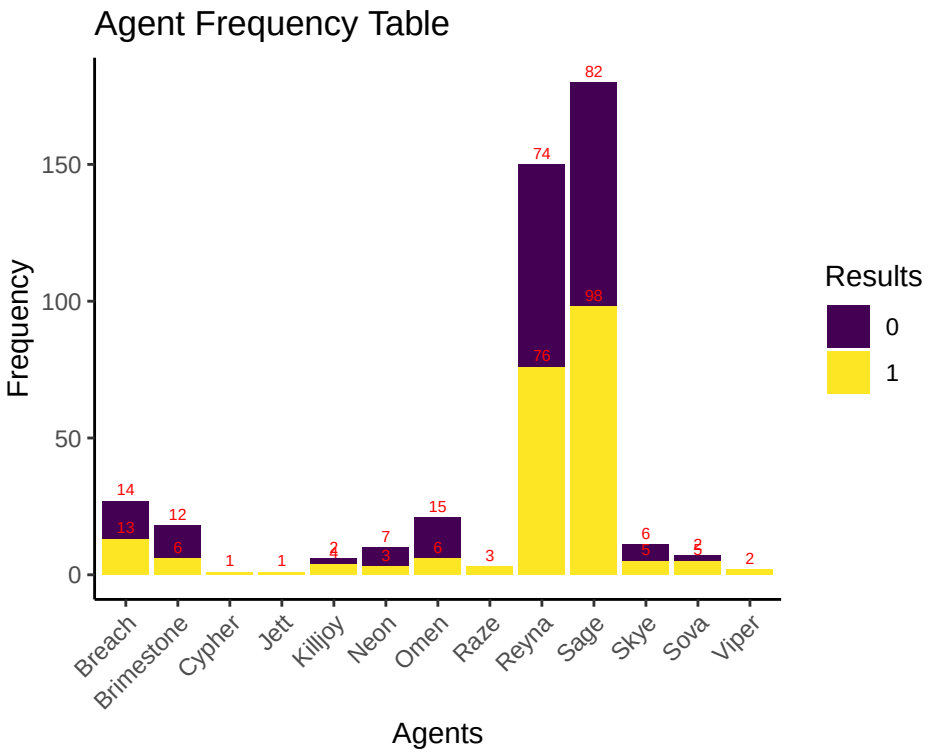
Analyzing the Data

Summary Statistics

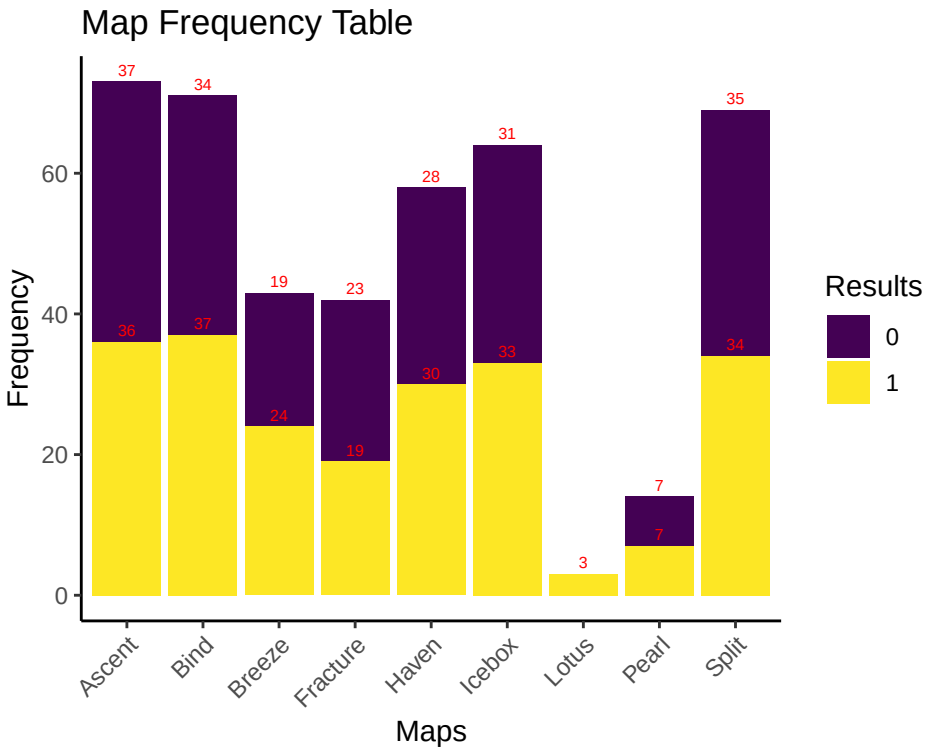
First, let's take a look at some graphs of the explanatory variables as well as the distribution of response variable.



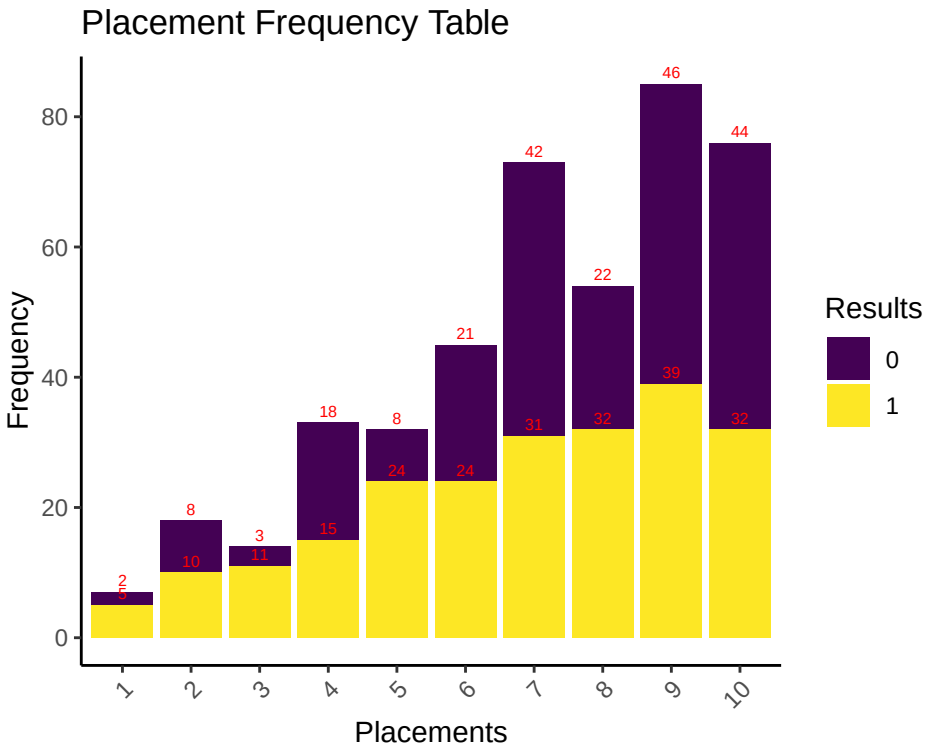
The response variable seems to have a balance of both wins and loses.



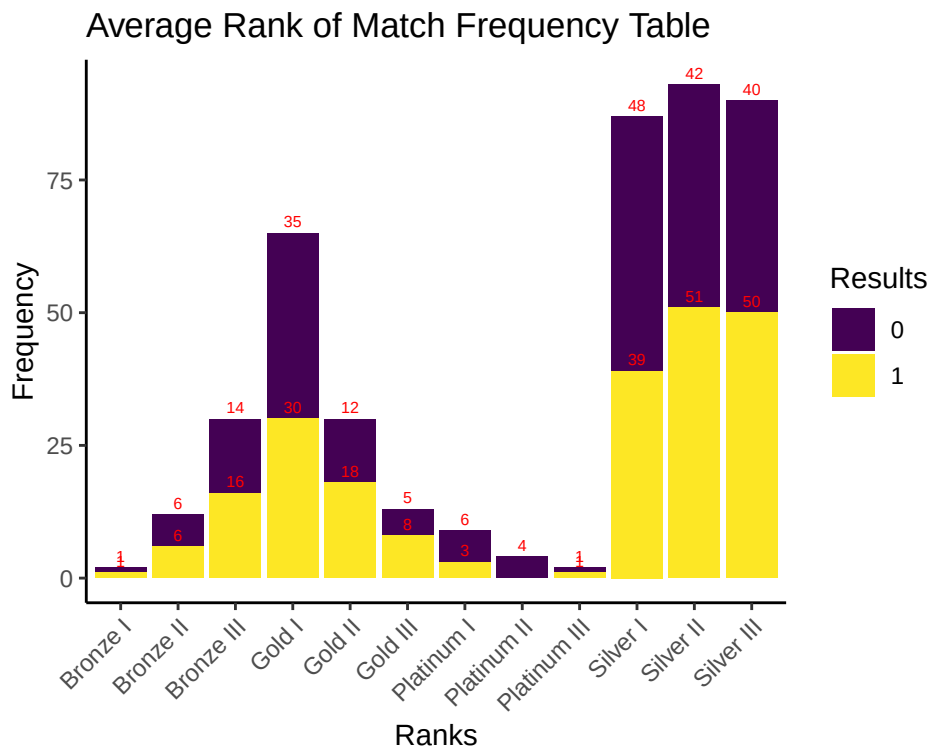
Here we can see an imbalance of the agents chosen. The target player seems to favor playing some agents more than others.



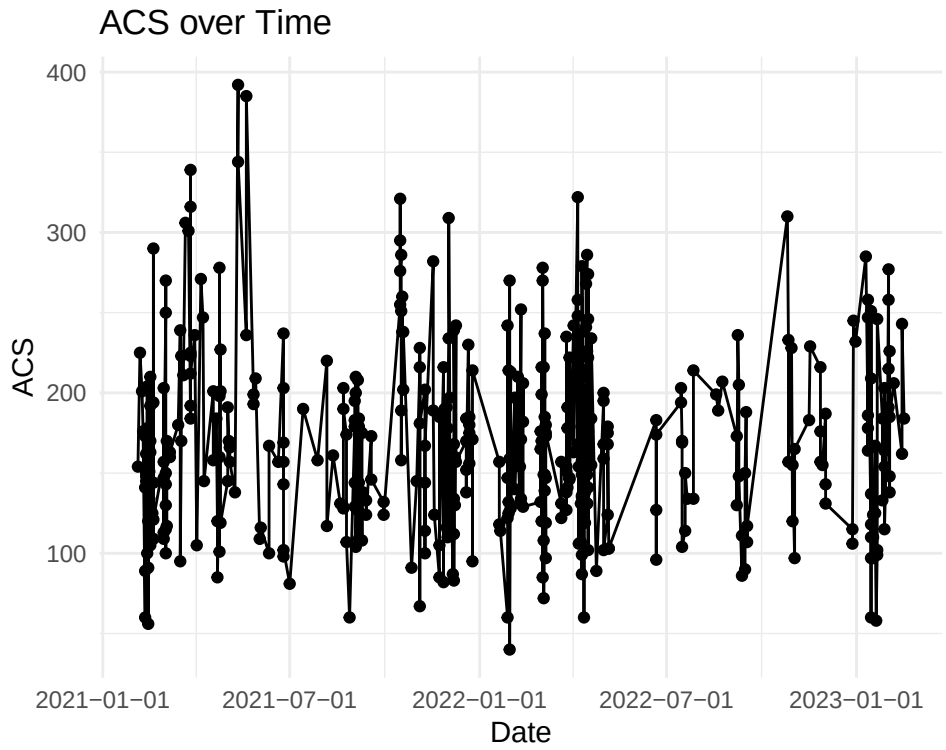
The maps are chosen randomly per match. The newer maps tend to have less frequency than the older maps. The wins and losses seem balanced for each map.



The player in the analysis tend to place below 5 with wins and losses practically even for each placement. There's noticeably more wins when the player places 5th.

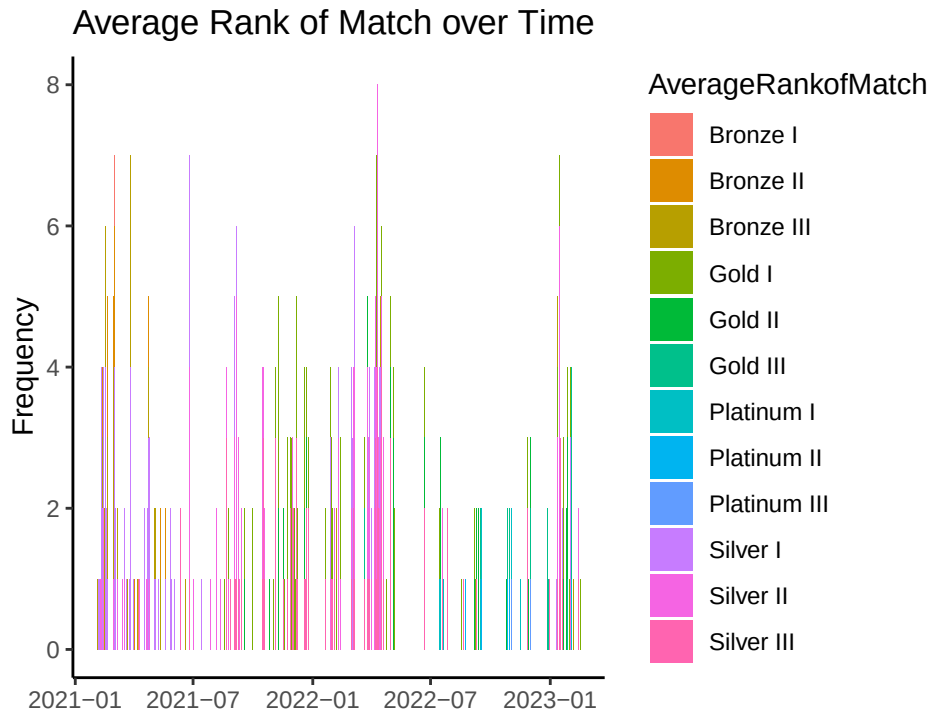


The player tend to be in matches with other Silver ranked player and occasionally Gold rank. Again, the wins and losses seem to be balanced for each rank too.

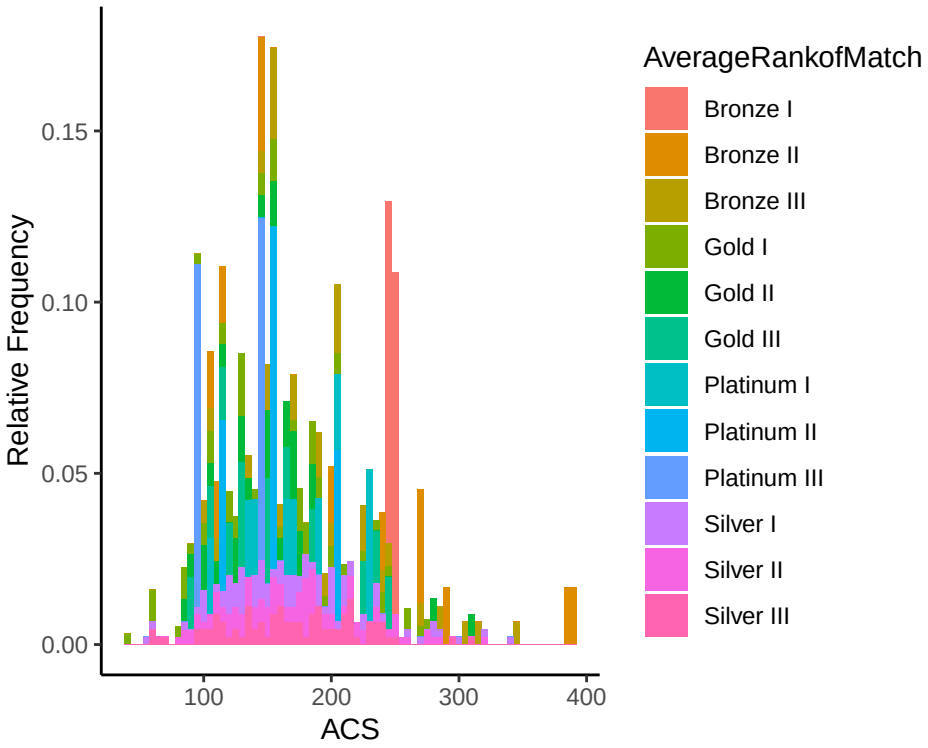


Most players tend to track their performance by using ACS. The higher the ACS the better the player

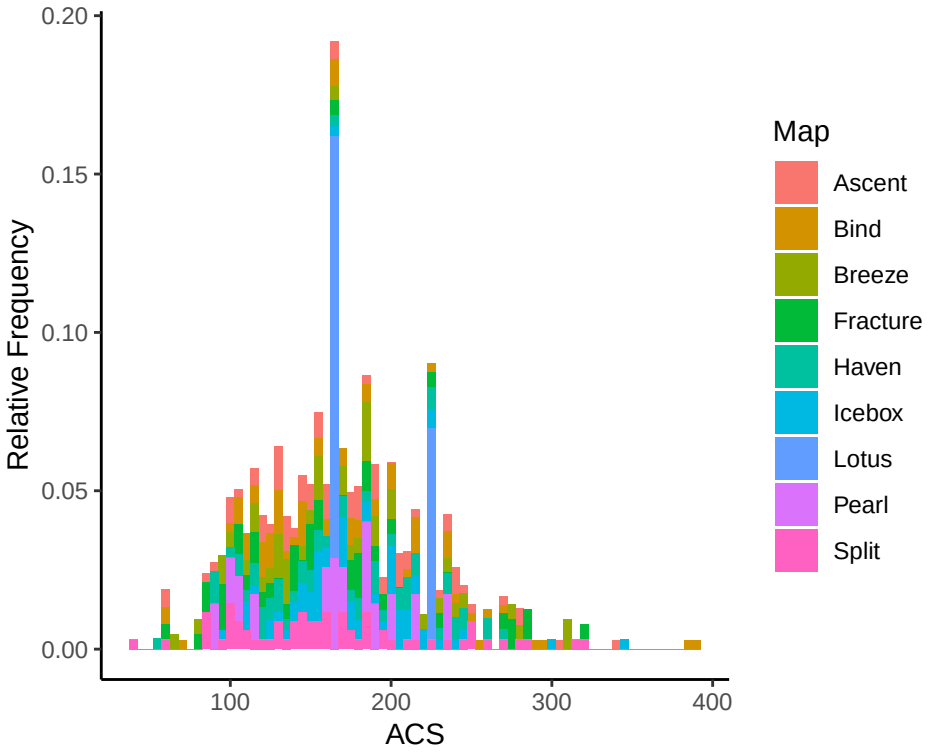
performed in the match. We can see that the ACS looks consistent, showing not much improvement in the player's performance over the years, but let's see if the other players' ranks in the match increased over time.



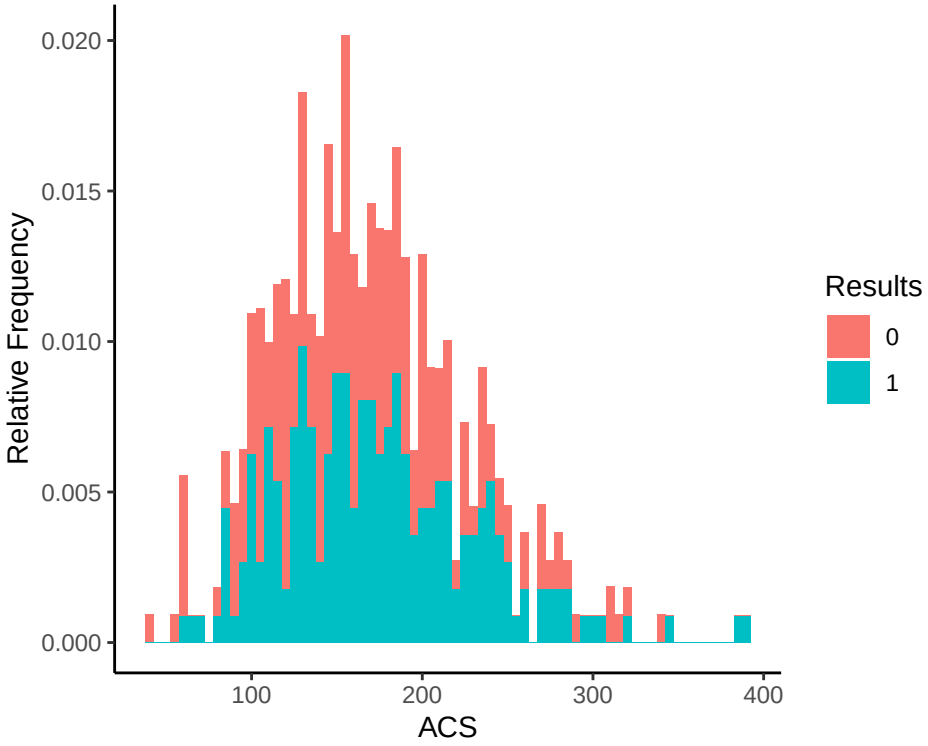
The player started in the Bronze-Silver range, but you can see a bit more occurrences of Platinum/Gold in the later year. This can cause the player's ACS to be consistent over time when the level of the difficulty in the matches increases.



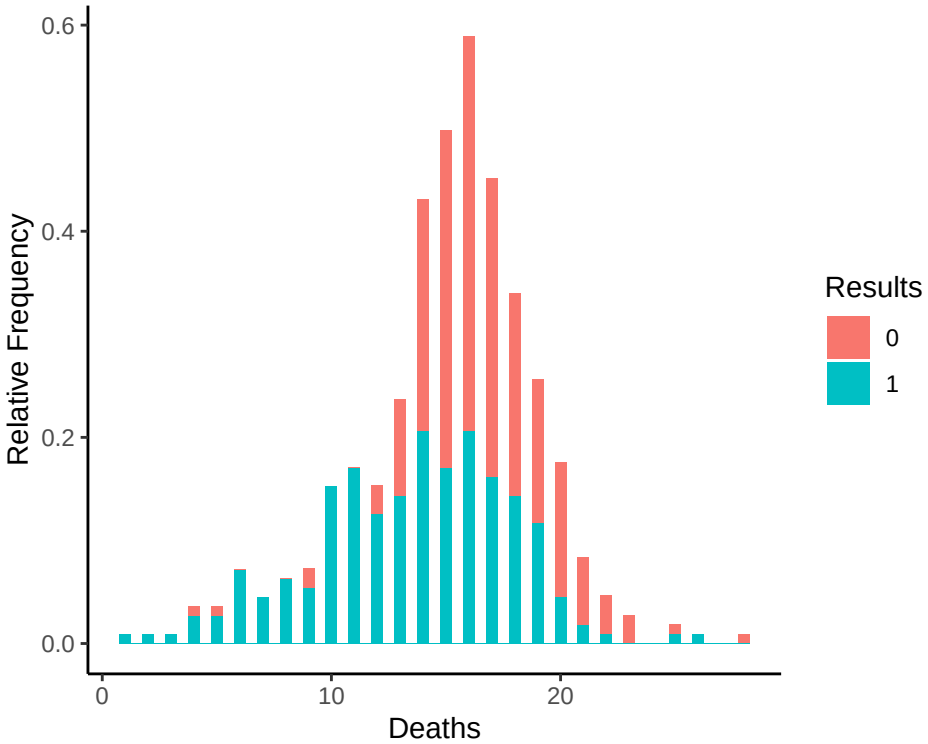
Most of the player's ACS ranged between 100 to 200 across all ranks of the matches and seems to be a bit higher in the lower ranks such as Bronze.



The player's ACS seems consistent across all maps, potentially slightly better in Lotus and Icebox.

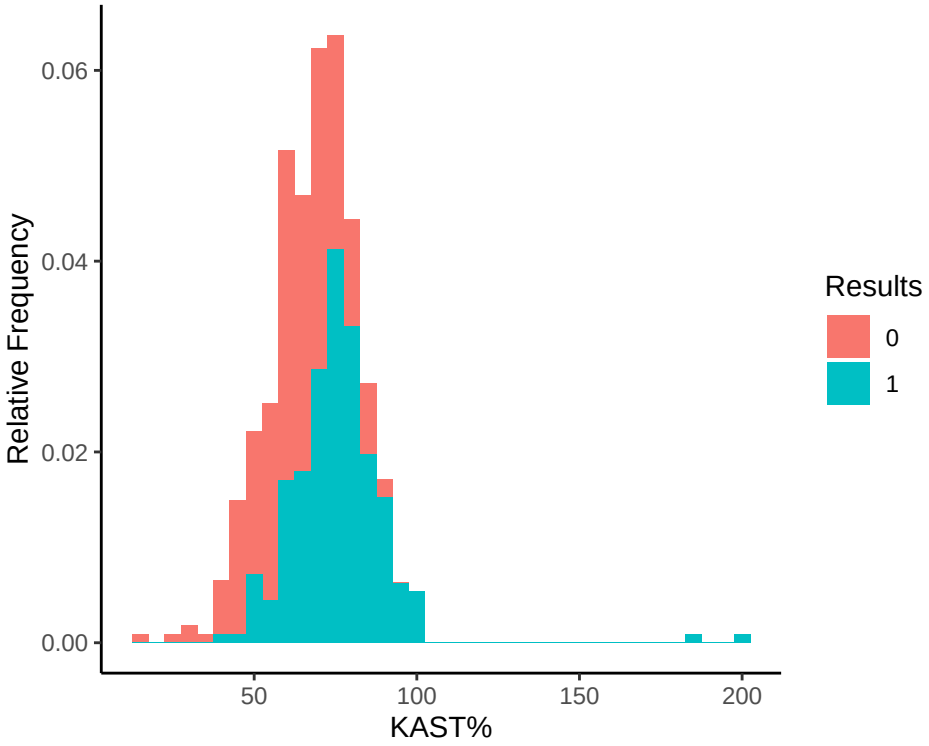


There's bit more wins than losses when the ACS of the player is over about 190.

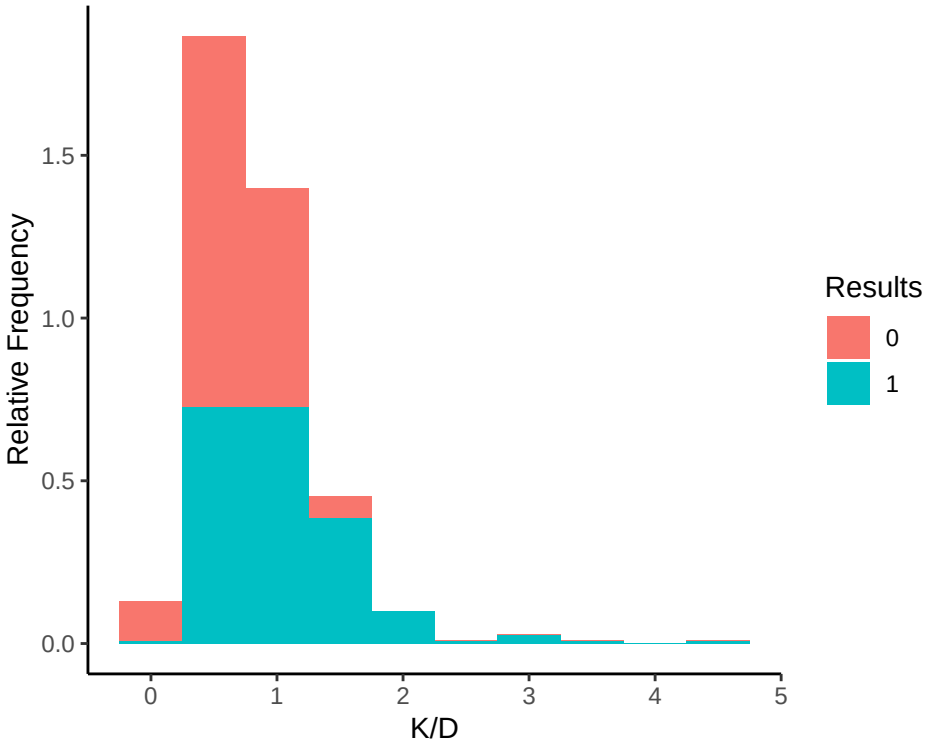


Here, we can see a more distinguishable difference between the 2 classes in the response when the player dies

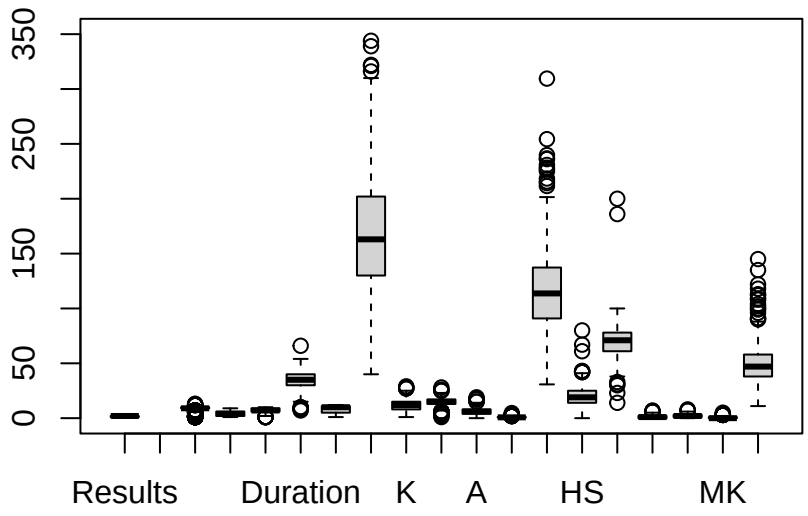
less in the match. The wins are much greater than losses when the death of player in a match is less than about 12.



There seems to be more wins than losses as the values of KAST increases.



There are no losses when the K/D of the play is greater than 2.



There are also outliers in many of the explanatory variables.

Prediction Models

Logistic Regression

First, let's try to do some predictions using a basic logistic regression model.

```
w= c(1,2,6,7,8,11,12,14,17,19) #strange choice of numbers due to the near zero frequency of some agents
log_misclass_rate <- vector(mode = "numeric", length =length(w))
log_auc <- vector(mode = "numeric", length =length(w))
N = nrow(df2)
n = length(df2)
for (i in seq_along(w)){
set.seed(w[i])
  index <- sample(1:N, size = N*.8, replace = F)
  train_x <- df2[index, 2:n]
  test_x <- df2[-index, 2:n]
  train_y <- df2[index, 1]
  test_y <- df2[-index, 1]
  train <- df2[index,]
  test <- df2[-index,]
logitmodel <- glm(Results ~., family = binomial(link = logit), data = train)
  pred_logitmod <- predict(logitmodel, newdata = test, type = "response")
```

```
  log_predicted_class <- ifelse(pred_logitmod > 0.5, 1, 0)
  log_predicted_class <- factor(log_predicted_class, levels = c(0, 1))

  log_error <- table(actual = test_y$Results, predicted = log_predicted_class)

  log_misclass_rate[i] <- 1-sum(diag(log_error))/sum(log_error)
  log_auc[i] <- auc(roc(log_predicted_class, test_y$Results))
}
results = data.frame(Log_Err = log_misclass_rate, Log_auc = log_auc)
summary_results=describe(results)
knitr::kable(results)
```

Log_Err	Log_auc
0.2272727	0.7636364
0.2613636	0.7386364
0.2386364	0.7613636
0.2727273	0.7255814
0.2613636	0.7386364
0.2613636	0.7386128
0.2840909	0.7310526
0.2500000	0.7505176
0.2727273	0.7271318
0.2613636	0.7417582

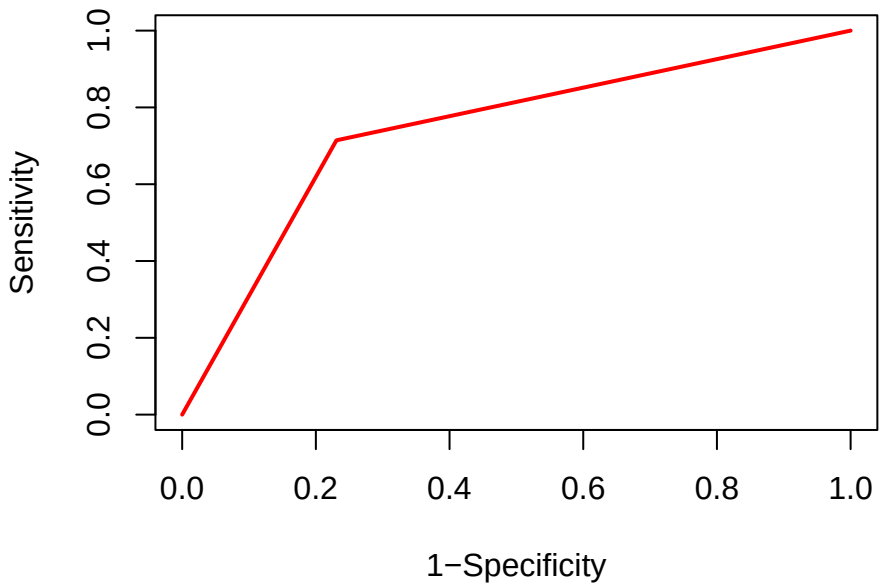
```
knitr::kable(round(summary_results, digits = 4))
```

	vars	n	mean	sd	median	trimmed	mad	min	max	range	skew	kurtosis	se
Log_Err	1	10	0.2591	0.0168	0.2614	0.2599	0.0168	0.2273	0.2841	0.0568	-0.4406	-0.8925	0.0053
Log_auc	2	10	0.7417	0.0132	0.7386	0.7410	0.0142	0.7256	0.7636	0.0381	0.4519	-1.2908	0.0042

From the logistic regression model, we can see that the mean misclassification rate of 10 iterations is about 25.91% and the mean area under the curve (AUC) score is about 0.7417. Below is also the confusion matrix and the plot of the ROC curve for the logistic regression model.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 30 14
##          1  9 35
##
##                Accuracy : 0.7386
##                  95% CI : (0.6341, 0.8266)
##     No Information Rate : 0.5568
##     P-Value [Acc > NIR] : 0.0003307
##
##                   Kappa : 0.4773
```

```
##
##  Mcnemar's Test P-Value : 0.4042485
##
##             Sensitivity : 0.7143
##             Specificity : 0.7692
##          Pos Pred Value : 0.7955
##          Neg Pred Value : 0.6818
##               Precision : 0.7955
##                  Recall : 0.7143
##                      F1 : 0.7527
##              Prevalence : 0.5568
##          Detection Rate : 0.3977
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.7418
##
##        'Positive' Class : 1
##
```



Now, let's check if there's multicollinearity and using deviance to check variable importance in the logistic model.

```
##                           GVIF Df GVIF^(1/(2*Df))
## Date                  3.037701  1        1.742900
## Agent                11.519616 12        1.107201
## Map                   6.894420  8        1.128252
## Placement            51.852940  9        1.245275
## Duration              6.742016  1        2.596539
## AverageRankofMatch    9.638334 11        1.108479
```

```
## AverageCombatScore    21.056702  1        4.588758
## K                     50.597019  1        7.113158
## D                      8.955863  1        2.992635
## A                      2.239509  1        1.496499
## K_D                   44.862140  1        6.697921
## AverageDamagePerRound 29.515384  1        5.432806
## HS                     1.633467  1        1.278072
## KAST                   2.288224  1        1.512688
## FirstKill              2.485275  1        1.576476
## FirstDeath             1.514941  1        1.230830
## MK                     2.774560  1        1.665701
## Econ                  24.197967  1        4.919143
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: Results
##
## Terms added sequentially (first to last)
##
##
##                       Df Deviance Resid. Df Resid. Dev
## NULL                                    348     483.81
## Date                   1    0.173       347     483.64
## Agent                 12   20.547       335     463.09
## Map                    8    3.864       327     459.23
## Placement              9   14.961       318     444.27
## Duration               1    0.398       317     443.87
## AverageRankofMatch    11   15.038       306     428.83
## AverageCombatScore     1    0.838       305     427.99
## K                      1   20.266       304     407.73
## D                      1  134.922       303     272.81
## A                      1    8.197       302     264.61
## K_D                    1   20.147       301     244.46
## AverageDamagePerRound  1    0.016       300     244.45
## HS                     1    0.417       299     244.03
## KAST                   1    8.692       298     235.34
## FirstKill              1    0.827       297     234.51
## FirstDeath             1    2.342       296     232.17
## MK                     1    0.015       295     232.15
## Econ                   1    5.663       294     226.49
```

There is some strong multicollinearity issues with some VIF values much greater than 10. This is expected since the "K_D" variable is the ratio of "K" and "D" and some of the explanatory variables used to calculate another. In the Deviance table, variables with larger deviance have more importance in the in the model. "D" seems to be most important in predicting the outcome of the match in logistic regression.

Not shown but stepwise variable selection was performed on the logistic model to try and relieve multicollinearity issues. The reduced model performed slightly better with misclass rate of 22% and AUC score 0.78. However, there were still high multicollinearity according to the VIF values.

Ridge Regression

Let's see if the predictions can be improved by relieving some multicollinearity problems using ridge regression. Dummy variables were created for the multilevel categorical variables since ridge regression can only handle numeric explanatory variables.

When $\alpha = 0$, then the ridge regression is used. The cv.glmnet() automatically fits the ridge regression using the most optimal lambda based on cross-validation.*Note: There's a strange choice of numbers for the seed due to the near zero frequency of some agent. The seed choice were based on each train and test sets have all levels of each categorical variables.*

```
w= c(1,2,6,7,8,11,12,14,17,19)
ridge_misclass_rate <- vector(mode = "numeric", length =length(w))
ridge_auc <- vector(mode = "numeric", length =length(w))
x = model.matrix(Results~., df3)[, -1]
y = df3$Results
N = nrow(df3)
for (i in seq_along(w)){
set.seed(w[i])
train <- sample(1:N, N*.8, replace = F)
test <- (-train)
y.test <- y[test]
train.df = data.frame(df3[train,])
test.df = data.frame(df3[test,])
ridge.mod = cv.glmnet(x[train,],y[train],family = "binomial", alpha=0)
ridge.pred = predict(ridge.mod, newx=x[test,], type = "response")

ridge_pred_class <- ifelse(ridge.pred > 0.5, 1, 0)
ridge_pred_class <- factor(ridge_pred_class, levels = c(0, 1))

  ridge_error <- table(actual = test_y$Results, predicted = ridge_pred_class)

  ridge_misclass_rate[i] <- 1-sum(diag(ridge_error))/sum(ridge_error)
  ridge_auc[i] <- auc(roc(ridge_pred_class, test_y$Results))
}
results = data.frame(Ridge_Err = ridge_misclass_rate, Ridge_auc = ridge_auc)
summary_results=describe(results)
knitr::kable(results)
```

Ridge_Err	Ridge_auc
0.4545455	0.5604396
0.5000000	0.5065411
0.5227273	0.4756672
0.6022727	0.4120879
0.4318182	0.5677656
0.3750000	0.6292517
0.4545455	0.5421245
0.3750000	0.6266353
0.4545455	0.5421245
0.2954545	0.7111460

```
knitr::kable(round(summary_results, digits = 4))
```

	vars	n	mean	sd	median	trimmed	mad	min	max	range	skew	kurtosis	se
Ridge_Err	1	10	0.4466	0.0859	0.4545	0.4460	0.0842	0.2955	0.6023	0.3068	0.0283	-0.7880	0.0272
Ridge_auc	2	10	0.5574	0.0847	0.5513	0.5563	0.0890	0.4121	0.7111	0.2991	0.0970	-0.8511	0.0268

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 30 17
##          1  9 32
##
##                Accuracy : 0.7045
##                  95% CI : (0.5978, 0.7971)
##     No Information Rate : 0.5568
##     P-Value [Acc > NIR] : 0.003207
##
##                   Kappa : 0.4136
##
##  Mcnemar's Test P-Value : 0.169811
##
##             Sensitivity : 0.6531
##             Specificity : 0.7692
##          Pos Pred Value : 0.7805
##          Neg Pred Value : 0.6383
##               Precision : 0.7805
##                  Recall : 0.6531
##                      F1 : 0.7111
##              Prevalence : 0.5568
##          Detection Rate : 0.3636
##    Detection Prevalence : 0.4659
##       Balanced Accuracy : 0.7111
##
##        'Positive' Class : 1
##
```

Even though some multicollinearity might be relieved, ridge regression actually does worse than logistic regression with average misclass rate of 44.66% and average AUC score 0.5574! Looking at the 10 iterations, there seems to be a bit of variability between each iteration of the AUC and misclass rate too.

Principle Component Analysis (PCA)

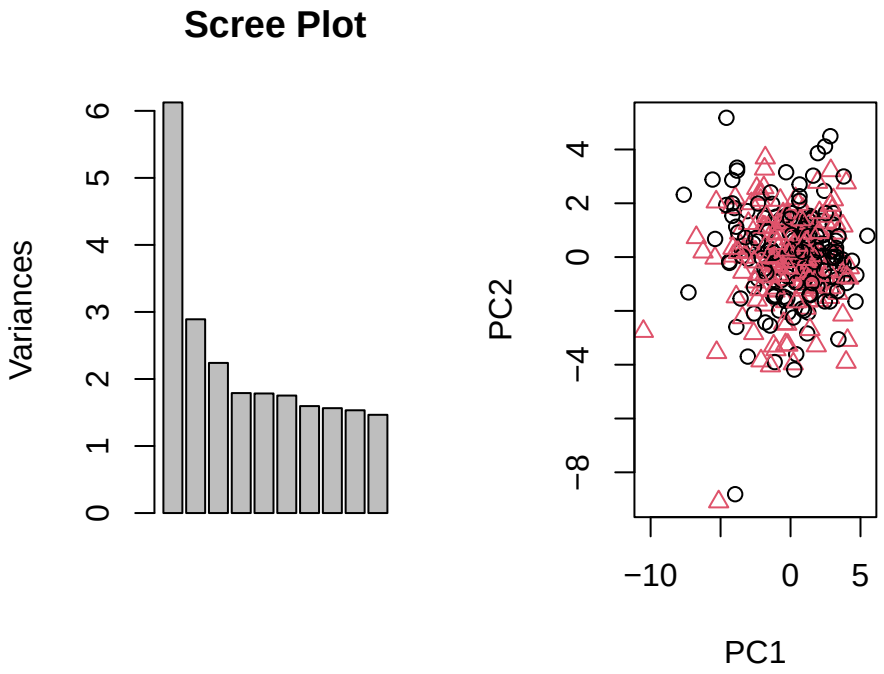
PCA can be used as a dimension reduction technique for supervised methods like in this case. PCA can reduce dimensions by extracting a linear combination of optimally-weighted observed variables.

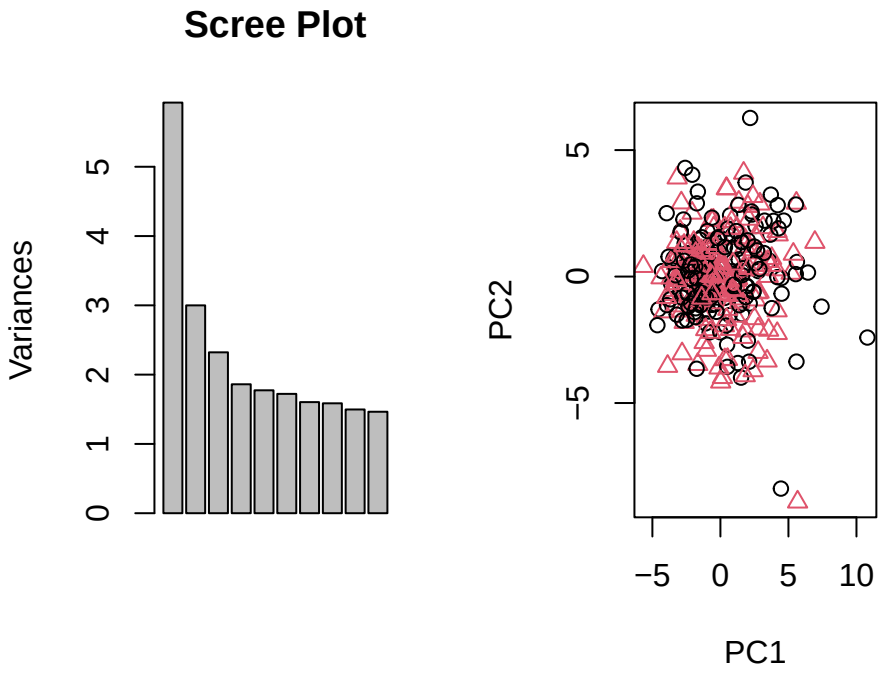
PCA is effective in dealing with multicollinearity because all the principal components are orthogonal and uncorrelated to each other. Then, a classification algorithm can be applied to the PCs instead of the original input variables.

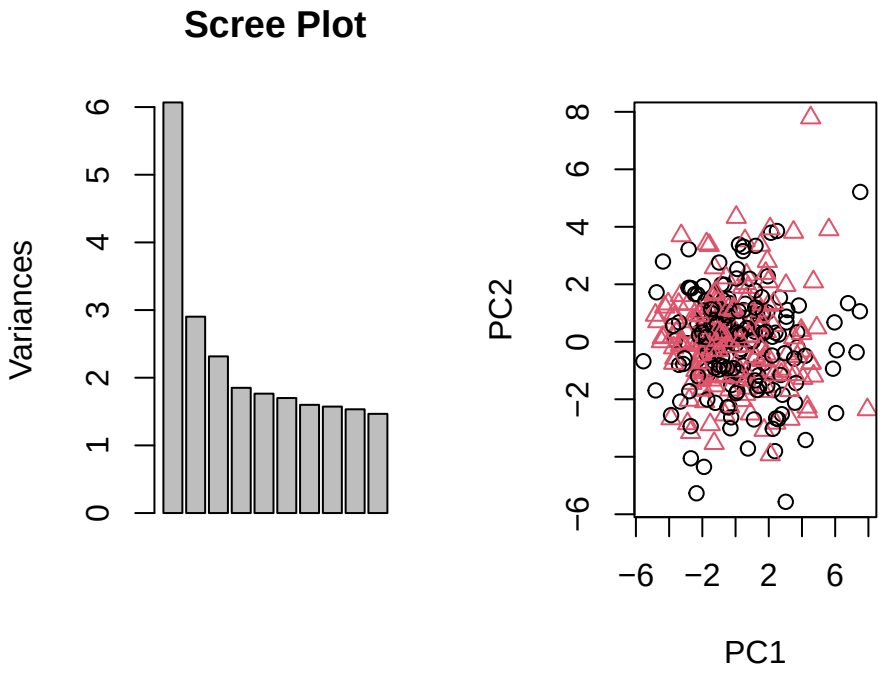
The variable "Date" has been removed for the PCA since PCA can only handle numeric variables. The dummy variables for the multilevel categorical variables were used here as well.

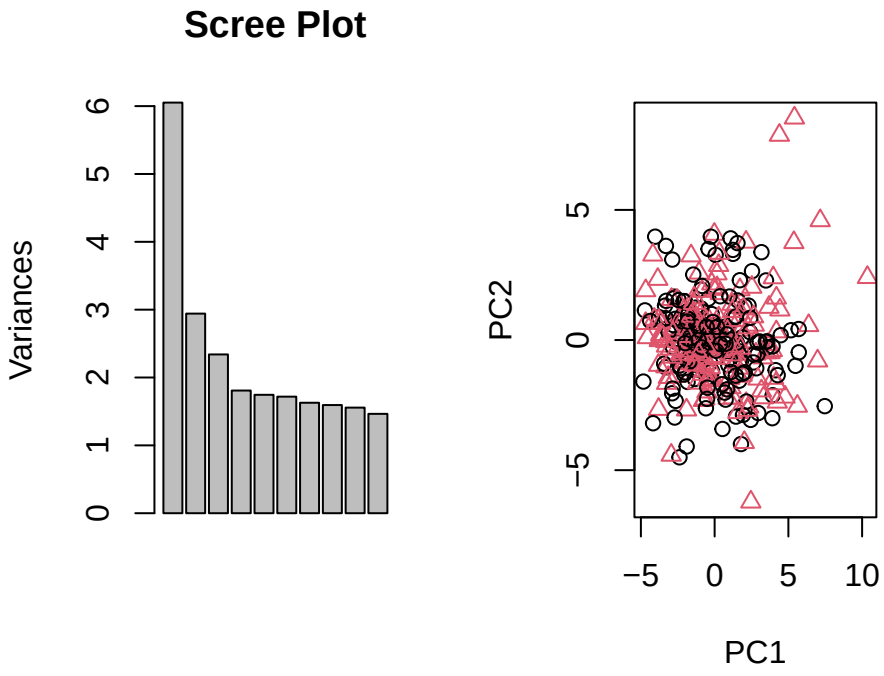
PCA was done on the standardized dataset because we are assuming no knowledge of which explanatory variables are important. The first PC shows the direction that maximizes the variance in that direction. The second PC is orthogonal to the first PC and is the one that maximizes the variance under the condition that is assigned to it.

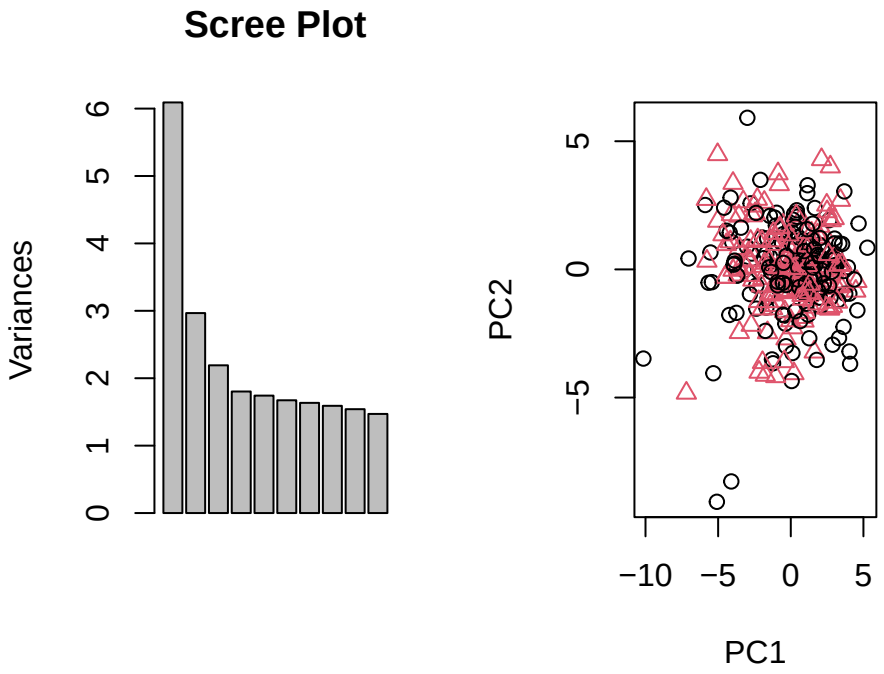
```
w= c(1,2,6,7,8,11,12,14,17,19)
pca_log_misclass_rate <- vector(mode = "numeric", length =length(w))
pca_log_auc <- vector(mode = "numeric", length =length(w))
n = length(df3)
N = nrow(df3)
for (i in seq_along(w)){
set.seed(w[i])
train <- sample(1:N, N*.8, replace = F)
test <- (-train)
y = df3$Results
y.test <- y[test]
x.test = df3[test, 3:n]
x.train = df3[train, 3:n]
y.train = y[train]
train.df = data.frame(df3[train,])
test.df = data.frame(df3[test,])

pca <- prcomp(x.train,scale=T)
ResultsPCH <- as.numeric(factor(df3$Results))
par(mfrow=c(1,2))
screeplot(pca,main="Scree Plot")
plot(pca$x[,1],pca$x[,2],xlab="PC1", ylab="PC2",type="p",col=ResultsPCH,pch=ResultsPCH)

train_pcs <- pca$x[, 1:2]
test_pcs <- predict(pca, newdata = x.test)[, 1:2]
pca.train = data.frame(Results = y.train, train_pcs)
pca.test = data.frame(Results = y.test, test_pcs)
pca.logitmodel <- glm(Results ~., family = binomial(link = logit), data = pca.train)
pca_pred_logitmod <- predict(pca.logitmodel, newdata = pca.test, type = "response")

  pca_log_predicted_class <- ifelse(pca_pred_logitmod > 0.5, 1, 0)
  pca_log_predicted_class <- factor(pca_log_predicted_class, levels = c(0, 1))
  pca_log_auc[i] <- auc(roc(pca_log_predicted_class, y.test))
pca_log_misclass_rate[i] = 1- confusionMatrix(pca_log_predicted_class, y.test, mode = "everything", posi
}
```

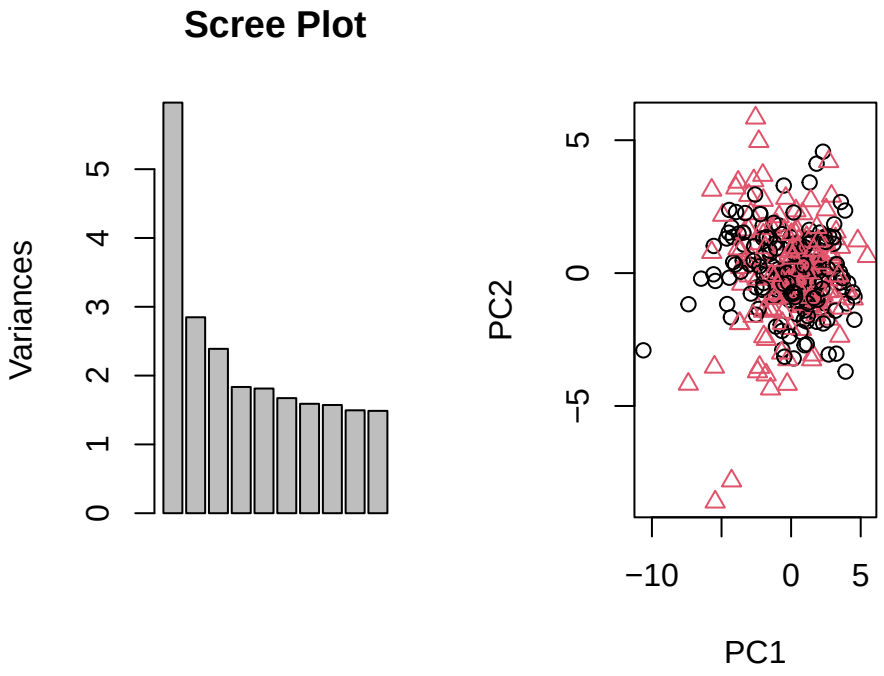

Scree Plot
Variances
0
1
2
3
4
5
6
PC2
−8
−4
0
2
4
−10
0
5
PC1

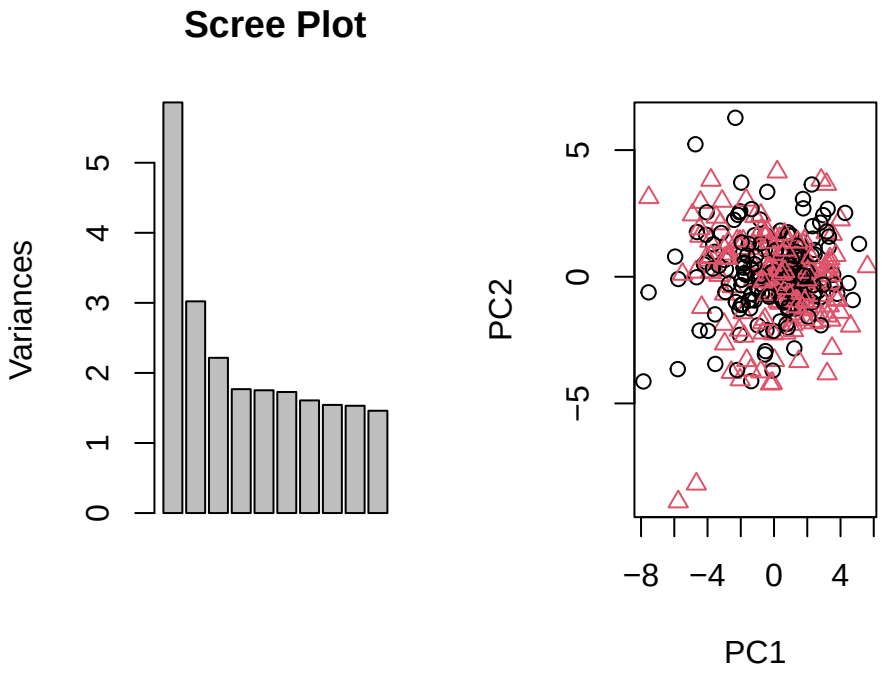

Scree Plot
Variances
0
1
2
3
4
5
PC2
−5
0
5
−5
0
5
10
PC1

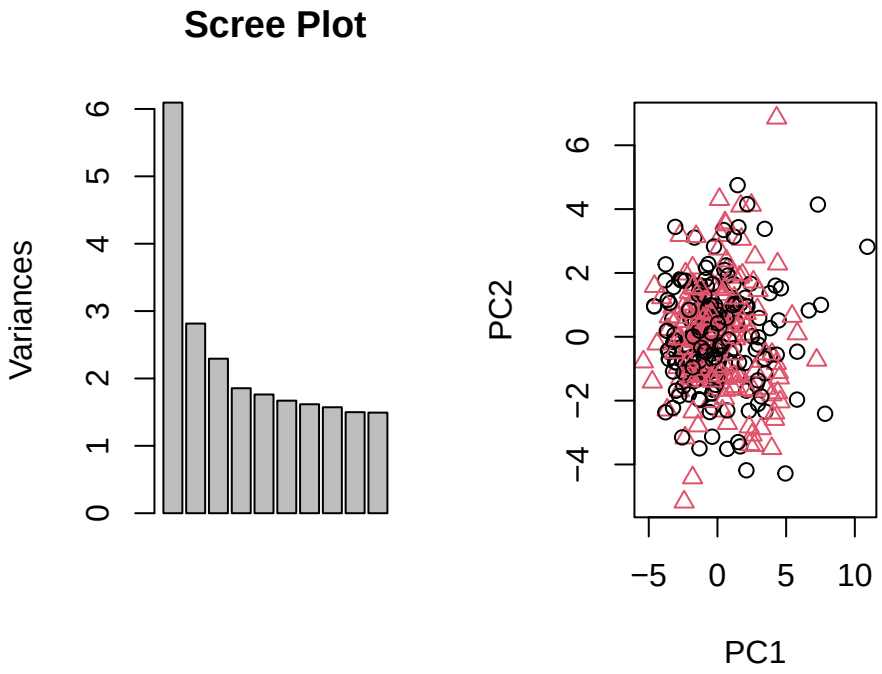

Scree Plot
Variances
0
1
2
3
4
5
6
PC2
−6
−2
2
6
8
−6
−2
2
6
PC1

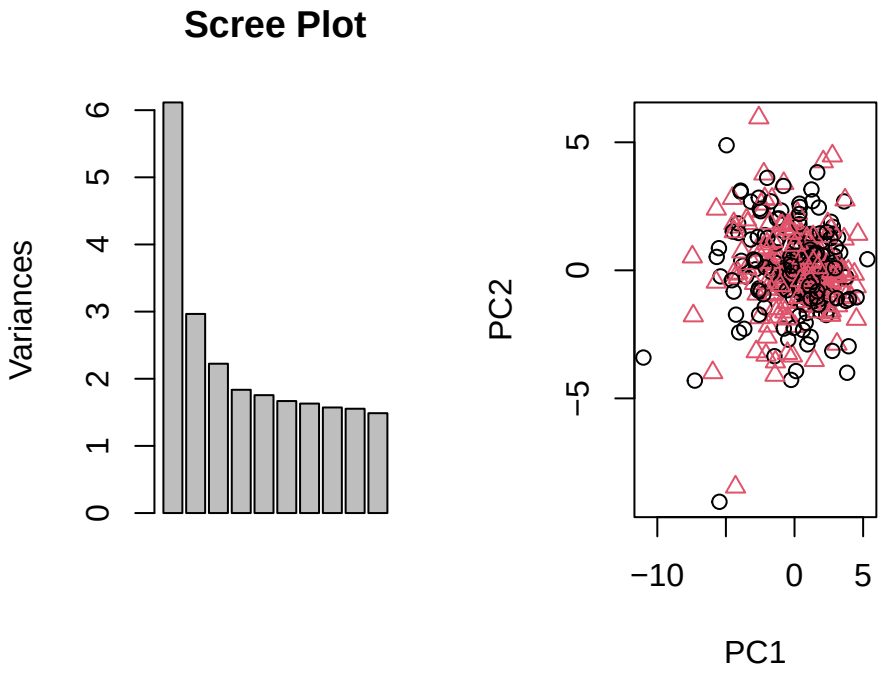

Scree Plot
Variances
0
1
2
3
4
5
6
PC2
−5
0
5
−5
0
5
10
PC1

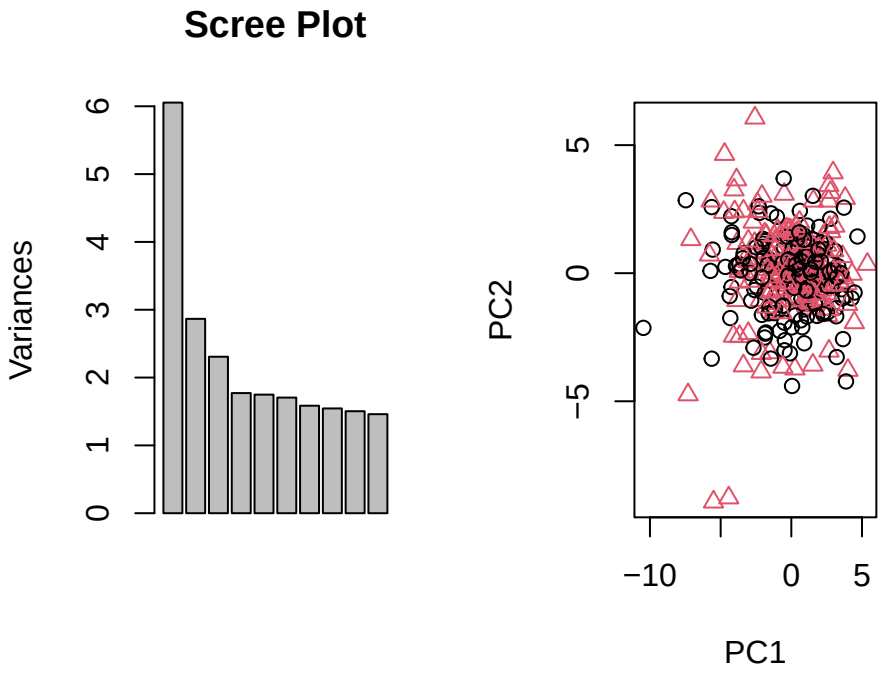

Scree Plot
Variances
0
1
2
3
4
5
6
PC2
5
0
−5
−10
−5
0
5
PC1


Scree Plot
Variances
0
1
2
3
4
5
PC2
5
0
−5
−10
0
5
PC1


Scree Plot
Variances
0
1
2
3
4
5
PC2
5
0
−5
−8
−4
0
4
PC1


Scree Plot
Variances
0
1
2
3
4
5
6
PC2
6
4
2
0
−2
−4
−5
0
5
10
PC1






```
results = data.frame(PCA_Err = pca_log_misclass_rate, PCA_auc = pca_log_auc)
summary_results=describe(results)
knitr::kable(results)
```

PCA_Err	PCA_auc
0.3636364	0.6363636
0.4431818	0.5568182
0.4431818	0.5568182
0.4090909	0.5870801
0.4545455	0.5454545
0.3863636	0.6138716
0.3181818	0.6884211
0.3522727	0.6444099
0.4318182	0.5653747
0.3068182	0.7009419

```
knitr::kable(round(summary_results, digits = 4))
```

	vars	n	mean	sd	median	trimmed	mad	min	max	range	skew	kurtosis	se
PCA_Err	1	10	0.3909	0.0539	0.3977	0.3935	0.0674	0.3068	0.4545	0.1477	-0.2808	-1.6363	0.0170
PCA_auc	2	10	0.6096	0.0564	0.6005	0.6061	0.0647	0.5455	0.7009	0.1555	0.3681	-1.5511	0.0178

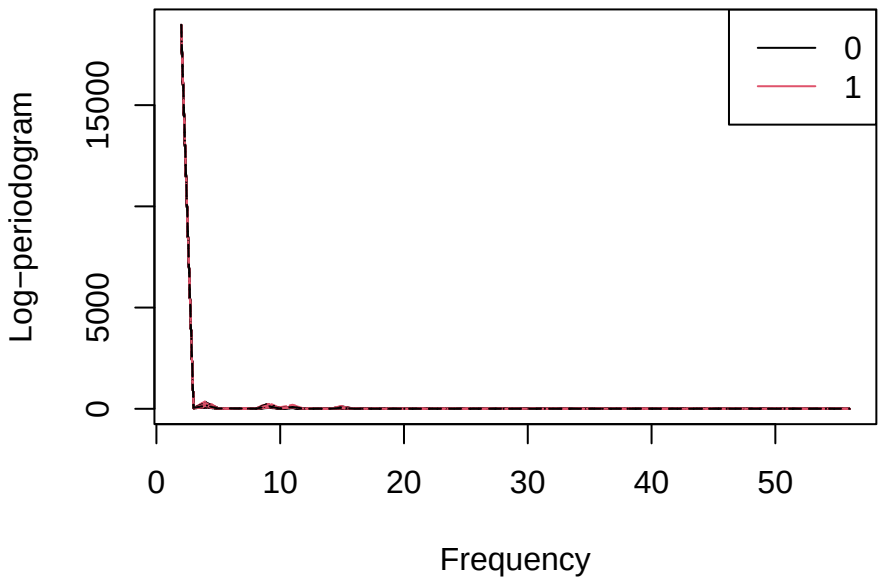
From all the scree plot, the optimal number of PCs is 1.

The logistic regression done on the PC actually gave the worse results out than the default logistics regression, but better than ridge regression with misclass rate of about 39.09% and AUC score of 0.6096! However, using PCs resolves the multicollinearity issues in logistic regression.

Data Visualization

First, let's visualize the raw data.

```
col=as.numeric(df3$Results)
matplot(2:n,t(x[1:200,]),col=col[1:200],type="l",
xlab="Frequency",ylab="Log-periodogram")
legend("topright",legend=levels(df3$Results),lty=1,col=1:2)
```



We can see that the two classes of the response variable are not easily distinguishable using the raw data.

Linear Discriminant Analysis (LDA)

LDA can be used to classify the boundary between the two classes in the binary response variable, Results, based on the other quantitative explanatory variables. The multi-level categorical variables were replaced with dummy variables here too.

```
# df3 = df2[,-2]
w= c(1,2,6,7,8,11,12,14,17,19)
lda_misclass_rate <- vector(mode = "numeric", length =length(w))
lda_auc <- vector(mode = "numeric", length =length(w))
n = length(df3)
N = nrow(df3)
for (i in seq_along(w)){
set.seed(w[i])
x.lda = as.matrix(df3[,3:length(df3)])
id <- sample(1:N, N*.8, replace = F)

fit.lda = lda(x.lda[id,], y[id])
lda.pred<-predict(fit.lda,x.lda[-id,])
table(lda.pred$class,y[-id])
lda_misclass_rate[i] = 1-sum(lda.pred$class==y[-id])/length(y[-id])
lda_auc[i] <- auc(roc(lda.pred$class,y[-id]))
}
results = data.frame(LDA_Err = lda_misclass_rate, LDA_auc = lda_auc)
summary_results=describe(results)
knitr::kable(results)
```

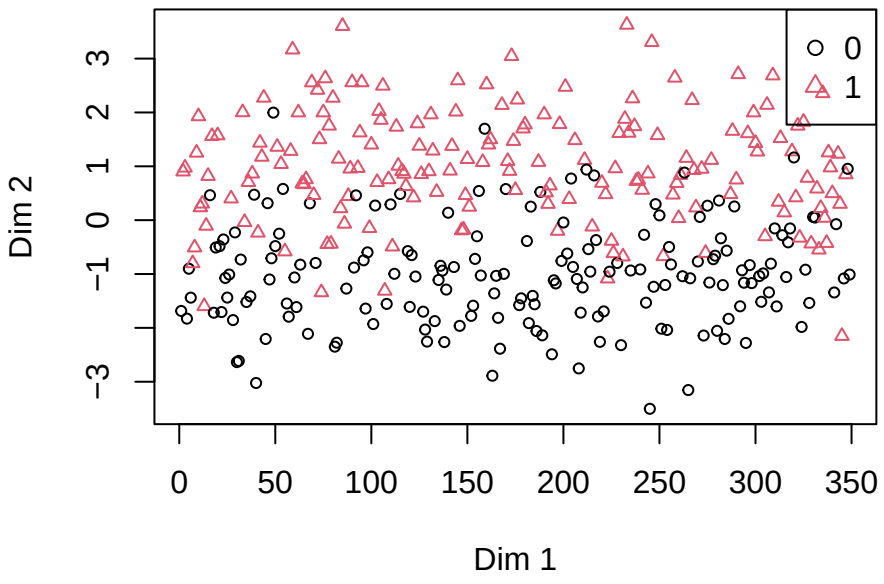
LDA_Err	LDA_auc
0.2159091	0.7787879
0.2500000	0.7500000
0.2500000	0.7500000
0.2613636	0.7356589
0.2500000	0.7500000
0.2500000	0.7494824
0.2954545	0.7242105
0.2272727	0.7701863
0.2500000	0.7498708
0.3181818	0.6855050

```
knitr::kable(round(summary_results, digits = 4))
```

	vars	n	mean	sd	median	trimmed	mad	min	max	range	skew	kurtosis	se
LDA_Err	1	10	0.2568	0.0299	0.2500	0.2543	0.0084	0.2159	0.3182	0.1023	0.7204	-0.5073	0.0095
LDA_auc	2	10	0.7444	0.0257	0.7499	0.7474	0.0109	0.6855	0.7788	0.0933	-0.9061	0.2101	0.0081

The average misclass rate for the LDA is 25.68% and AUC score is 0.7444, which is on par with the default logistic regression model which is what we discussed in class.

```
lda.var <- predict(fit.lda,dimen=2)$x
plot(lda.var,xlab="Dim 1",ylab="Dim 2",col=col[id],pch=col[id],cex=0.7)
legend("topright",legend=levels(y),pch=1:2,col=1:2,cex=1)
```



LDA distinguishes the two classes pretty well with some minor overlap.

Random Forest (RF)

RF is one of the best models for predictions when handling big and complex data for both regression and classification problems. Let's compare the unoptimized RF to the optimize RF to demonstrate the importance of optimizing the hyperparameters.

RF_Err	RF_auc
0.1931818	0.7969697
0.2500000	0.7500000
0.2613636	0.7386364
0.2840909	0.7160207
0.3181818	0.6818182
0.2613636	0.7406832
0.3409091	0.6652632
0.3295455	0.6692547
0.2954545	0.7033592
0.2840909	0.7161172

	vars	n	mean	sd	median	trimmed	mad	min	max	range	skew	kurtosis	se
RF_Err	1	10	0.2818	0.0435	0.2841	0.2855	0.0421	0.1932	0.3409	0.1477	-0.4631	-0.6927	0.0137
RF_auc	2	10	0.7178	0.0407	0.7161	0.7145	0.0434	0.6653	0.7970	0.1317	0.3583	-0.9645	0.0129

Below is the confusion matrix for the unoptimized RF. The average misclass rate of the RF is 28.18% and the AUC score is 0.7178. The unoptimized RF model performs better than the ridge regression and PCA.

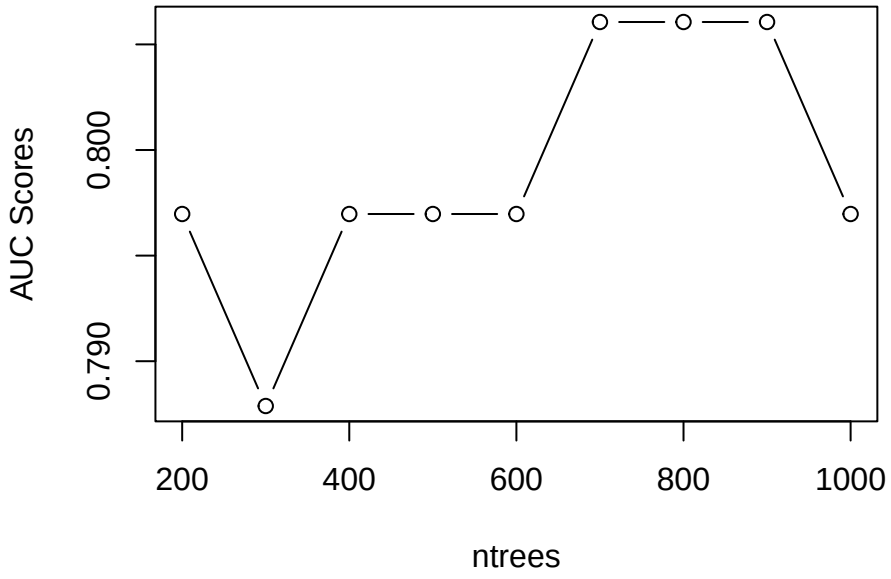
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 28 14
##          1 11 35
##
##                Accuracy : 0.7159
##                  95% CI : (0.6098, 0.807)
##     No Information Rate : 0.5568
##     P-Value [Acc > NIR] : 0.001585
##
##                   Kappa : 0.4289
##
##  Mcnemar's Test P-Value : 0.689157
##
##             Sensitivity : 0.7143
##             Specificity : 0.7179
##          Pos Pred Value : 0.7609
##          Neg Pred Value : 0.6667
##               Precision : 0.7609
```

```
##                  Recall : 0.7143
##                      F1 : 0.7368
##              Prevalence : 0.5568
##          Detection Rate : 0.3977
##    Detection Prevalence : 0.5227
##       Balanced Accuracy : 0.7161
##
##        'Positive' Class : 1
##
```

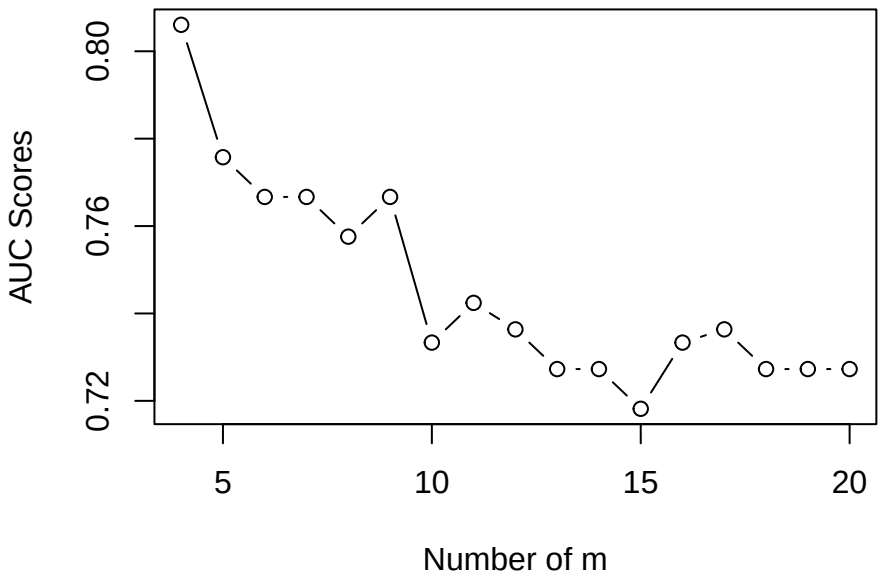
Optimizing Random Forest

Plots will be shown to display the best values for each hyperparameter based on AUC scores.

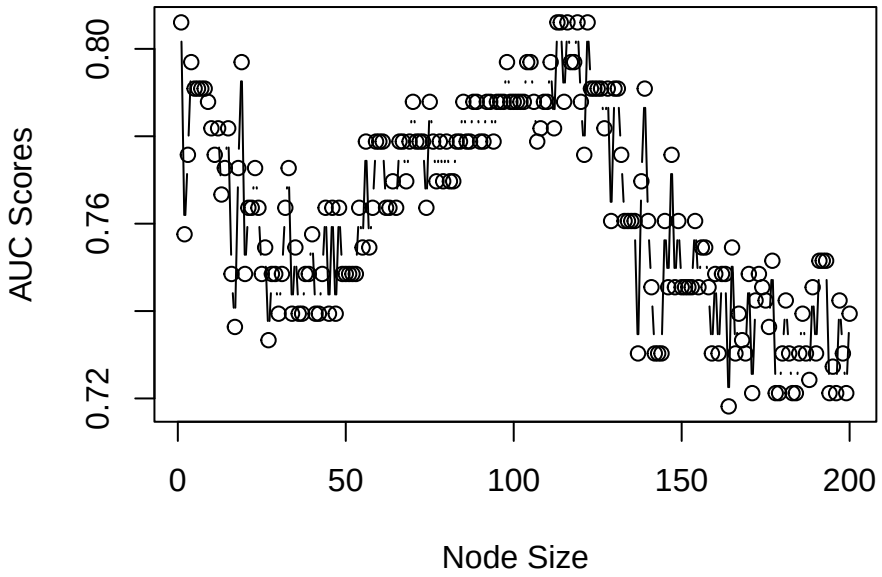
First, we want to tune the number of trees, m, and depth to maximize area under curve (AUC) score.



We will now find the optimal m using the optimal number of trees based on AUC score. As we know, too large m will have high correlation and low bias.



Now, find the optimal tree depth using the optimal number of trees and optimal m based on AUC score. The smaller the node size, the deeper the tree.



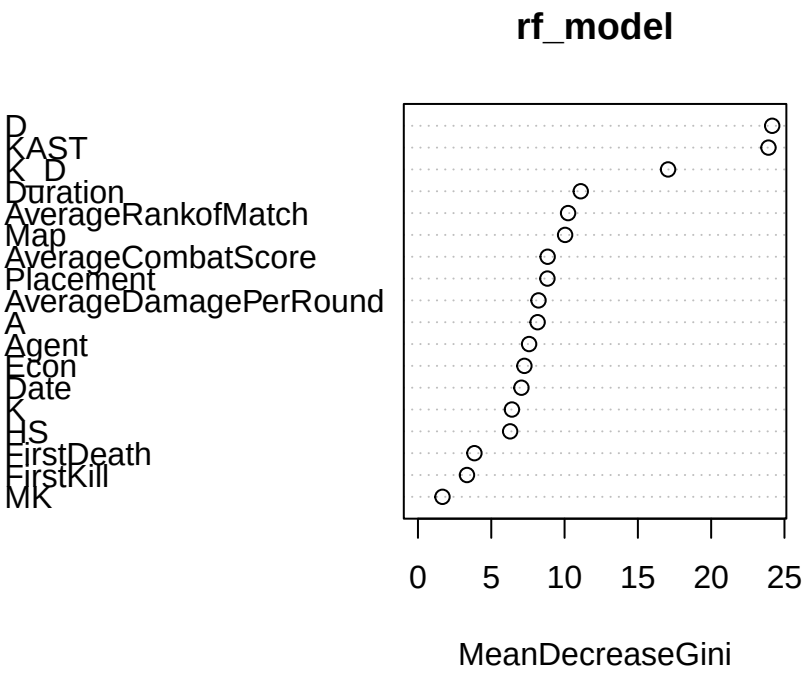
RF_Err	RF_auc
0.1818182	0.8060606
0.2500000	0.7500000
0.2500000	0.7500000
0.2840909	0.7160207
0.3068182	0.6931818
0.2727273	0.7298137
0.3295455	0.6752632
0.3181818	0.6811594
0.2954545	0.7028424
0.2954545	0.7059131

	vars	n	mean	sd	median	trimmed	mad	min	max	range	skew	kurtosis	se
RF_Err	1	10	0.2784	0.0429	0.2898	0.2841	0.0337	0.1818	0.3295	0.1477	-0.9064	-0.1020	0.0136
RF_auc	2	10	0.7210	0.0395	0.7110	0.7161	0.0361	0.6753	0.8061	0.1308	0.7699	-0.4744	0.0125

As we can see, the optimized RF does the best so far with misclass rate of 27.84% and AUC score 0.7210. The most important variables in the optimized RF prediction model is "D", followed by "KAST" and "K_D". Both the logistic and RF model agree that the variable "D" contribute most in predicting the response and is most related to the response.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 28 15
##          1 11 34
##
##                Accuracy : 0.7045
##                  95% CI : (0.5978, 0.7971)
##     No Information Rate : 0.5568
##     P-Value [Acc > NIR] : 0.003207
##
##                   Kappa : 0.4076
##
##  Mcnemar's Test P-Value : 0.556298
##
##             Sensitivity : 0.6939
##             Specificity : 0.7179
##          Pos Pred Value : 0.7556
##          Neg Pred Value : 0.6512
##               Precision : 0.7556
##                  Recall : 0.6939
##                      F1 : 0.7234
##              Prevalence : 0.5568
##          Detection Rate : 0.3864
##    Detection Prevalence : 0.5114
##       Balanced Accuracy : 0.7059
##
```

```
##        'Positive' Class : 1
##
```



MARS (Multivariate Adaptive Regression Splines), CART-related Method

MARS is a flexible and adaptive technique well suited for complex data. MARS is a generalization of stepwise linear regression and a modification of the CART method to improve its performance. It can also automatically model complex interactions between variables and can handle both numerical and categorical predictors. A restriction of MARS on the formation of model terms is each input can appear at most once in a product. This prevents the formation of higher-order powers of an input, which increase or decrease too sharply near the boundaries of the feature space.

Dummy variables must be used for multilevel categorical variables.

```
N = nrow(df3)
n = length(df3)
w= c(1,2,6,7,8,11,12,14,17,19)
mars_misclass_rate <- vector(mode = "numeric", length =length(w))
mars_auc <- vector(mode = "numeric", length =length(w))
for (i in seq_along(w)){
set.seed(w[i])
train <- sample(1:N, N*.8, replace = F)
test <- (-train)
y = df3$Results
y.test <- y[test]
x.test = df3[test, 3:n]
```

```
x.train = df3[train, 3:n]
y.train = y[train]
train.df = data.frame(df3[train,])
test.df = data.frame(df3[test,])
 fit1 <- mars(x.train,y.train)

 pred1 <- predict(fit1,x.test)
 temp1 <- as.numeric(pred1>=0.5)
 res1 <- table(temp1,y.test)
mars_misclass_rate[i] = 1-sum(diag(res1))/sum(res1) #misclassification rate
mars_auc[i] <- auc(roc(factor(temp1),y.test))
}
results = data.frame(MARS_Err = mars_misclass_rate, MARS_auc = mars_auc)
summary_results=describe(results)
knitr::kable(results)
```

MARS_Err	MARS_auc
0.2159091	0.7909091
0.2840909	0.7159091
0.3068182	0.6931818
0.3409091	0.6542636
0.2727273	0.7272727
0.2613636	0.7375776
0.2840909	0.7184211
0.2500000	0.7536232
0.2613636	0.7377261
0.2727273	0.7289377

```
knitr::kable(round(summary_results, digits = 4))
```

	vars	n	mean	sd	median	trimmed	mad	min	max	range	skew	kurtosis	se
MARS_Err1	1	10	0.2750	0.0334	0.2727	0.2741	0.0168	0.2159	0.3409	0.1250	0.2503	-0.3734	0.0106
MARS_auc2	2	10	0.7258	0.0360	0.7281	0.7266	0.0162	0.6543	0.7909	0.1366	-0.2163	-0.2538	0.0114

The MARS misclass rate is 27.50% and AUC score 0.7258 which is on par with the optimized random forest!

Comparing MARS vs RF

I wanted to compare these two models specifically because they are the most robust when there are low frequency of some levels in the multilevel categorical variables and can handle zero frequencies in some levels.

```
w=20
mars_misclass_rate <- vector(mode = "numeric", length = w)
mars_auc <- vector(mode = "numeric", length = w)
rf_misclass_rate <- vector(mode = "numeric", length = w)
```

```
rf_auc <- vector(mode = "numeric", length = w)
for (i in 1:w) {
  set.seed(i)
  n=length(df2)
  index <- sample(1:N, size = N*.8, replace = F)
  train_x <- df2[index, 2:n]
  test_x <- df2[-index, 2:n]
  train_y <- df2[index, 1]
  test_y <- df2[-index, 1]
  train <- df2[index,]
  test <- df2[-index,]
rf_model <- randomForest(Results~.,data = train, xtest = test_x,
                     ytest=test_y$Results, mtry = optimal.m, nodesize = optimal.depth,
                     ntree=optimal.ntrees, keep.forest = TRUE)
  rf_predicted_class <- rf_model$test$predicted
  rf_error <- table(actual = test_y$Results, predicted = rf_predicted_class)
  rf_misclass_rate[i] <- 1-sum(diag(rf_error))/sum(rf_error)
  rf_auc[i] <- auc(roc(rf_predicted_class,test_y$Results))
}

for (i in 1:w){
  set.seed(i)
  n = length(df3)
  N = nrow(df3)
  train <- sample(1:N, N*.8, replace = F)
  test <- (-train)
  y = df3$Results
  y.test <- y[test]
  x.test = df3[test, 3:n]
  x.train = df3[train, 3:n]
  y.train = y[train]
  train.df = data.frame(df3[train,])
  test.df = data.frame(df3[test,])
  mars.fit <- mars(x.train,y.train)

  pred1 <- predict(mars.fit,x.test)
 temp1 <- as.numeric(pred1>=0.5)
 res1 <- table(temp1,y.test)
mars_misclass_rate[i] = 1-sum(diag(res1))/sum(res1) #misclassification rate
mars_auc[i] <- auc(roc(factor(temp1),y.test))
}
```

	vars	n	mean	sd	median	trimmed	mad	min	max	range	skew	kurtosis	se
MARS_Err1	1	20	0.2710	0.0364	0.2670	0.2706	0.0253	0.1932	0.3409	0.1477	0.0925	-0.1668	0.0081
RF_Err	2	20	0.2773	0.0386	0.2841	0.2798	0.0421	0.1818	0.3295	0.1477	-0.5807	-0.3466	0.0086

	vars	n	mean	sd	median	trimmed	mad	min	max	range	skew	kurtosis	se
MARS_auc	1	20	0.7306	0.0366	0.7333	0.7303	0.0310	0.6543	0.8085	0.1543	0.0183	-0.1912	0.0082
RF_auc	2	20	0.7245	0.0359	0.7220	0.7220	0.0421	0.6753	0.8061	0.1308	0.4549	-0.8168	0.0080

Surprisingly, MARS performed better than the optimized RF across the average of 20 iterations!

Note: Not shown but MARS performed slightly better when the model is additive with no interactions. Also, using the PC instead of the input variables in the RF and MARS model made both model substantially worse.

Conclusion

BELOW ARE THE RESULTS THAT COMPARES EACH MODEL TO THE ONE ITERATION WITH SEED SET TO 1.

Table 17: One Iteration Comparison

Model	AUC.Scores	Misclass.Rate
Logistic Regression	0.760	23%
Ridge Regression	0.560	45%
PCA	0.636	36%
LDA	0.779	21.6%
Unopt RF	0.797	19.3%
Opt RF	0.806	18%
MARS	0.791	21.6%

BELOW ARE THE RESULTS THAT COMPARES AVERAGE AUC SCORE AND MISCLASS RATE OF EACH MODEL ACROSS TEN ITERATIONS.

Table 18: Ten Iterations Comparison

Model	AUC.Scores	Misclass.Rate
Logistic Regression	0.7417	25.91%
Ridge Regression	0.5574	44.66%
PCA	0.6096	39.09%
LDA	0.7444	25.68%
Unopt RF	0.7178	28.18%
Opt RF	0.7210	27.84%
MARS	0.7258	27.5%

Based on the comparison of various models such as logistic regression, PCA, LDA, RF, and MARS, we found that they all had similar prediction accuracies in terms of AUC scores and misclassification rates, but ridge regression performed the worst out of all the models. In comparing RF to MARS, the optimized RF performed the best, but after multiple iterations, MARS showed slightly better performance than RF. Although RF is known for its robustness, especially with complex data and multicollinearity, MARS was able to outperform it in this scenario.

Both MARS and RF offer good accuracy and robustness. However, personally, RF tends to be more interpretable, and it doesn't require manual creation of dummy variables for categorical data, unlike MARS. Also, one could argue that RF should be optimized to produce the best accuracy.

It is noteworthy how significantly the comparison results vary between single iterations and the average of multiple iterations for each model. This substantial variation is likely attributed to the imbalances observed in certain levels of the categorical variables. Among the models examined, LDA and Logistic Regression demonstrate superior performance. However, LDA exhibits lower sensitivity to multicollinearity compared to logistic regression, albeit with reduced interpretability. Despite this, LDA, logistic regression, RF, and MARS emerge as reliable options for predicting match outcomes within this dataset.

An intriguing finding emerged regarding the significance of the "D" variable, representing deaths, in the prediction model. This finding is notable given that numerous players gauge their progress based on metrics such as Average Combat Score (ACS) or Kill/Death ratio (K/D). This suggests a potential area of improvement where players could benefit from emphasizing the reduction of deaths during matches.

In conclusion, the analysis reveals that certain player statistics play a crucial role in accurately predicting match outcomes. Despite Valorant being a team-oriented game, these findings underscore the individual impact of a player in determining whether a match culminates in victory or defeat.

Future enhancements to this model could involve integrating more granular statistics pertaining to a player's performance within each round, such as their weapon selections, utilization of in-game utilities, and other tactical decisions. By capturing these nuanced gameplay elements, the model could potentially forecast match outcomes with greater precision, even preempting results before or early in the match.

References

Li, Bin. "Principle Component Analysis." 2023.

Li, Bin. "Lab Notes: Ridge Regression Examples in R." 2024.

Li, Bin. "Phoneme Classification Example." 2024.

Li, Bin. "Two CART-Related Methods - PRIM and MARS." 2024.