Final Project

EDA on the Chess Games dataset

By: Abhijeet Vichare

Date: 12/05/2022

Table of Contents

1. Introduction
2. Data
3. Data Cleaning
4. Exploratory Data Analysis
5. Model Building
6. Conclusion
7. Data Wrangling
   a. Data importing
   b. Checking the data types of variables and performing necessary steps to change the data type based on relevancy.
   c. Univariate Statistics of the variables
   d. Dealing with null values
   e. Outlier analysis
8. Data Visualization
   a. Bivariate analysis
   b. Correlation analysis
9. Feature
   a. Chi square test
   b. ANOVA test
   c. Correlation test
10. Linear Regression
   a. Linear Regression assumption validation
   b. Model training
   c. Hyperparameter tuning
   d. Regularization
   e. Result visualization ## 1. Introduction

We have used the [chess dataset]{https://www.kaggle.com/datasets/datasnaek/chess} from kaggle which contains the data of chess games played by different players. The dataset contains 20058 rows and 16 columns and its collected from Lichess.org. A short description of the dataset:

1. id: Unique ID for the game
2. rated: Indicates whether the game was rated or unrated
3. created_at: Timestamp of the game creation
4. last_move_at: Timestamp of the last move
5. turns: Number of turns in the game
6. victory_status: Indicates the status of the game (win, draw, outoftime, resign, mate)
7. winner: Indicates the winner of the game (white, black, draw)
8. increment_code: Increment code

9. white_id: Unique ID of the white player
10. white_rating: Rating of the white player
11. black_id: Unique ID of the black player
12. black_rating: Rating of the black player
13. moves: Moves in the game
14. opening_eco: Opening ECO
15. opening_name: Opening name
16. opening_ply: Opening ply

2. Data

2.1 Importing the libraries

```
# install.packages(c("assertive.data","plotly","tidyverse", "ggplot2", "dplyr", "ggcorrplot", "corrplot
```

```
library(ggplot2)
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
library(caret)
```

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

```
library(plotly)
```

```
##
## Attaching package: 'plotly'

## The following object is masked from 'package:ggplot2':
##
##     last_plot

## The following object is masked from 'package:stats':
##
##     filter

## The following object is masked from 'package:graphics':
##
##     layout
```

```
library(assertive.data)
source("model_training.R")
```

```
##
## Call:  glm(formula = winner ~ ., family = "binomial", data = train)
##
## Coefficients:
## (Intercept)  rating_diff
##    0.070573     0.003486
##
## Degrees of Freedom: 5907 Total (i.e. Null);  5906 Residual
## Null Deviance:       8176
## Residual Deviance: 7344  AIC: 7348
##
## Call:
## glm(formula = winner ~ ., family = "binomial", data = train)
##
```

```
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6876  -1.1090   0.4832   1.0654   2.9009
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) 0.0705734  0.0279195   2.528   0.0115 *
## rating_diff 0.0034858  0.0001413  24.672   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 8175.6  on 5907  degrees of freedom
## Residual deviance: 7343.6  on 5906  degrees of freedom
## AIC: 7347.6
##
## Number of Fisher Scoring iterations: 4
##
##
##     FALSE TRUE
##   0  1029  162
##   1   829  511
## [1] 0.6084552
##
## Call:  glm(formula = winner ~ ., family = "binomial", data = train)
##
## Coefficients:
##  (Intercept)   rating_diff  black_rating
##     0.375824      0.003604     -0.000199
##
## Degrees of Freedom: 5907 Total (i.e. Null);  5905 Residual
## Null Deviance:       8177
## Residual Deviance: 7263  AIC: 7269
##
## Call:
## glm(formula = winner ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7248  -1.0968   0.4597   1.0592   2.7636
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)   0.3758243  0.1675160   2.244   0.0249 *
## rating_diff   0.0036039  0.0001533  23.510   <2e-16 ***
## black_rating -0.0001990  0.0001048  -1.900   0.0575 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 8177.3  on 5907  degrees of freedom
## Residual deviance: 7263.3  on 5905  degrees of freedom
```

```
## AIC: 7269.3
##
## Number of Fisher Scoring iterations: 4
##
##
##     FALSE TRUE
##   0  1003  179
##   1   847  502
## [1] 0.5946266
##
## Call:  glm(formula = winner ~ ., family = "binomial", data = train)
##
## Coefficients:
##  (Intercept)   rating_diff  white_rating
##     0.470723      0.003657     -0.000255
##
## Degrees of Freedom: 5907 Total (i.e. Null);  5905 Residual
## Null Deviance:       8175
## Residual Deviance: 7326  AIC: 7332
##
## Call:
## glm(formula = winner ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5469  -1.1047   0.4776   1.0617   2.9654
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)   0.4707228  0.1667977   2.822  0.00477 **
## rating_diff   0.0036572  0.0001524  23.998  < 2e-16 ***
## white_rating -0.0002550  0.0001042  -2.447  0.01439 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 8175.4  on 5907  degrees of freedom
## Residual deviance: 7326.5  on 5905  degrees of freedom
## AIC: 7332.5
##
## Number of Fisher Scoring iterations: 4
##
##
##     FALSE TRUE
##   0  1030  162
##   1   833  506
## [1] 0.6068748
```

```
## Warning in chisq.test(table): Chi-squared approximation may be incorrect
```

```
## Warning in chisq.test(table): Chi-squared approximation may be incorrect
```

```
##
## Call:  glm(formula = winner ~ ., family = "binomial", data = train)
```

```
##
## Coefficients:
## (Intercept)    ratedTRUE
##     0.10627      0.04199
##
## Degrees of Freedom: 5907 Total (i.e. Null);  5906 Residual
## Null Deviance:       8162
## Residual Deviance: 8162  AIC: 8166
##
## Call:
## glm(formula = winner ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.241  -1.241   1.115   1.115   1.133
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.10627    0.05364   1.981   0.0476 *
## ratedTRUE    0.04199    0.06139   0.684   0.4940
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 8162.0  on 5907  degrees of freedom
## Residual deviance: 8161.6  on 5906  degrees of freedom
## AIC: 8165.6
##
## Number of Fisher Scoring iterations: 3
##
##
##     FALSE
##   0  1248
##   1  1283
## [1] 0.4930857
```

```
## Warning in chisq.test(table): Chi-squared approximation may be incorrect
```

```
## Warning in chisq.test(table): Chi-squared approximation may be incorrect
```

```
##
## Call:  glm(formula = winner ~ ., family = "binomial", data = train)
##
## Coefficients:
## (Intercept)    ratedTRUE
##     0.09044      0.02255
##
## Degrees of Freedom: 5907 Total (i.e. Null);  5906 Residual
## Null Deviance:       8173
## Residual Deviance: 8173  AIC: 8177
##
## Call:
## glm(formula = winner ~ ., family = "binomial", data = train)
##
```

```
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.226  -1.226   1.130   1.130   1.139
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.09044    0.05405   1.673   0.0943 .
## ratedTRUE    0.02255    0.06169   0.366   0.7147
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 8173.1  on 5907  degrees of freedom
## Residual deviance: 8173.0  on 5906  degrees of freedom
## AIC: 8177
##
## Number of Fisher Scoring iterations: 3
##
##
##     FALSE
##   0  1203
##   1  1328
## [1] 0.4753062
##
## Call:  glm(formula = winner ~ ., family = "binomial", data = train)
##
## Coefficients:
## (Intercept)  rating_diff
##     0.09552      0.00362
##
## Degrees of Freedom: 5907 Total (i.e. Null);  5906 Residual
## Null Deviance:       8171
## Residual Deviance: 7296  AIC: 7300
##
## Call:
## glm(formula = winner ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7451  -1.1022   0.5065   1.0551   2.5925
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) 0.0955225  0.0280299   3.408 0.000655 ***
## rating_diff 0.0036199  0.0001434  25.240  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 8171.3  on 5907  degrees of freedom
## Residual deviance: 7295.7  on 5906  degrees of freedom
## AIC: 7299.7
```

```
##
## Number of Fisher Scoring iterations: 4
##
##
##     FALSE TRUE
##   0  1025  186
##   1   776  544
## [1] 0.6199131
```

2.2 Importing the dataset

```r
chess <- read.csv("games.csv")
```

```r
colnames(chess)
```

```
##  [1] "id"             "rated"          "created_at"     "last_move_at"
##  [5] "turns"          "victory_status" "winner"         "increment_code"
##  [9] "white_id"       "white_rating"   "black_id"       "black_rating"
## [13] "moves"          "opening_eco"    "opening_name"   "opening_ply"
```

2.3 Data Exploration

```r
#types of data
str(chess)
```

```
## 'data.frame':    20058 obs. of  16 variables:
##  $ id            : chr  "TZJHLljE" "l1NXvwaE" "mIICvQHh" "kWKvrqYL" ...
##  $ rated         : chr  "FALSE" "TRUE" "TRUE" "TRUE" ...
##  $ created_at    : num  1.5e+12 1.5e+12 1.5e+12 1.5e+12 1.5e+12 ...
##  $ last_move_at  : num  1.5e+12 1.5e+12 1.5e+12 1.5e+12 1.5e+12 ...
##  $ turns         : int  13 16 61 61 95 5 33 9 66 119 ...
##  $ victory_status: chr  "outoftime" "resign" "mate" "mate" ...
##  $ winner        : chr  "white" "black" "white" "white" ...
##  $ increment_code: chr  "15+2" "5+10" "5+10" "20+0" ...
##  $ white_id      : chr  "bourgris" "a-00" "ischia" "daniamurashov" ...
##  $ white_rating  : int  1500 1322 1496 1439 1523 1250 1520 1413 1439 1381 ...
##  $ black_id      : chr  "a-00" "skinnerua" "a-00" "adivanov2009" ...
##  $ black_rating  : int  1191 1261 1500 1454 1469 1002 1423 2108 1392 1209 ...
##  $ moves         : chr  "d4 d5 c4 c6 cxd5 e6 dxe6 fxe6 Nf3 Bb4+ Nc3 Ba5 Bf4" "d4 Nc6 e4 e5 f4 f6 dxe5
##  $ opening_eco   : chr  "D10" "B00" "C20" "D02" ...
##  $ opening_name  : chr  "Slav Defense: Exchange Variation" "Nimzowitsch Defense: Kennedy Variation"
##  $ opening_ply   : int  5 4 3 3 5 4 10 5 6 4 ...
```

```r
#summary of data
summary(chess)
```

```
##       id               rated             created_at         last_move_at
##  Length:20058       Length:20058       Min.   :1.377e+12   Min.   :1.377e+12
##  Class :character   Class :character   1st Qu.:1.478e+12   1st Qu.:1.478e+12
##  Mode  :character   Mode  :character   Median :1.496e+12   Median :1.496e+12
##                                        Mean   :1.484e+12   Mean   :1.484e+12
##                                        3rd Qu.:1.503e+12   3rd Qu.:1.503e+12
##                                        Max.   :1.504e+12   Max.   :1.504e+12
##      turns        victory_status        winner          increment_code
##  Min.   :  1.00   Length:20058       Length:20058       Length:20058
```

```
##  1st Qu.: 37.00   Class :character   Class :character   Class :character
##  Median : 55.00   Mode  :character   Mode  :character   Mode  :character
##  Mean   : 60.47
##  3rd Qu.: 79.00
##  Max.   :349.00
##    white_id          white_rating    black_id          black_rating
##  Length:20058       Min.   : 784   Length:20058       Min.   : 789
##  Class :character   1st Qu.:1398   Class :character   1st Qu.:1391
##  Mode  :character   Median :1567   Mode  :character   Median :1562
##                     Mean   :1597                      Mean   :1589
##                     3rd Qu.:1793                      3rd Qu.:1784
##                     Max.   :2700                      Max.   :2723
##     moves           opening_eco        opening_name        opening_ply
##  Length:20058       Length:20058       Length:20058       Min.   : 1.000
##  Class :character   Class :character   Class :character   1st Qu.: 3.000
##  Mode  :character   Mode  :character   Mode  :character   Median : 4.000
##                                                           Mean   : 4.817
##                                                           3rd Qu.: 6.000
##                                                           Max.   :28.000
```

```
#checking for null values
colSums(is.na(chess))
```

```
##             id          rated     created_at   last_move_at          turns
##              0              0              0              0              0
## victory_status         winner increment_code       white_id   white_rating
##              0              0              0              0              0
##       black_id   black_rating          moves    opening_eco   opening_name
##              0              0              0              0              0
##    opening_ply
##              0
```

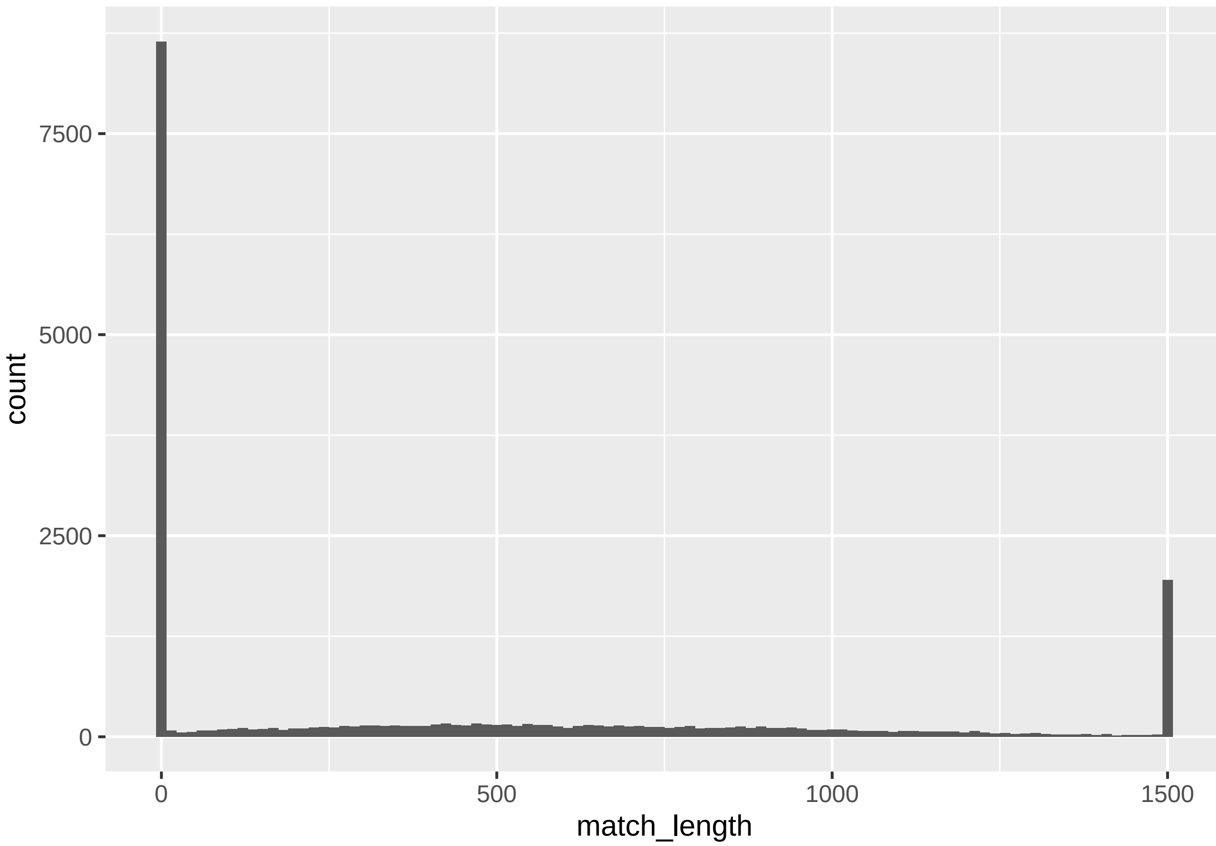
There are 20058 rows and 16 columns in the dataset. Columns like rated can be converted to boolean. We can convert the numerical created_at and last_move_at to date time format. The columns like white_id, black_id can be dropped as they are not relevant to the analysis. The column id can be dropped as it is just a unique identifier for the game. The column turns can be dropped as it is highly correlated with the column white_rating. The column victory_status can be dropped as it is highly correlated with the column winner. There are no null values in the dataset.

```
# converting rated to boolean
chess$rated <- as.logical(chess$rated)
# converting created_at and last_move_at to date time format
chess$created_at <- as.POSIXct(chess$created_at/1000, origin = "1970-01-01")
chess$last_move_at <- as.POSIXct(chess$last_move_at/1000, origin = "1970-01-01")
```

```
# difference of created_at and last_move_at
chess$match_length <- chess$last_move_at - chess$created_at
# limit the max to 2000s
chess$match_length <- ifelse(chess$match_length > 1500, 1500, chess$match_length)
```

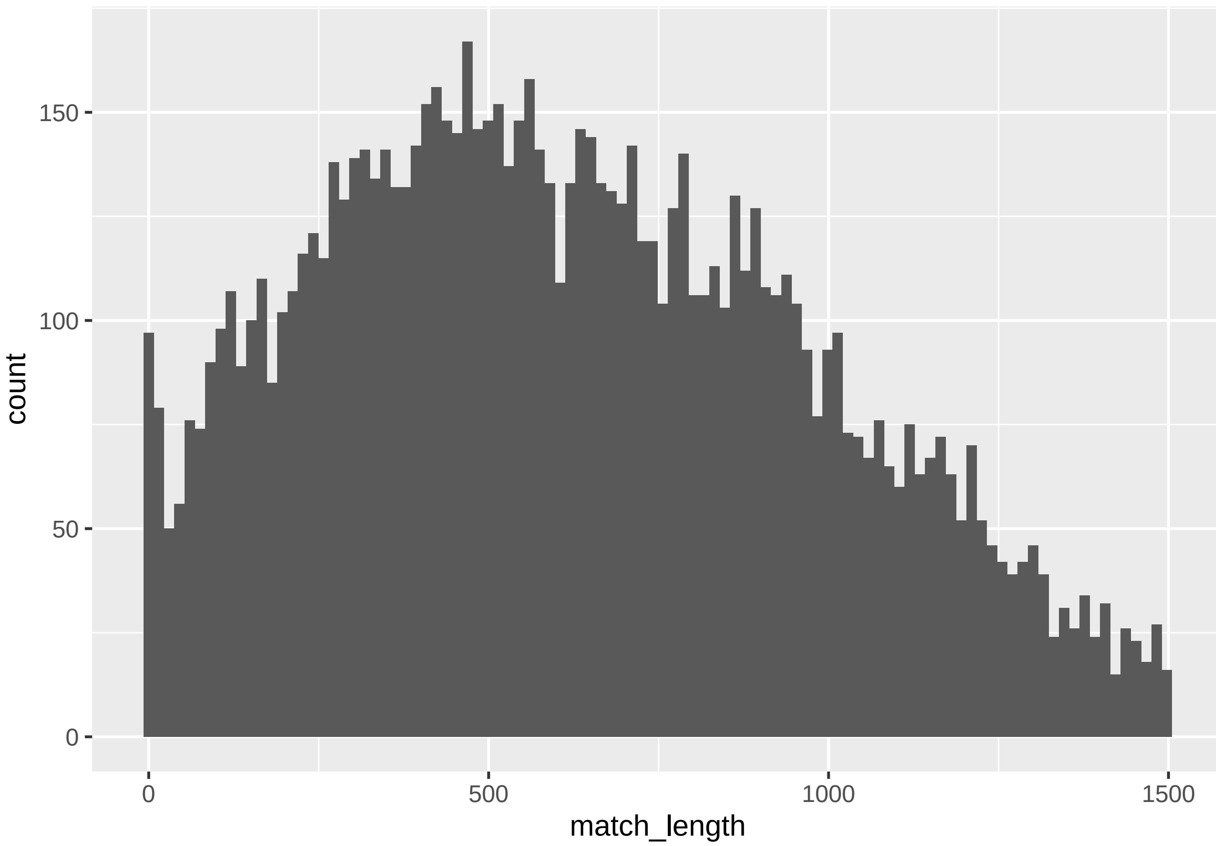
2.4 Univariate Visualization

```
# convert match_length to float and plot the histogram
chess$match_length <- as.numeric(chess$match_length)
ggplot(chess, aes(x = match_length)) + geom_histogram(bins = 100)
```

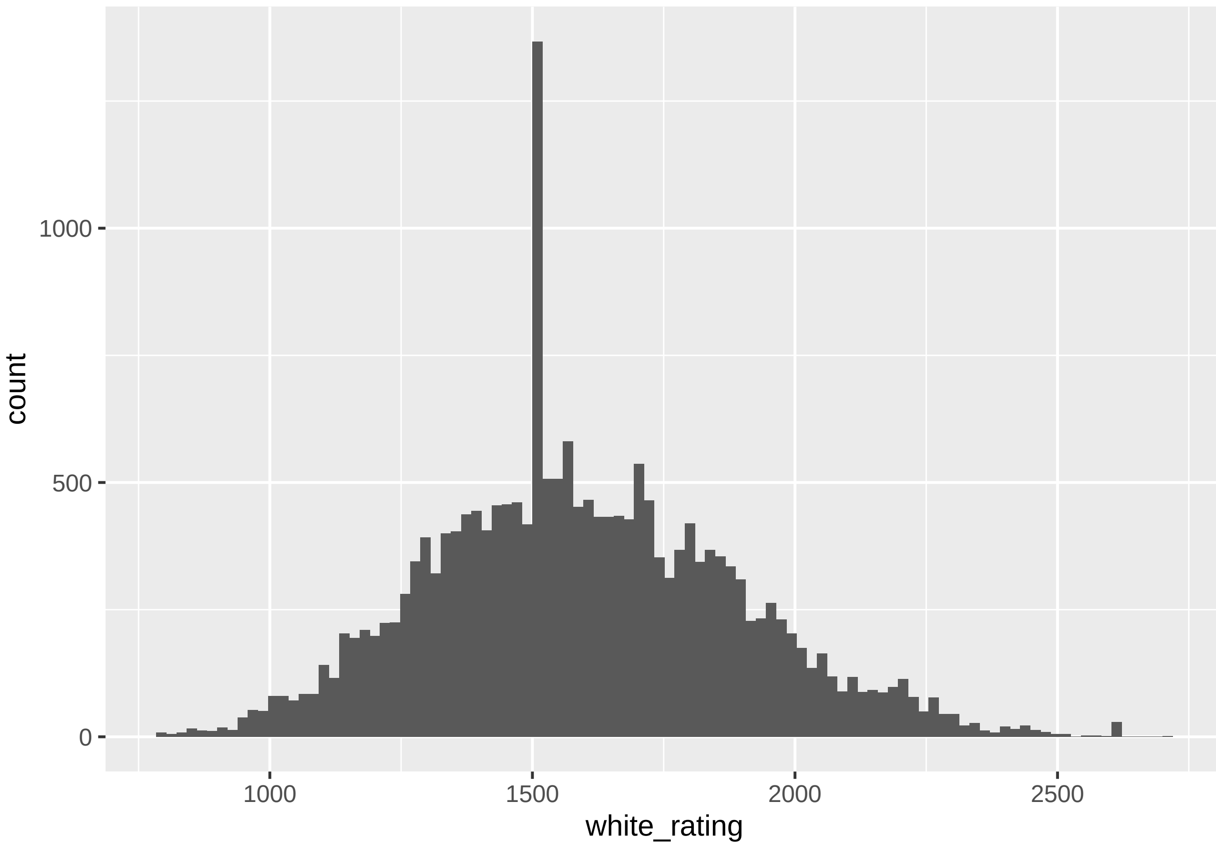


There are outliers at 0 and 1500. We can remove the outliers and plot the histogram again.

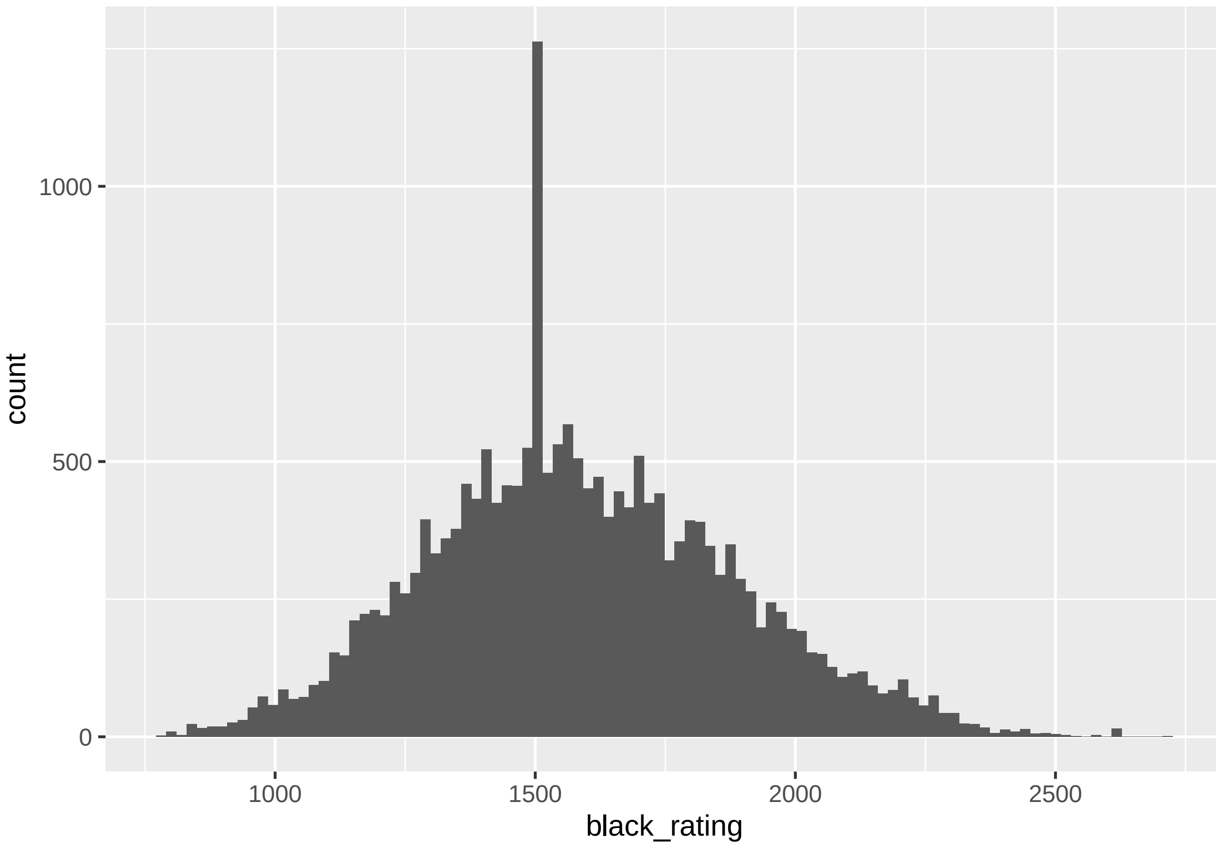
```
# remove the outliers and plot the histogram
temp <- filter(chess, match_length > 0 & match_length < 1500)
ggplot(temp, aes(x = match_length)) + geom_histogram(bins = 100)
```



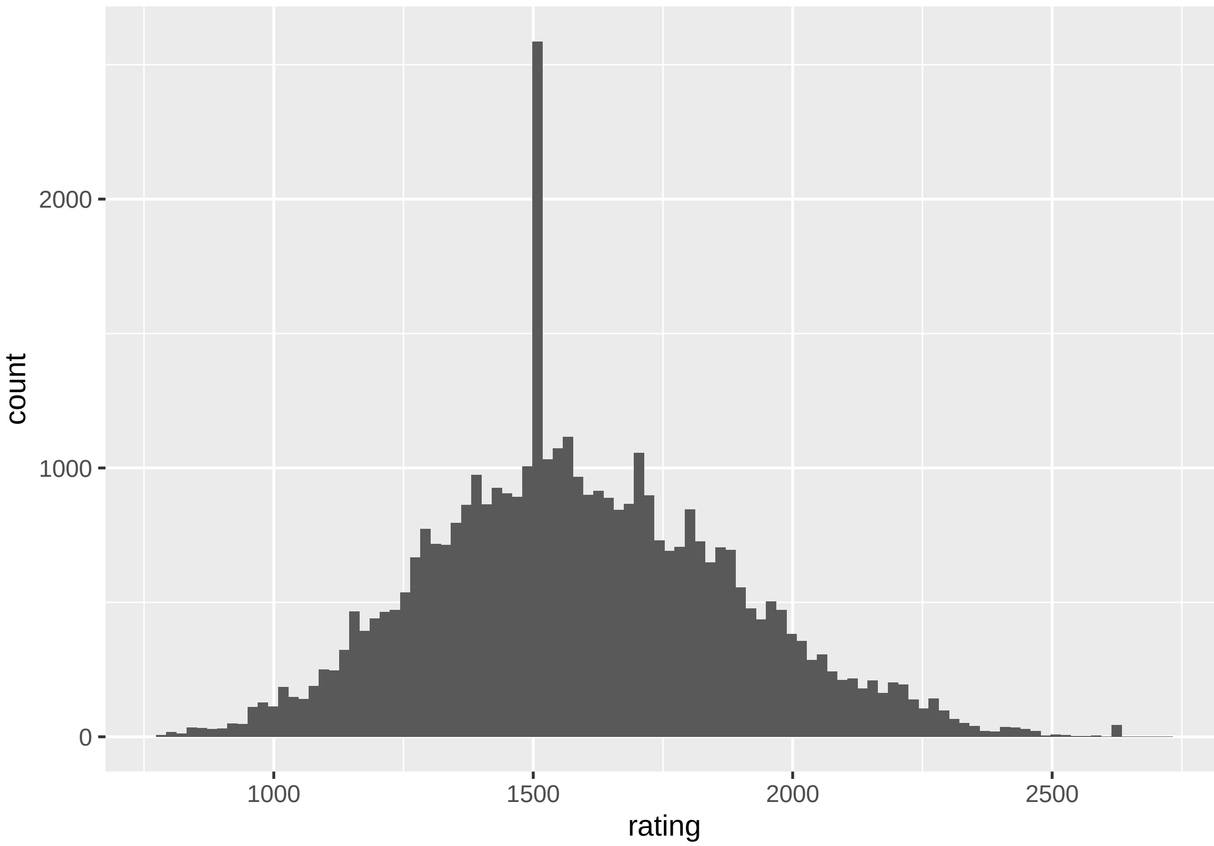

```
#histogram of white_rating
ggplot(chess, aes(x = white_rating)) + geom_histogram(bins = 100)
```




```
#histogram of black_rating
ggplot(chess, aes(x = black_rating)) + geom_histogram(bins = 100)
```




```
# combine the white_rating and black_rating into one vector and plot the histogram
rating <- c(chess$white_rating, chess$black_rating)
ggplot(data.frame(rating), aes(x = rating)) + geom_histogram(bins = 100)
```



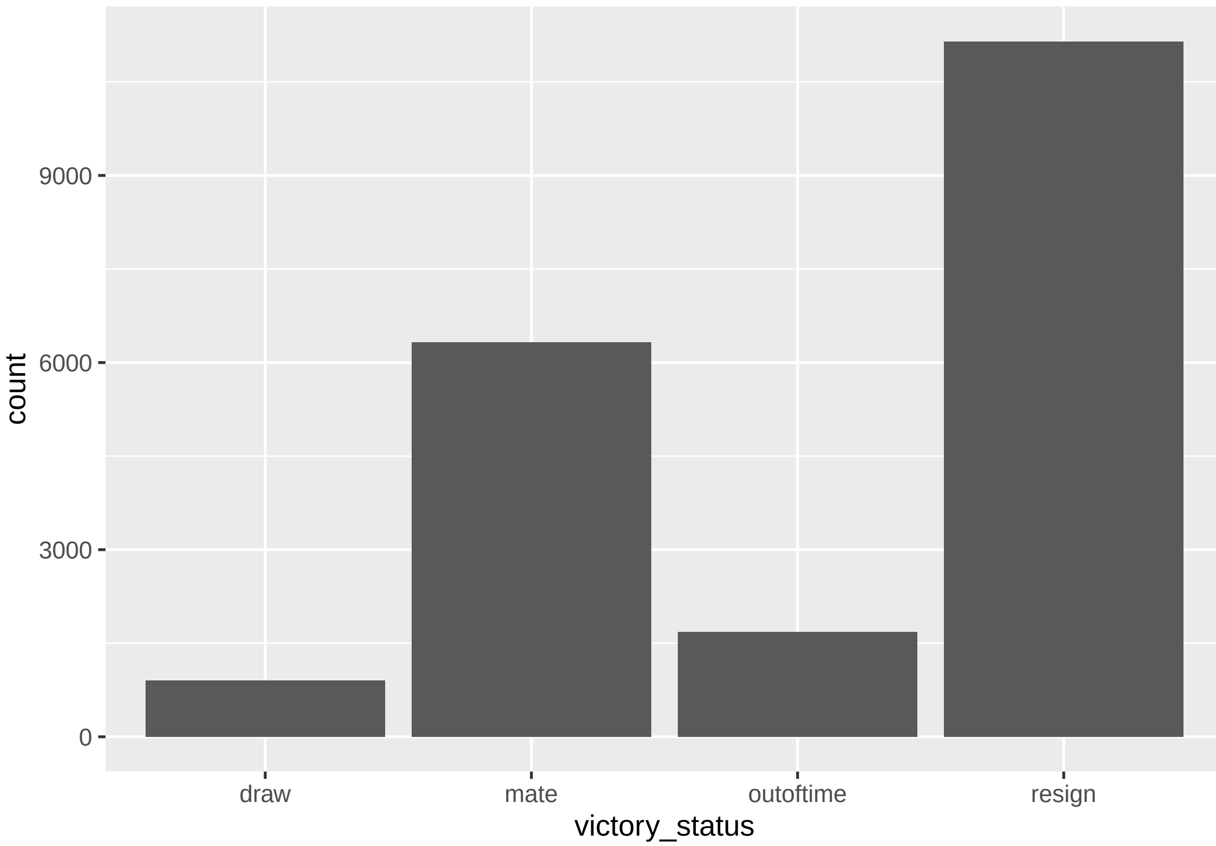

```
#value counts of rating and sort it in descending order

rating <- c(chess$white_rating, chess$black_rating)
rating <- as.data.frame(table(rating))
rating <- rating[order(-rating$Freq),]
head(rating)
```

```
##     rating Freq
## 667   1500 1609
## 567   1400  117
## 668   1501   97
## 647   1480   91
## 729   1562   88
## 703   1536   84
```

The ratings follow a normal distribution. The mean of the ratings is 1591. There is one outlier of 1500 where The most common rating is 1500.

```
# Distribution of the Victory Status
ggplot(chess, aes(x = victory_status)) + geom_bar()
```

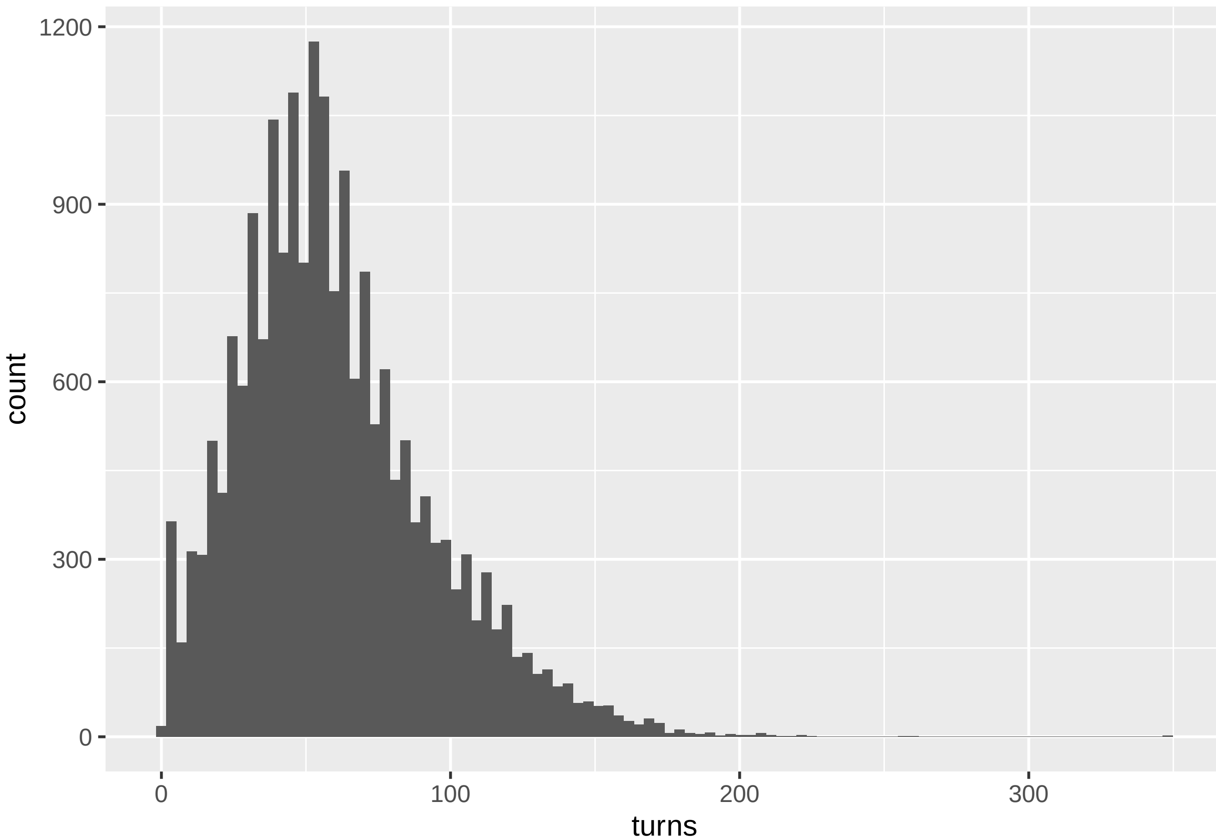



```r
#value counts as percentage of victory_status and sort it in descending order
victory_status <- as.data.frame(table(chess$victory_status))
victory_status$percent <- round(victory_status$Freq/sum(victory_status$Freq)*100, 2)
victory_status <- victory_status[order(-victory_status$percent),]
victory_status
```

```
##        Var1  Freq percent
## 4    resign 11147   55.57
## 2      mate  6325   31.53
## 3 outoftime  1680    8.38
## 1      draw   906    4.52
```

55.5% of the games are resigned and 31.53% of the games end by a mate.

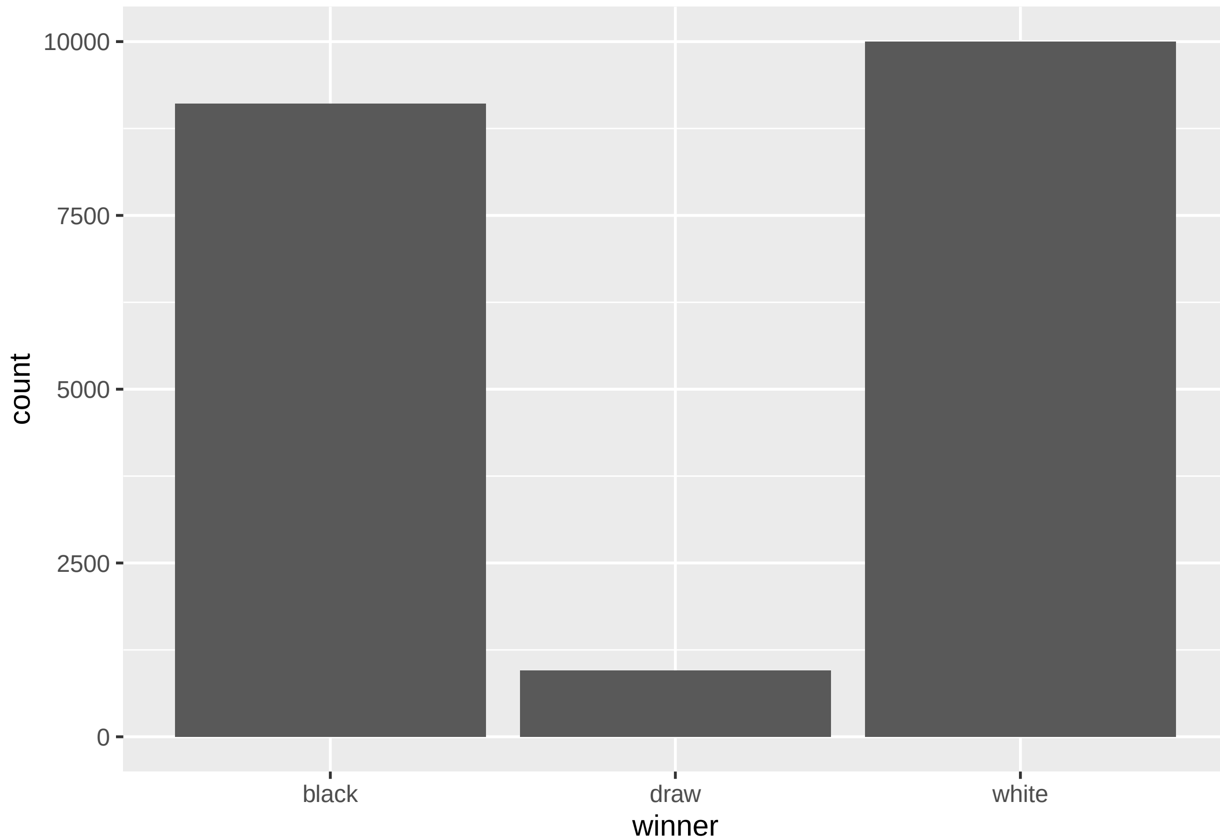
```r
# Turn Distribution
ggplot(chess, aes(x = turns)) + geom_histogram(bins = 100)
```




```r
summary(chess$turns)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00   37.00   55.00   60.47   79.00  349.00
```

```r
# Distribution of the Winner
ggplot(chess, aes(x = winner)) + geom_bar()
```

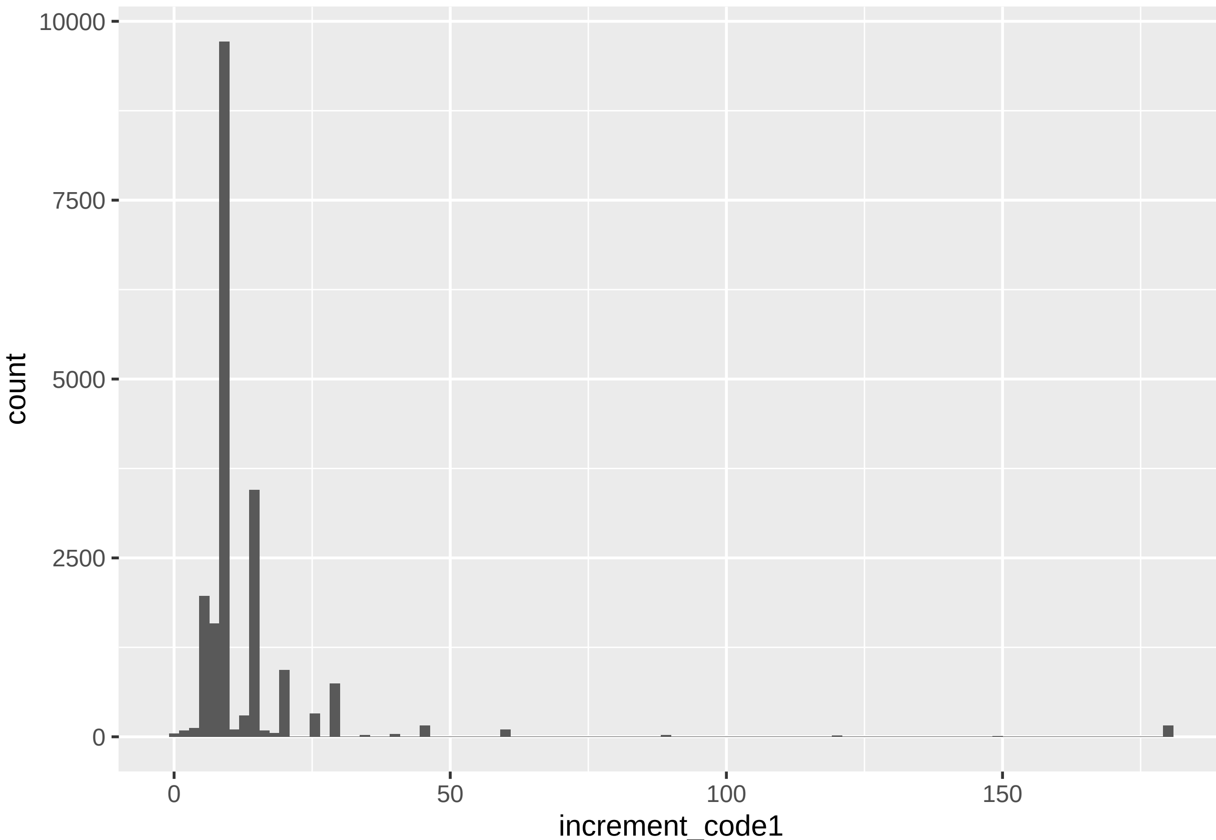



```
#value counts as percentage of winner and sort it in descending order
winner <- as.data.frame(table(chess$winner))
winner$percent <- round(winner$Freq/sum(winner$Freq)*100, 2)
winner <- winner[order(-winner$percent),]
winner
```

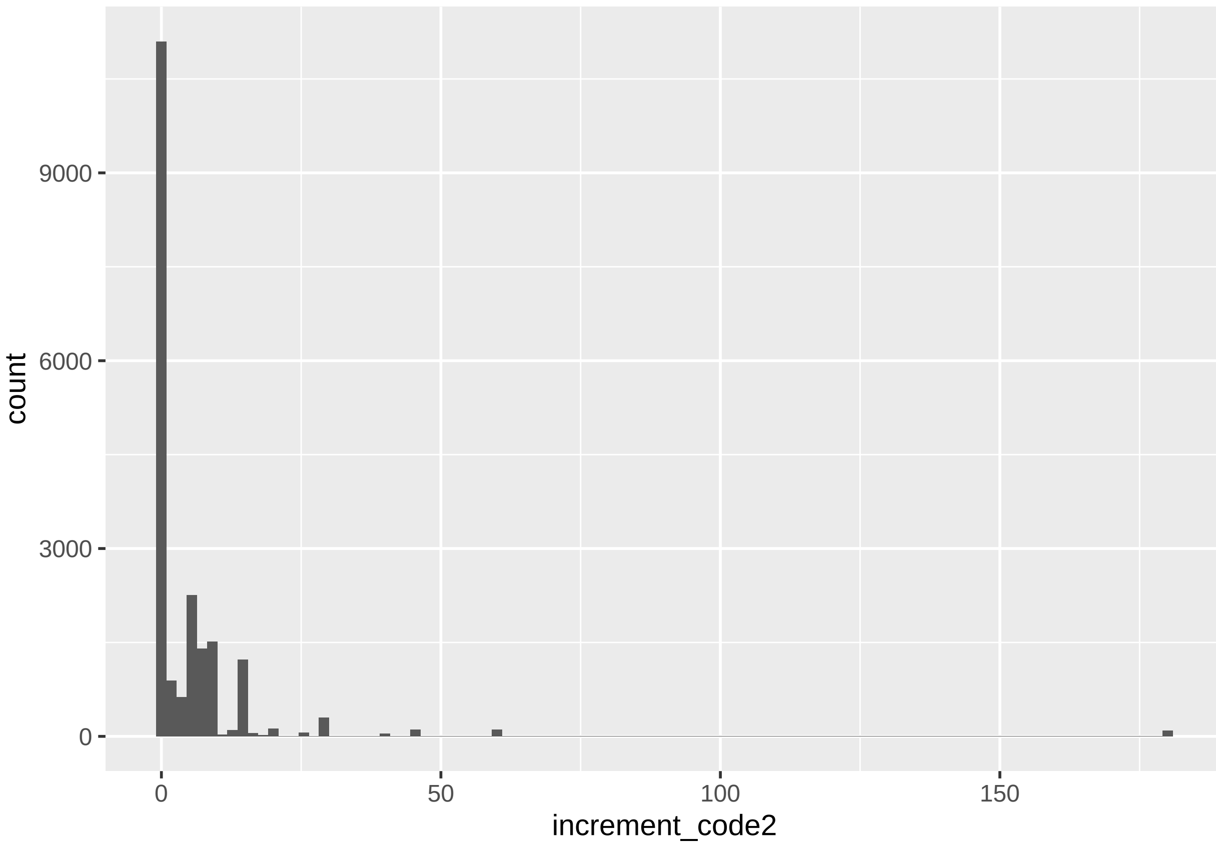
```
##    Var1  Freq percent
## 3 white 10001   49.86
## 1 black  9107   45.40
## 2  draw   950    4.74
```

```
# Divide the increment code into two columns
chess$increment_code <- as.character(chess$increment_code)
# everthing before the plus sign
chess$increment_code1 <- strsplit(chess$increment_code, split = "\\+")
chess$increment_code1 <- sapply(chess$increment_code1, function(x) x[1])
# everything after the plus sign
chess$increment_code2 <- strsplit(chess$increment_code, split = "\\+")
chess$increment_code2 <- sapply(chess$increment_code2, function(x) x[2])
```

```
# convert to numeric and plot the histogram
chess$increment_code1 <- as.numeric(chess$increment_code1)
ggplot(chess, aes(x = increment_code1)) + geom_histogram(bins = 100)
```




```
# convert to numeric and plot the histogram
chess$increment_code2 <- as.numeric(chess$increment_code2)
ggplot(chess, aes(x = increment_code2)) + geom_histogram(bins = 100)
```




```
# Median of the increment code1
median(chess$increment_code1)
```

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

```
# Median of the increment code2
median(chess$increment_code2)
```

```
## [1] 0
```

The median base time of match is 10 minutes and the increment is 0 second which is sudden death.

```
#value counts as percentage of opening_name and sort it in descending order
opening_name <- as.data.frame(table(chess$opening_name))
opening_name$percent <- round(opening_name$Freq/sum(opening_name$Freq)*100, 2)
opening_name <- opening_name[order(-opening_name$percent),]
head(opening_name)
```

```
##                                              Var1 Freq percent
## 1434                         Van't Kruijs Opening  368    1.83
## 1195                             Sicilian Defense  358    1.78
## 1209             Sicilian Defense: Bowdler Attack  296    1.48
## 353              French Defense: Knight Variation  271    1.35
## 1151                                  Scotch Game  271    1.35
## 1142 Scandinavian Defense: Mieses-Kotroc Variation  259    1.29
```

```
dim(opening_name)
```

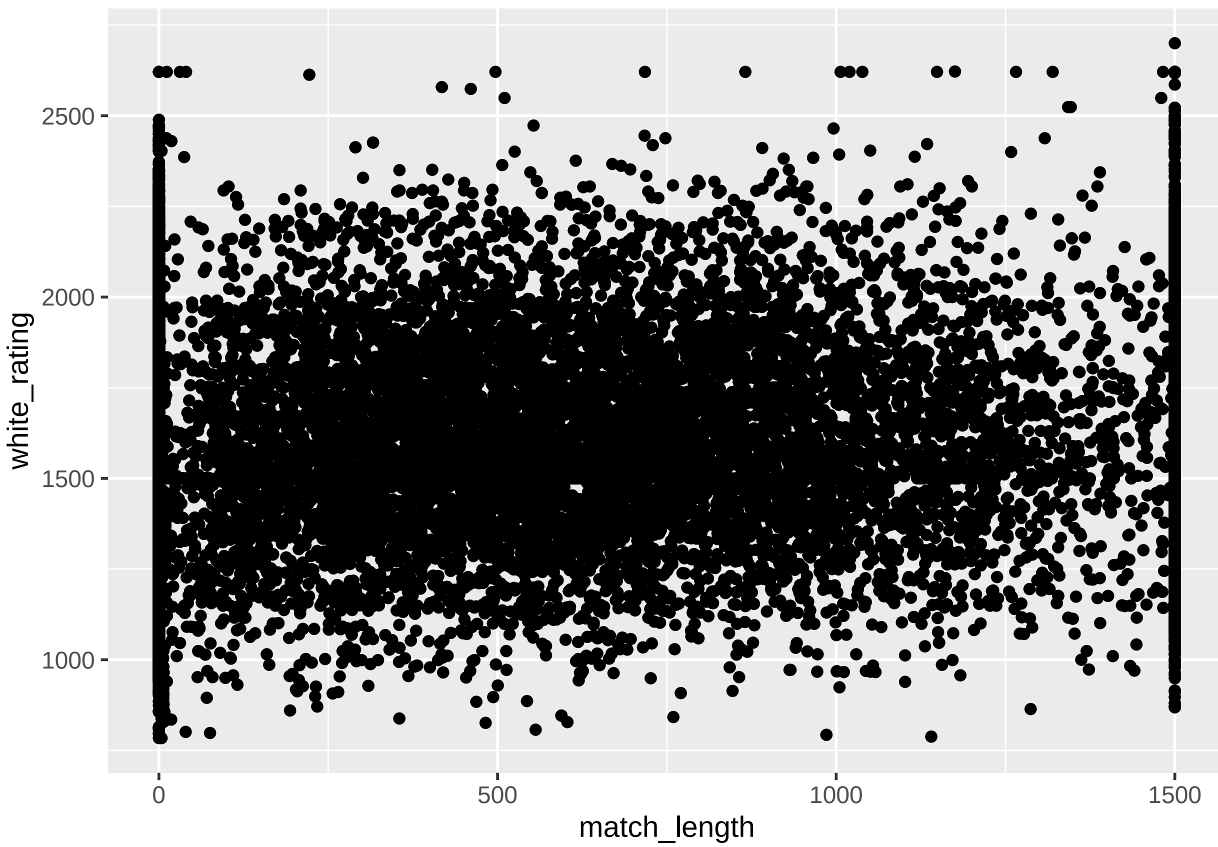
```
## [1] 1477    3
```

There are 1477 unique opening names. The most common opening name is Van't Kruijs Opening.

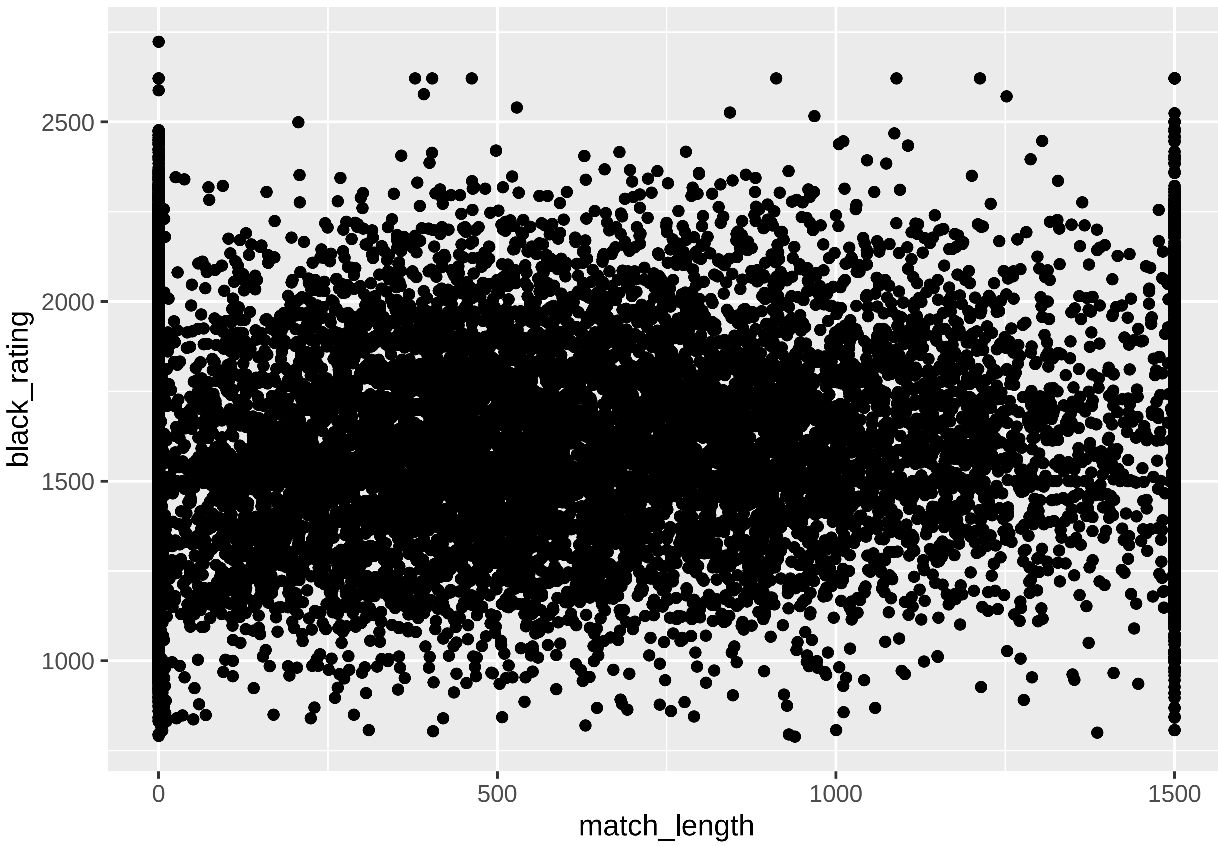
2.5 Bivariate Visualization

Points to check: - Is there a relationship between the match length and the ratings? - Is there a relationship between the match length and the victory status? - Is there a relationship between the match length and the winner? - Is there a relationship between the match length and the increment code? - Is there a relationship between the match length and the opening name? - Is there a relationship between the ratings and the victory status? - Is there a relationship between the ratings and the winner? - Is there a relationship between the ratings and the increment code? - Is there a relationship between the ratings and the opening name? - Is there a relationship between the victory status and the winner? - Is there a relationship between the victory status and the increment code? - Is there a relationship between the victory status and the opening name? - Is there a relationship between the winner and the increment code? - Is there a relationship between the winner and the opening name? - Is there a relationship between the increment code and the opening name?

```
# Is there a relationship between the match length and the ratings?
ggplot(chess, aes(x = match_length, y = white_rating)) + geom_point()
```




```
ggplot(chess, aes(x = match_length, y = black_rating)) + geom_point()
```




```r
# correlation between match_length and white_rating and the best fit line
cor.test(chess$match_length, chess$white_rating)
```

```
##
##  Pearson's product-moment correlation
##
## data:  chess$match_length and chess$white_rating
## t = 16.35, df = 20056, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.1010089 0.1283231
## sample estimates:
##       cor
## 0.1146877
```

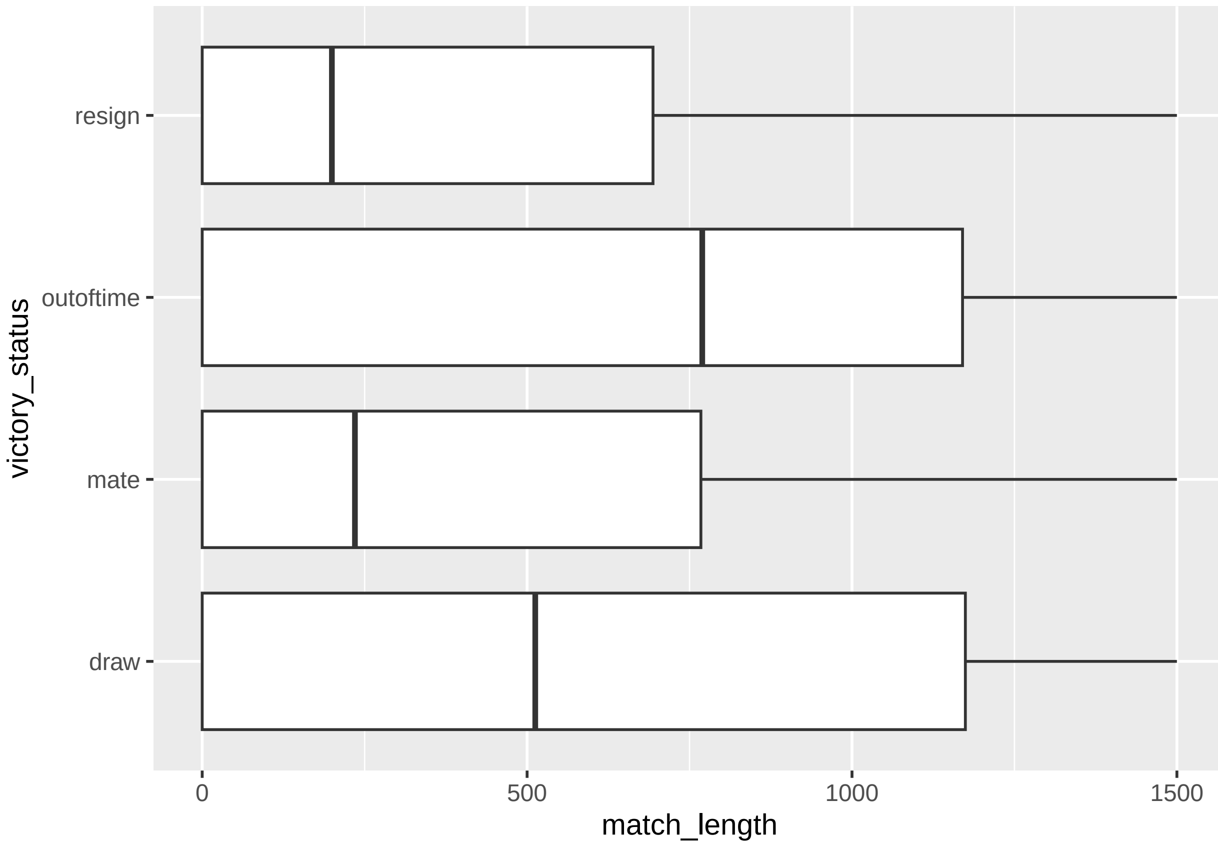
```r
# correlation between match_length and black_rating and the best fit line
cor.test(chess$match_length, chess$black_rating)
```

```
##
##  Pearson's product-moment correlation
##
## data:  chess$match_length and chess$black_rating
## t = 17.919, df = 20056, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.1118833 0.1391255
## sample estimates:
##      cor
```

```
## 0.125528
```

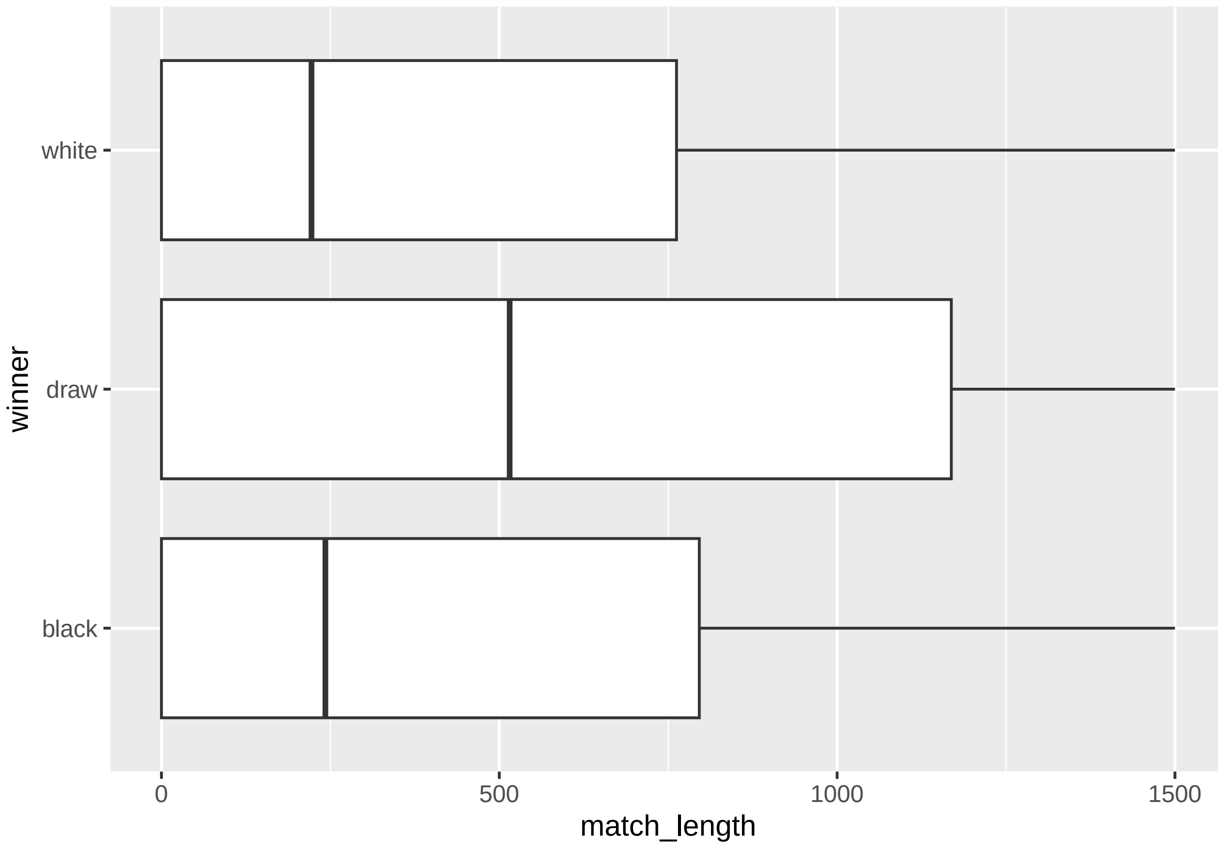
There seems to be no relationship between the match length and the ratings.

```r
# Is there a relationship between the match length and the victory status?
# Check the relationship between a catergorical variable and a numerical variable
ggplot(chess, aes(x = match_length, y = victory_status)) + geom_boxplot()
```



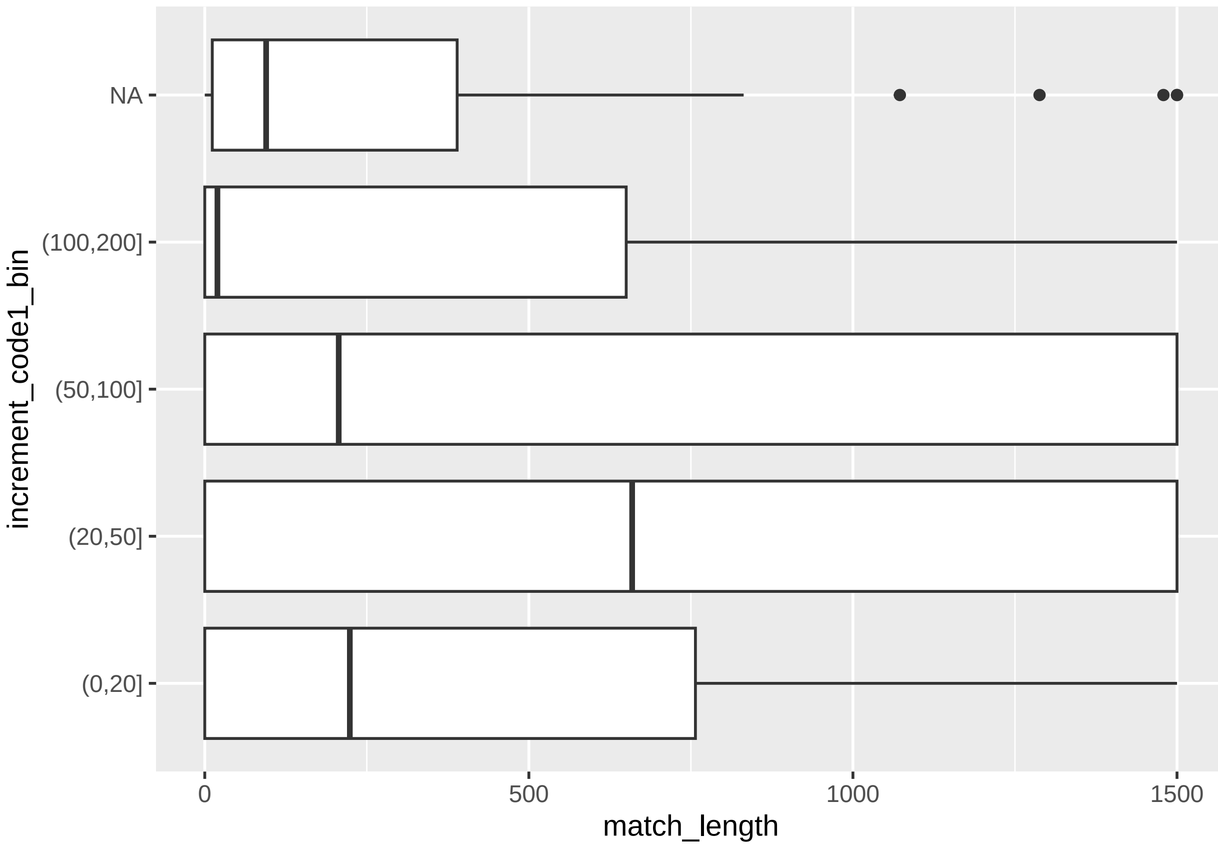
The resigns and mates have a lower match length than the others. The out_of_time has the highest match length than the others. Is it a correlation or a causation?

```r
# Is there a relationship between the match length and the winner?
ggplot(chess, aes(x = match_length, y = winner)) + geom_boxplot()
```



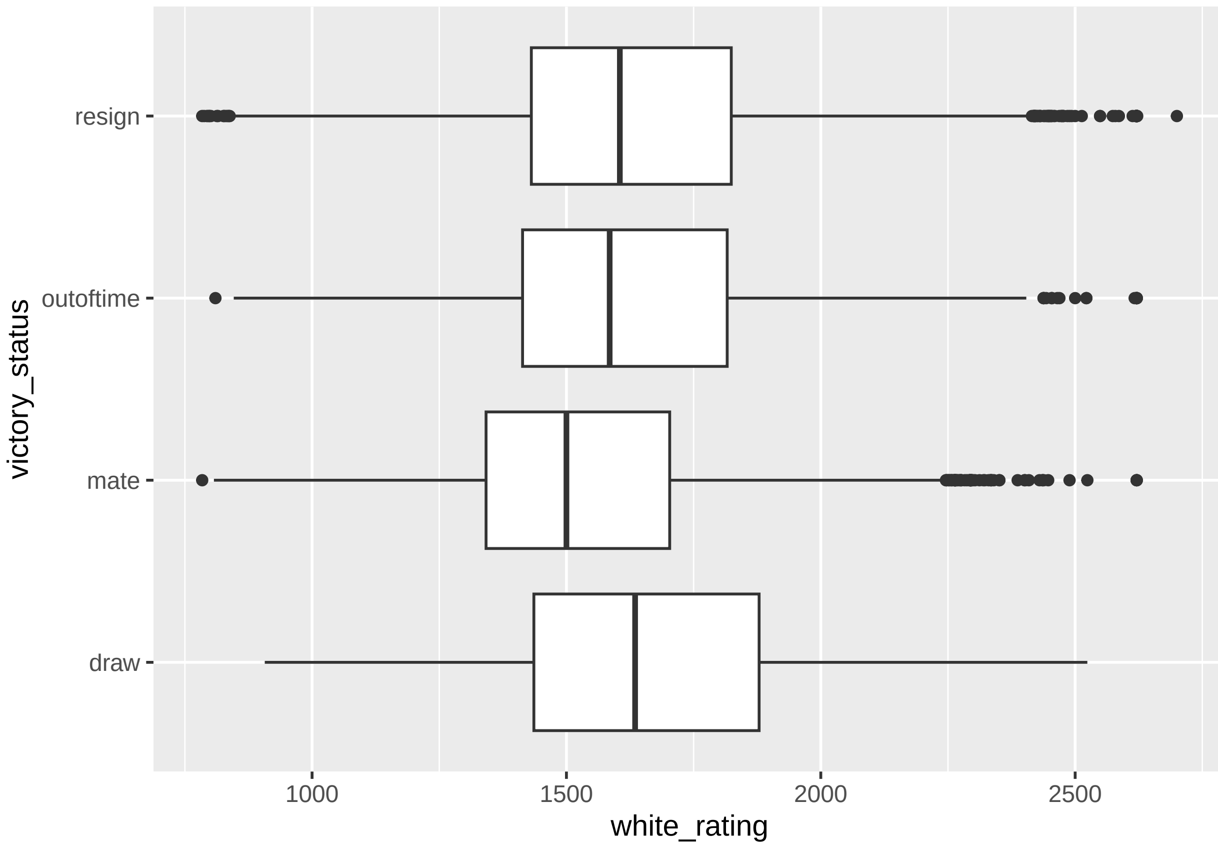
If it's a draw the match length is longer than the others. It seems natural as the draw matches tend to be longer than the others towards the end of the game.

```
# Is there a relationship between the match length and the increment code?
# Bin the increment_code1 into bins based on custom bins
chess$increment_code1_bin <- cut(chess$increment_code1, breaks = c(0, 20, 50, 100, 200, 1000))
ggplot(chess, aes(x = match_length, y = increment_code1_bin)) + geom_boxplot()
```

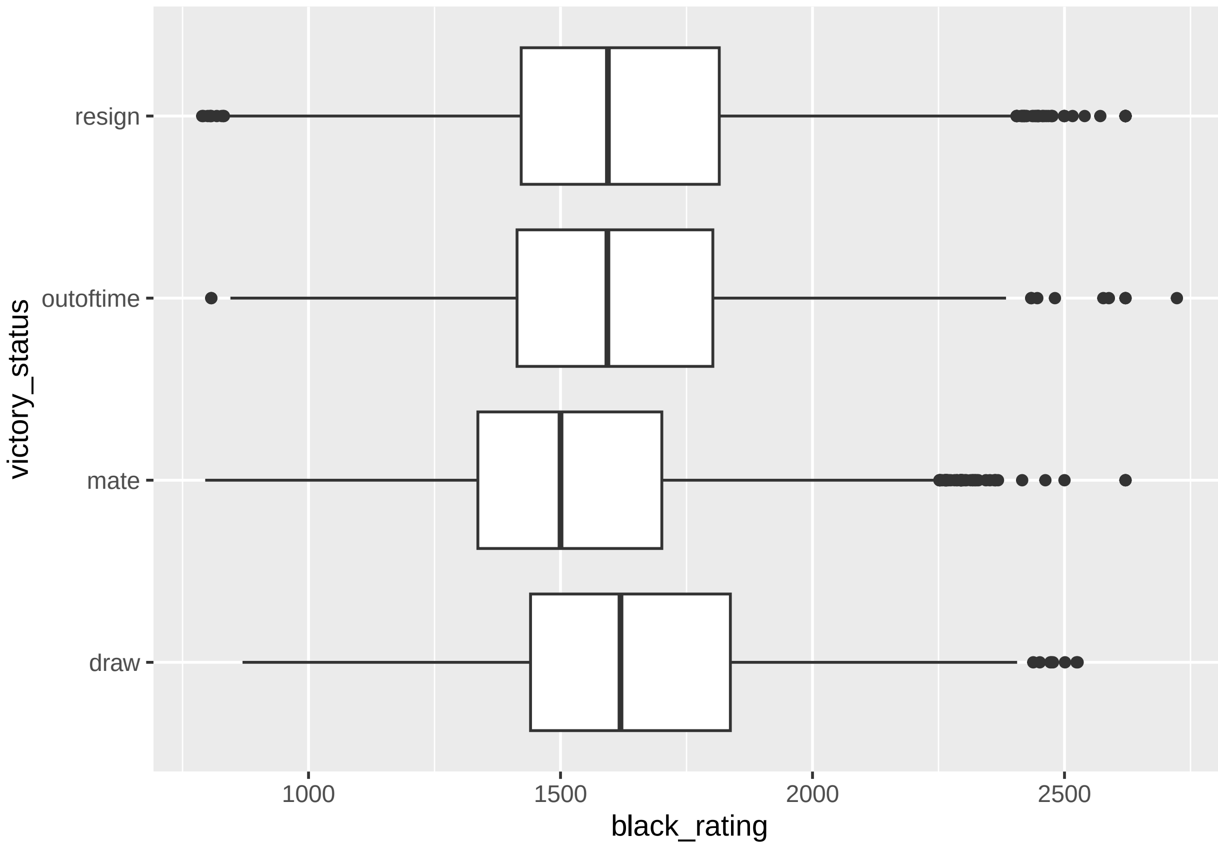


The match length seems to be lower for increment codes between 0-20 as that seems natural due to the sudden death nature of the game. Surprisingly, the match length of increment codes between 50-100 is lower than the match length of increment codes between 20-50.

```
# Is there a relationship between the ratings and the victory status?
ggplot(chess, aes(x = white_rating, y = victory_status)) + geom_boxplot()
```

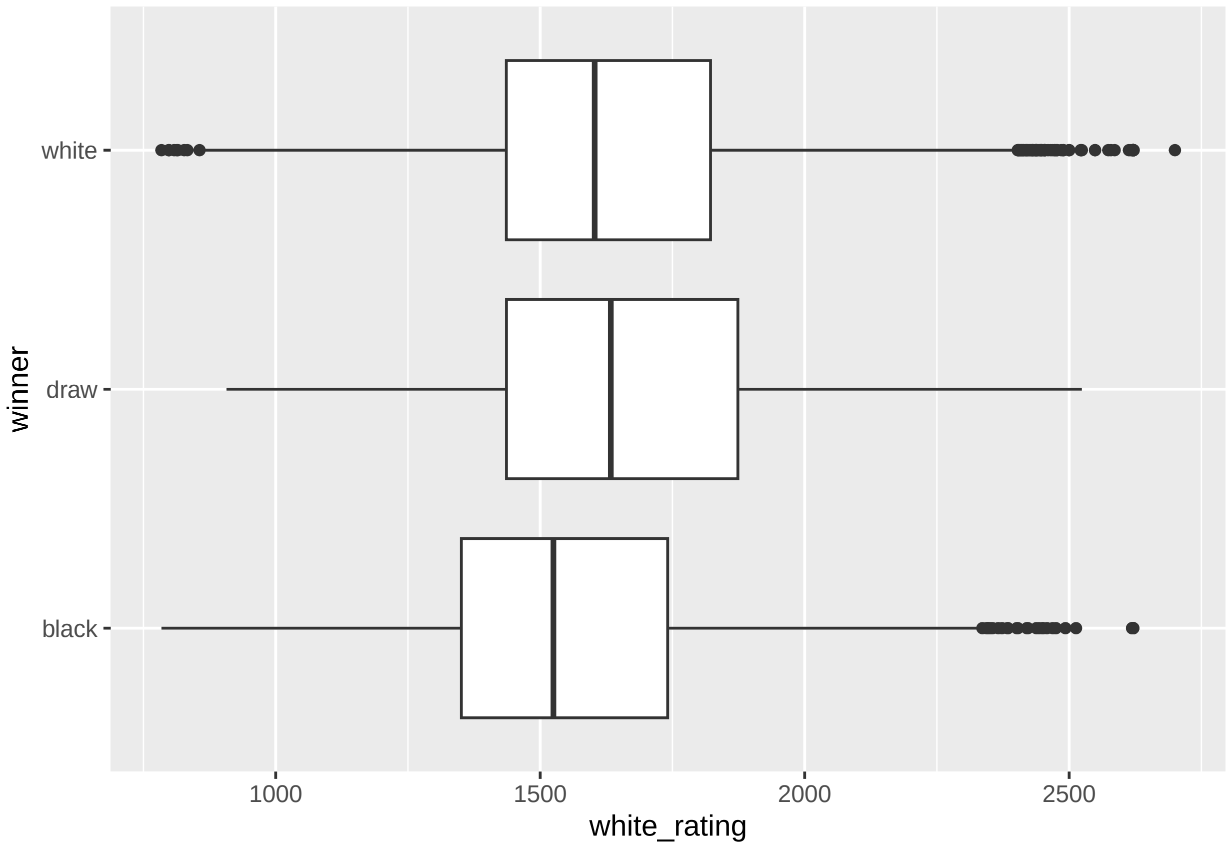



```
ggplot(chess, aes(x = black_rating, y = victory_status)) + geom_boxplot()
```

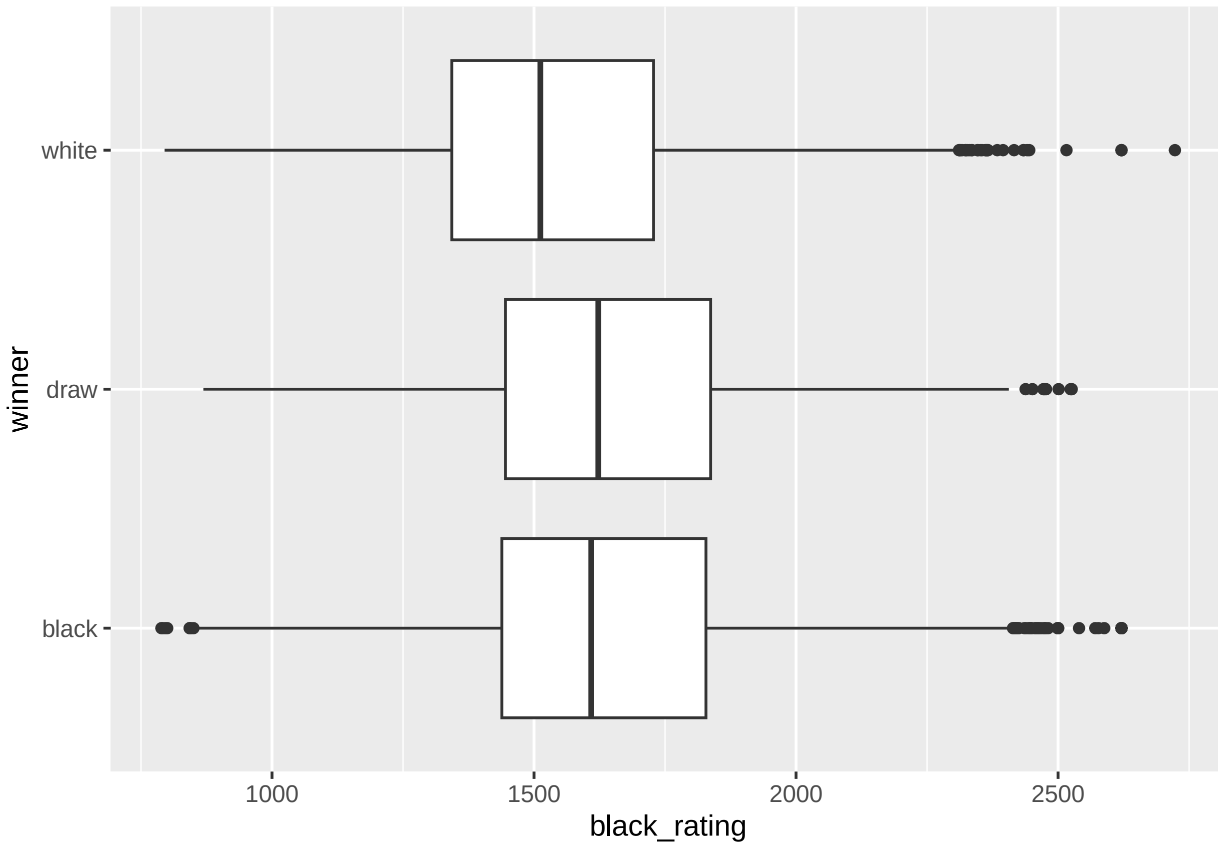


There seems to be no relationship between the ratings and the victory status.

```
# Is there a relationship between the ratings and the winner?
ggplot(chess, aes(x = white_rating, y = winner)) + geom_boxplot()
```

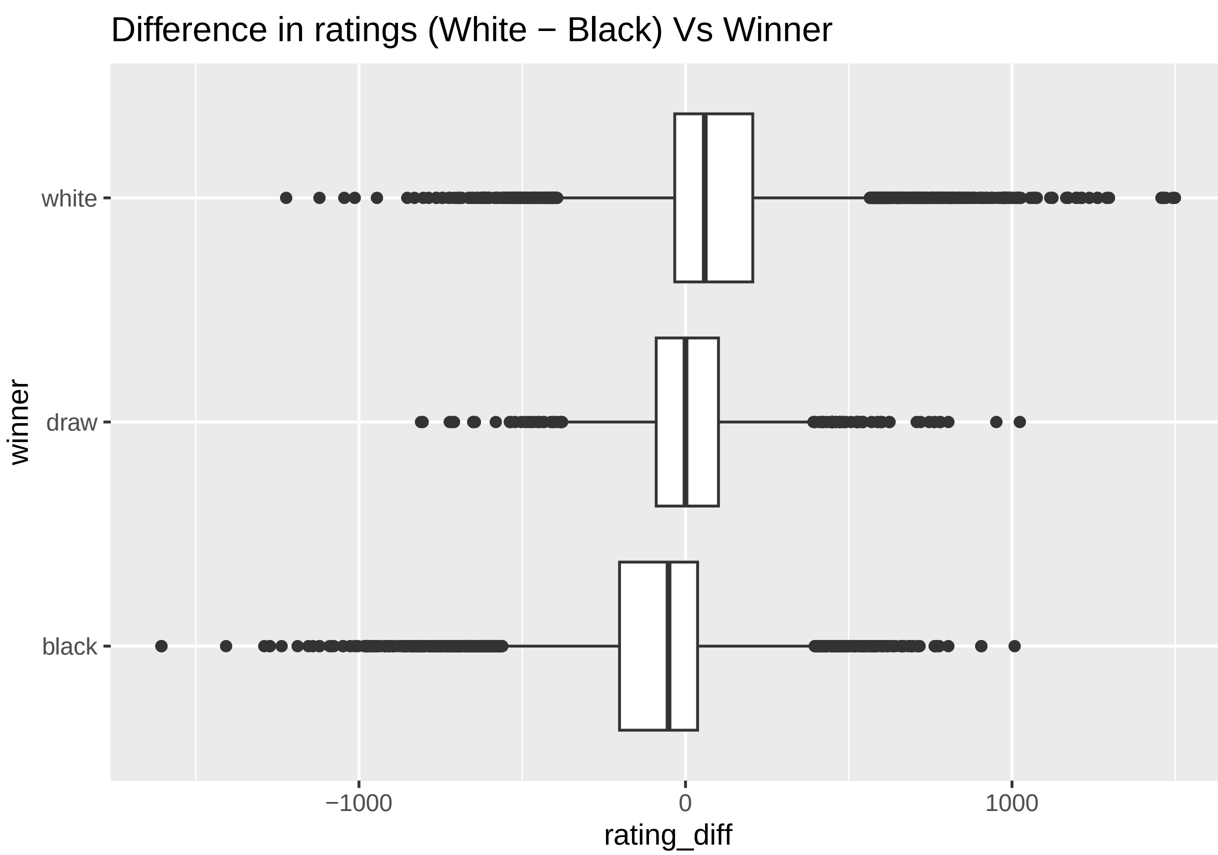



```
ggplot(chess, aes(x = black_rating, y = winner)) + geom_boxplot()
```



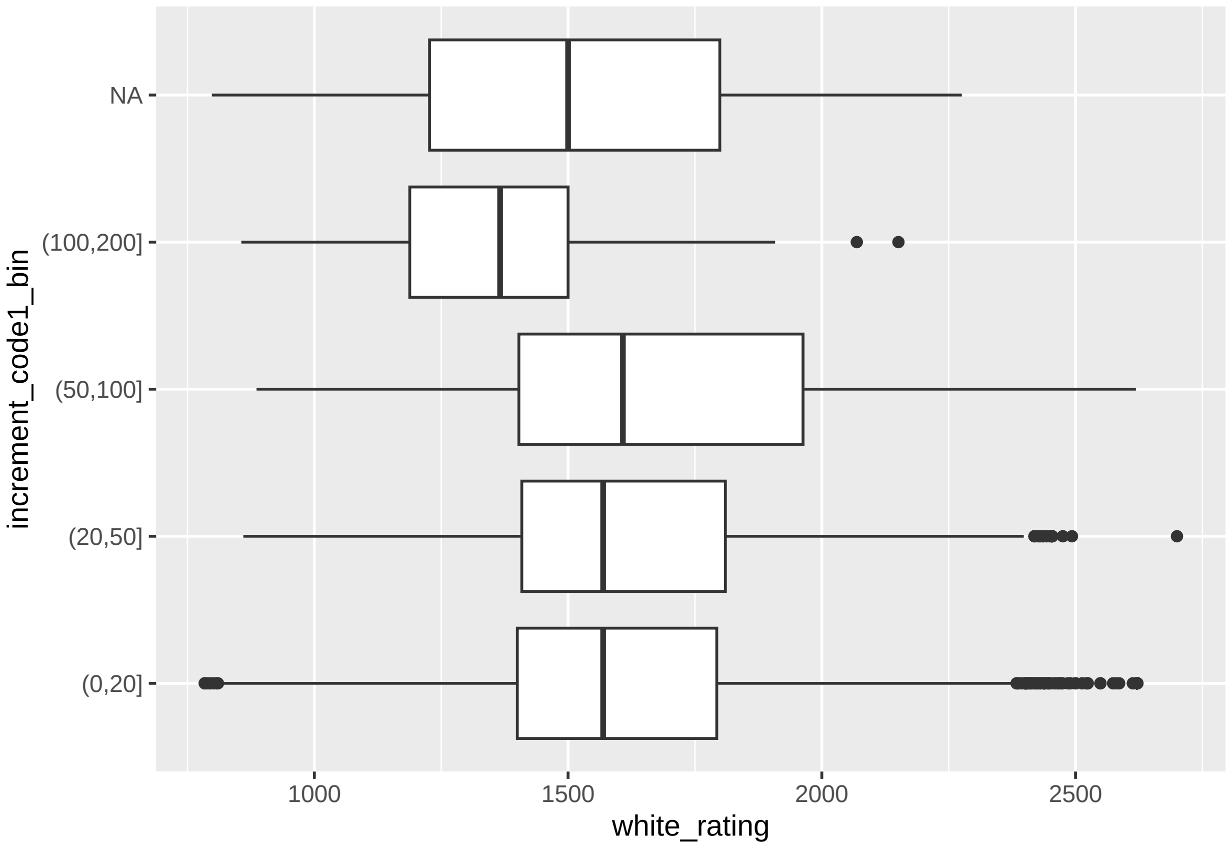
As expected, the higher the rating the higher the chance of winning.

```
# Let's check the difference in ratings between the winner and the loser
chess$rating_diff <- chess$white_rating - chess$black_rating
# Put the title of the plot as the difference in ratings
ggplot(chess, aes(x = rating_diff, y = winner)) + geom_boxplot() + labs(title = "Difference in ratings
```

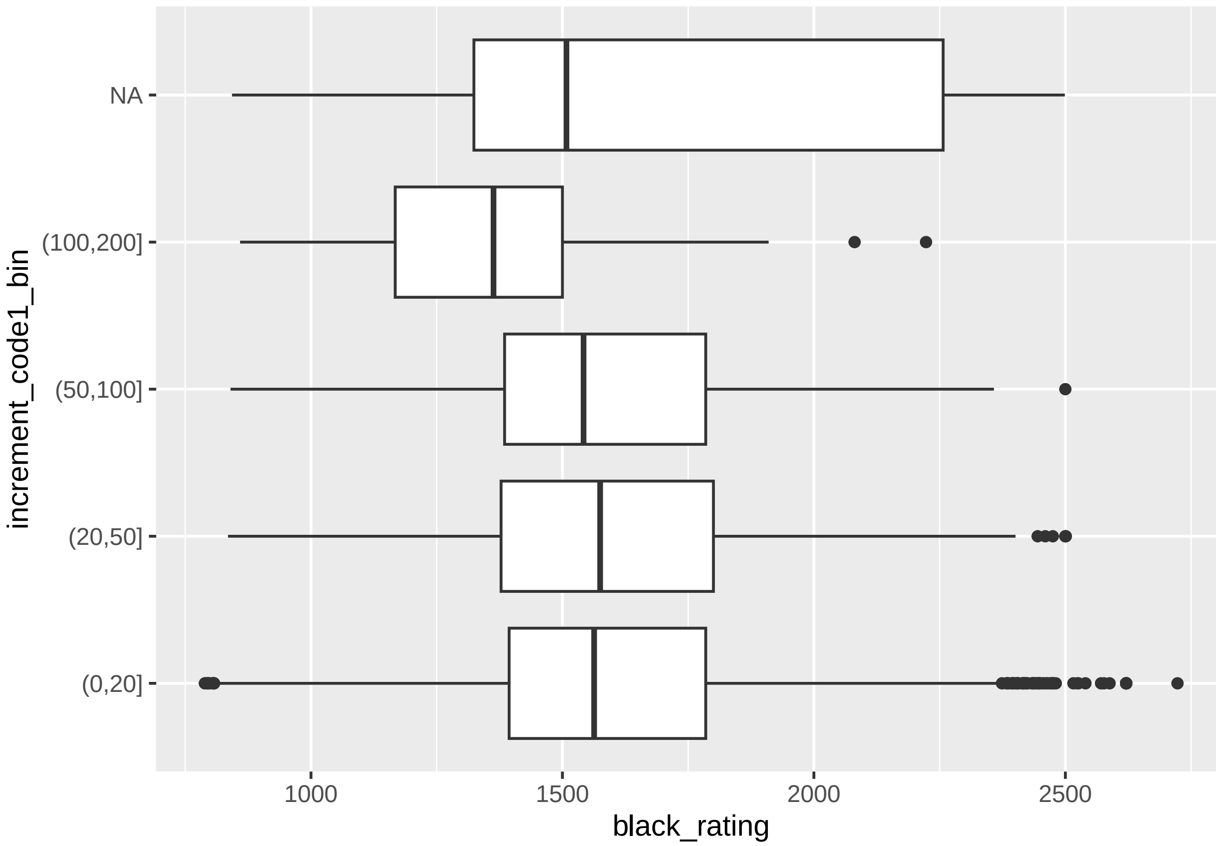


It can be seen in the boxplots that majority of the draws have a difference in ratings of 0. And the difference is postive when white wins and negative when black wins with a few outliers.

```
# Is there a relationship between the ratings and the increment code?
# Bin the increment_code1 into bins based on custom bins
chess$increment_code1_bin <- cut(chess$increment_code1, breaks = c(0, 20, 50, 100, 200, 1000))
ggplot(chess, aes(x = white_rating, y = increment_code1_bin)) + geom_boxplot()
```




```
ggplot(chess, aes(x = black_rating, y = increment_code1_bin)) + geom_boxplot()
```



There seems to be no relationship between the ratings and the increment code.

```
chess$increment_code2_bin <- cut(chess$increment_code2, breaks = c(0, 20, 50, 100, 200, 1000))
```

```
# Is there a relationship between the victory status and the winner?
# Create a table with counts across the two variables
tab <- table(chess$victory_status, chess$winner)
tab
```

```
##
##              black draw white
##   draw           0  906     0
##   mate        2981    0  3344
##   outoftime    823   44   813
##   resign      5303    0  5844
```

```
# Create a table with percentages across the two variables
tab_percent <- round(prop.table(tab, 1)*100, 2)
tab_percent
```

```
##
##               black   draw  white
##   draw         0.00 100.00   0.00
##   mate        47.13   0.00  52.87
##   outoftime   48.99   2.62  48.39
##   resign      47.57   0.00  52.43
```

The table shows that black and white are equally probable to run out of time. There's a small advantage for white to resign and black to mate.

```
# Is there a relationship between the victory status and the increment code?
# Create a table with counts across the two variables
tab <- table(chess$victory_status, chess$increment_code1)
tab
```

```
##
##              0    1    2    3    4    5    6    7    8    9   10   11   12
##   draw       2    1    1    0    0   77    5   30   44    8  376    2   11
##   mate       8    9   19   25   23  537   51  218  273   91 3035   30   67
##   outoftime 13    3    4    4   10  166   15   53  128   30  890   11   17
##   resign    22   18   31   28   32 1015  105  360  479  132 5155   55  130
##
##             13   14   15   16   17   18   19   20   21   25   29   30   35
##   draw       1    2  169    6    1    0    1   56    0   13    0   48    2
##   mate      21   21 1091   18   17    7   12  331    0   78    0  207    5
##   outoftime  1    8  207   11    0    3    0   46    0   15    0   22    2
##   resign    51   39 1915   14   22   16   15  503    1  216    2  469   17
##
##             40   45   60   90  120  150  180
##   draw       5   15   11    0    1    1   17
##   mate       9   34   32   12    4    3   37
##   outoftime  0    0    3    1    3    3   11
##   resign    23  109   57   13    8    6   89
```

```
# Create a table with percentages across the two variables
tab_percent <- round(prop.table(tab, 1)*100, 2)
tab_percent
```

```
##
##                 0     1     2     3     4     5     6     7     8     9    10
##   draw       0.22  0.11  0.11  0.00  0.00  8.50  0.55  3.31  4.86  0.88 41.50
##   mate       0.13  0.14  0.30  0.40  0.36  8.49  0.81  3.45  4.32  1.44 47.98
##   outoftime  0.77  0.18  0.24  0.24  0.60  9.88  0.89  3.15  7.62  1.79 52.98
##   resign     0.20  0.16  0.28  0.25  0.29  9.11  0.94  3.23  4.30  1.18 46.25
##
##                11    12    13    14    15    16    17    18    19    20    21
##   draw       0.22  1.21  0.11  0.22 18.65  0.66  0.11  0.00  0.11  6.18  0.00
##   mate       0.47  1.06  0.33  0.33 17.25  0.28  0.27  0.11  0.19  5.23  0.00
##   outoftime  0.65  1.01  0.06  0.48 12.32  0.65  0.00  0.18  0.00  2.74  0.00
##   resign     0.49  1.17  0.46  0.35 17.18  0.13  0.20  0.14  0.13  4.51  0.01
##
##                25    29    30    35    40    45    60    90   120   150   180
##   draw       1.43  0.00  5.30  0.22  0.55  1.66  1.21  0.00  0.11  0.11  1.88
##   mate       1.23  0.00  3.27  0.08  0.14  0.54  0.51  0.19  0.06  0.05  0.58
##   outoftime  0.89  0.00  1.31  0.12  0.00  0.00  0.18  0.06  0.18  0.18  0.65
##   resign     1.94  0.02  4.21  0.15  0.21  0.98  0.51  0.12  0.07  0.05  0.80
```

```
# Is there a relationship between the winner and the increment code?
# Create a table with counts across the two variables
tab <- table(chess$winner, chess$increment_code1_bin)
tab
```

```
##
##          (0,20] (20,50] (50,100] (100,200] (200,1e+03]
##   black    8396     560       47        81           0
##   draw      834      84       11        19           0
```

```
##   white   9179     648       71        83           0
```

```
tab_percent <- round(prop.table(tab, 1)*100, 2)
tab_percent
```

```
##
##          (0,20] (20,50] (50,100] (100,200] (200,1e+03]
##   black  92.43    6.16     0.52      0.89        0.00
##   draw   87.97    8.86     1.16      2.00        0.00
##   white  91.96    6.49     0.71      0.83        0.00
```

```
# Let's do some analysis on Moves
# Is there a rela tionship between the match length and the number of moves?
# correlation between match_length and moves and the best fit line
cor.test(chess$match_length, chess$turns)
```

```
##
##  Pearson's product-moment correlation
##
## data:  chess$match_length and chess$turns
## t = 49.485, df = 20056, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.3174771 0.3421442
## sample estimates:
##       cor
## 0.3298669
```

Ther's seems a weak correlation between the match length and the number of moves.

```
# Deep down into the column moves
# Create a new column with the number of moves
chess$number_of_moves <- sapply(chess$moves, function(x) length(strsplit(x, " ")[[1]]))

# correlation between turns and number_of_moves and the best fit line
cor.test(chess$turns, chess$number_of_moves)
```

```
##
##  Pearson's product-moment correlation
##
## data:  chess$turns and chess$number_of_moves
## t = Inf, df = 20056, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  1 1
## sample estimates:
## cor
##   1
```

```
# confirming that the number of moves and turns is equal for all rows.
chess[chess$number_of_moves != chess$turns]
```

```
## data frame with 0 columns and 20058 rows
```

3 Feature Engineering and Selection

```
a. Chi square test
```

```
b. ANOV A test
c. Correlation test
```

Given the data, we are going to predict the winner of the game. We'll treat black and white as 1, and 0 respectively. We'll remove the draw cases as they are very few in number.

```
# Remove the draws
chess <- subset(chess, winner != "draw")
# Convert the winner column to 1 and 0
chess$winner <- ifelse(chess$winner == "white", 1, 0)
```

We'll perform chi square test to check the relationship between the categorical variables and the winner.

```
# get column names
colnames(chess)
```

```
##  [1] "id"                  "rated"               "created_at"
##  [4] "last_move_at"        "turns"               "victory_status"
##  [7] "winner"              "increment_code"      "white_id"
## [10] "white_rating"        "black_id"            "black_rating"
## [13] "moves"               "opening_eco"         "opening_name"
## [16] "opening_ply"         "match_length"        "increment_code1"
## [19] "increment_code2"     "increment_code1_bin" "rating_diff"
## [22] "increment_code2_bin" "number_of_moves"
```

Chi square test

Catergorical variables that we will consider:

victory_status, increment_code1_bin, rated

Create a function to perform chi square test

```
chisq_test <- function(x, y) {
  # Create a table with counts across the two variables
  tab <- table(x, y)
  # Create a table with percentages across the two variables
  tab_percent <- round(prop.table(tab, 1)*100, 2)
  # Perform chi square test
  chi_sq <- chisq.test(tab)
  # Print the results
  print(chi_sq)
  print(tab_percent)
}
```

```
# Loop over categorical variables and perform chi square test
for (i in c("victory_status", "increment_code1_bin", "increment_code2_bin", "rated")) {
  print(i)
  chisq_test(chess[[i]], chess$winner)
}
```

```
## [1] "victory_status"
##
##  Pearson's Chi-squared test
##
```

```
## data:  tab
## X-squared = 5.335, df = 2, p-value = 0.06942
##
##            y
## x                0     1
##   mate      47.13 52.87
##   outoftime 50.31 49.69
##   resign    47.57 52.43
## [1] "increment_code1_bin"

## Warning in chisq.test(tab): Chi-squared approximation may be incorrect

##
##  Pearson's Chi-squared test
##
## data:  tab
## X-squared = NaN, df = 4, p-value = NA
##
##              y
## x                  0     1
##   (0,20]      47.77 52.23
##   (20,50]     46.36 53.64
##   (50,100]    39.83 60.17
##   (100,200]   49.39 50.61
##   (200,1e+03]
## [1] "increment_code2_bin"

## Warning in chisq.test(tab): Chi-squared approximation may be incorrect

##
##  Pearson's Chi-squared test
##
## data:  tab
## X-squared = NaN, df = 4, p-value = NA
##
##              y
## x                  0     1
##   (0,20]      47.69 52.31
##   (20,50]     45.17 54.83
##   (50,100]    38.46 61.54
##   (100,200]   43.68 56.32
##   (200,1e+03]
## [1] "rated"
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  tab
## X-squared = 0.95618, df = 1, p-value = 0.3282
##
##        y
## x           0     1
##   FALSE 46.92 53.08
##   TRUE  47.84 52.16
```

```r
# chisq_test(chess$increment_code1_bin, chess$winner)
head(chess$increment_code1_bin)
```

```
## [1] (0,20]  (0,20]  (0,20]  (0,20]  (20,50] (0,20]
## Levels: (0,20] (20,50] (50,100] (100,200] (200,1e+03]
```

ANOVA test

Numerical variables that we will consider:

white_rating, black_rating, rating_diff, match_length, turns, number_of_moves,

increment_code1, increment_code2

Create a function to perform ANOVA test

```r
anova_test <- function(x, y) {
  # Perform ANOVA test
  anova <- aov(x ~ y)
  # Print the results
  print(anova)
#   get the summary
  print(summary(anova))
}
```

```r
# Loop over numerical variables and perform ANOVA test
for (i in c("white_rating", "black_rating", "rating_diff", "match_length",  "increment_code1", "incremen
  print(i)
  anova_test(chess[[i]], chess$winner)
}
```

```
## [1] "white_rating"
## Call:
##    aov(formula = x ~ y)
##
## Terms:
##                             y  Residuals
## Sum of Squares    34388051 1571953569
## Deg. of Freedom          1      19106
##
## Residual standard error: 286.8369
## Estimated effects may be unbalanced
##                  Df    Sum Sq  Mean Sq F value Pr(>F)
## y                 1 3.439e+07 34388051     418 <2e-16 ***
## Residuals   19106 1.572e+09    82275
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## [1] "black_rating"
## Call:
##    aov(formula = x ~ y)
##
## Terms:
##                             y  Residuals
## Sum of Squares    47048349 1561473816
```

```
## Deg. of Freedom          1      19106
##
## Residual standard error: 285.8791
## Estimated effects may be unbalanced
##                Df    Sum Sq  Mean Sq F value Pr(>F)
## y               1 4.705e+07 47048349   575.7 <2e-16 ***
## Residuals   19106 1.561e+09    81727
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## [1] "rating_diff"
## Call:
##    aov(formula = x ~ y)
##
## Terms:
##                          y  Residuals
## Sum of Squares   161882682 1038564506
## Deg. of Freedom          1      19106
##
## Residual standard error: 233.1481
## Estimated effects may be unbalanced
##                Df    Sum Sq   Mean Sq F value Pr(>F)
## y               1 1.619e+08 161882682    2978 <2e-16 ***
## Residuals   19106 1.039e+09     54358
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## [1] "match_length"
## Call:
##    aov(formula = x ~ y)
##
## Terms:
##                          y  Residuals
## Sum of Squares      895523 5015295832
## Deg. of Freedom          1      19106
##
## Residual standard error: 512.346
## Estimated effects may be unbalanced
##                Df    Sum Sq Mean Sq F value Pr(>F)
## y               1 8.955e+05  895523   3.412 0.0648 .
## Residuals   19106 5.015e+09  262498
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## [1] "increment_code1"
## Call:
##    aov(formula = x ~ y)
##
## Terms:
##                      y Residuals
## Sum of Squares      187   5330892
## Deg. of Freedom       1     19106
##
## Residual standard error: 16.70379
## Estimated effects may be unbalanced
##                Df  Sum Sq Mean Sq F value Pr(>F)
## y               1     187   186.7   0.669  0.413
```

```
## Residuals   19106 5330892   279.0
## [1] "increment_code2"
## Call:
##    aov(formula = x ~ y)
##
## Terms:
##                        y Residuals
## Sum of Squares       398   3737684
## Deg. of Freedom        1     19106
##
## Residual standard error: 13.98674
## Estimated effects may be unbalanced
##                Df  Sum Sq Mean Sq F value Pr(>F)
## y               1     398   398.4   2.036  0.154
## Residuals   19106 3737684   195.6
```

Increment codes have no relationship with the winner. black_rating and white_rating have a high significance with the winner. Rating diff also has a high significance with the winner.

```r
# Correlation test
# Numerical variables that we will consider:
# white_rating, black_rating, rating_diff,

# Create a function to perform correlation test
cor_test <- function(x, y) {
  # Perform correlation test
  cor <- cor.test(x, y)
  # Print the results
  print(cor)
}
```

```r
# Loop over numerical variables and perform correlation test
for (i in c("white_rating", "black_rating", "rating_diff")) {
  print(i)
  cor_test(chess[[i]], chess$winner)
}
```

```
## [1] "white_rating"
##
##  Pearson's product-moment correlation
##
## data:  x and y
## t = 20.444, df = 19106, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.1324093 0.1601604
## sample estimates:
##       cor
## 0.1463136
##
## [1] "black_rating"
##
##  Pearson's product-moment correlation
##
## data:  x and y
## t = -23.993, df = 19106, p-value < 2.2e-16
```

```
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.1847556 -0.1572269
## sample estimates:
##        cor
## -0.1710246
##
## [1] "rating_diff"
##
##  Pearson's product-moment correlation
##
## data:  x and y
## t = 54.572, df = 19106, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.3548908 0.3794254
## sample estimates:
##      cor
## 0.367222
```

```r
# correaltion between the numerical variables
cor(chess[, c("white_rating", "black_rating", "rating_diff")])
```

```
##              white_rating black_rating rating_diff
## white_rating    1.0000000    0.6265948   0.4314525
## black_rating    0.6265948    1.0000000  -0.4327292
## rating_diff     0.4314525   -0.4327292   1.0000000
```

4. Linear Regression

a. Linear Regression assumption validation

```r
source("model_training.R")
```

```
##
## Call:  glm(formula = winner ~ ., family = "binomial", data = train)
##
## Coefficients:
## (Intercept)  rating_diff
##    0.094058     0.003668
##
## Degrees of Freedom: 5907 Total (i.e. Null);  5906 Residual
## Null Deviance:       8170
## Residual Deviance: 7275  AIC: 7279
##
## Call:
## glm(formula = winner ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7617  -1.1061   0.4932   1.0547   2.9678
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) 0.0940575  0.0280758    3.35 0.000808 ***
```

```
## rating_diff 0.0036678  0.0001447   25.34  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 8169.7  on 5907  degrees of freedom
## Residual deviance: 7274.8  on 5906  degrees of freedom
## AIC: 7278.8
##
## Number of Fisher Scoring iterations: 4
##
##
##     FALSE TRUE
##   0  1026  192
##   1   771  542
## [1] 0.619518
##
## Call:  glm(formula = winner ~ ., family = "binomial", data = train)
##
## Coefficients:
##  (Intercept)   rating_diff  black_rating
##    1.626e-01     3.451e-03    -5.729e-05
##
## Degrees of Freedom: 5907 Total (i.e. Null);  5905 Residual
## Null Deviance:       8176
## Residual Deviance: 7349  AIC: 7355
##
## Call:
## glm(formula = winner ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6746  -1.1134   0.5035   1.0666   2.9080
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)   1.626e-01  1.681e-01   0.967    0.333
## rating_diff   3.451e-03  1.497e-04  23.048   <2e-16 ***
## black_rating -5.729e-05  1.053e-04  -0.544    0.586
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 8176.2  on 5907  degrees of freedom
## Residual deviance: 7349.3  on 5905  degrees of freedom
## AIC: 7355.3
##
## Number of Fisher Scoring iterations: 4
##
##
##     FALSE TRUE
##   0  1042  146
```

```
##   1   804  539
## [1] 0.6246543

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

##
## Call:  glm(formula = winner ~ ., family = "binomial", data = train)
##
## Coefficients:
##  (Intercept)   rating_diff  black_rating  white_rating
##    0.4275684     0.0035067    -0.0002229            NA
##
## Degrees of Freedom: 5907 Total (i.e. Null);  5905 Residual
## Null Deviance:       8174
## Residual Deviance: 7300  AIC: 7306
##
## Call:
## glm(formula = winner ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5540  -1.1046   0.5001   1.0622   2.9931
##
## Coefficients: (1 not defined because of singularities)
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)   0.4275684  0.1678447   2.547   0.0109 *
## rating_diff   0.0035067  0.0001529  22.927   <2e-16 ***
## black_rating -0.0002229  0.0001051  -2.121   0.0339 *
## white_rating         NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 8173.5  on 5907  degrees of freedom
## Residual deviance: 7299.5  on 5905  degrees of freedom
## AIC: 7305.5
##
## Number of Fisher Scoring iterations: 4
##
##
##     FALSE TRUE
##   0  1040  161
##   1   799  531
## [1] 0.6207033

## Warning in chisq.test(table): Chi-squared approximation may be incorrect

## Warning in chisq.test(table): Chi-squared approximation may be incorrect

##
## Call:  glm(formula = winner ~ ., family = "binomial", data = train)
##
## Coefficients:
## (Intercept)    ratedTRUE
```

```
##     0.14761     -0.03242
##
## Degrees of Freedom: 5907 Total (i.e. Null);  5906 Residual
## Null Deviance:       8168
## Residual Deviance: 8168  AIC: 8172
##
## Call:
## glm(formula = winner ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.241  -1.227   1.115   1.129   1.129
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.14761    0.05416   2.725  0.00642 **
## ratedTRUE   -0.03242    0.06179  -0.525  0.59976
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 8168.0  on 5907  degrees of freedom
## Residual deviance: 8167.8  on 5906  degrees of freedom
## AIC: 8171.8
##
## Number of Fisher Scoring iterations: 3
##
##
##     FALSE
##   0  1225
##   1  1306
## [1] 0.4839984

## Warning in chisq.test(table): Chi-squared approximation may be incorrect

## Warning in chisq.test(table): Chi-squared approximation may be incorrect

##
## Call:  glm(formula = winner ~ ., family = "binomial", data = train)
##
## Coefficients:
## (Intercept)    ratedTRUE
##     0.16140     -0.07593
##
## Degrees of Freedom: 5907 Total (i.e. Null);  5906 Residual
## Null Deviance:       8175
## Residual Deviance: 8173  AIC: 8177
##
## Call:
## glm(formula = winner ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.247  -1.214   1.110   1.141   1.141
```

```
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.16140    0.05429   2.973  0.00295 **
## ratedTRUE   -0.07593    0.06188  -1.227  0.21985
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 8174.6  on 5907  degrees of freedom
## Residual deviance: 8173.1  on 5906  degrees of freedom
## AIC: 8177.1
##
## Number of Fisher Scoring iterations: 3
##
##
##     FALSE
##   0  1196
##   1  1335
## [1] 0.4725405
##
## Call:  glm(formula = winner ~ ., family = "binomial", data = train)
##
## Coefficients:
##  (Intercept)   rating_diff  black_rating
##    0.3404556     0.0036076    -0.0001561
##
## Degrees of Freedom: 5907 Total (i.e. Null);  5905 Residual
## Null Deviance:       8169
## Residual Deviance: 7269  AIC: 7275
##
## Call:
## glm(formula = winner ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7382  -1.1097   0.4934   1.0523   3.0026
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)   0.3404556  0.1683834   2.022   0.0432 *
## rating_diff   0.0036076  0.0001541  23.414   <2e-16 ***
## black_rating -0.0001561  0.0001055  -1.480   0.1388
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 8168.8  on 5907  degrees of freedom
## Residual deviance: 7269.3  on 5905  degrees of freedom
## AIC: 7275.3
##
## Number of Fisher Scoring iterations: 4
```

```
##
##
##     FALSE TRUE
##   0  1033  189
##   1   776  533
## [1] 0.6187278
```

```r
# Preprocessing function
chess <- read.csv("games.csv")
# Preprocess the data
chess <- preprocess(chess)
chess <- chess %>% select(-rating_diff) # nolint
model <- train_model(chess)
```

```r
# Linear Regression assumption validation
# 1. Linearity
# Let's check the linearity of the numerical variables
# Create a function to check the linearity of the numerical variables
linearity_test <- function(x, y) {
  # Perform linear regression
  lm <- lm(x ~ y)
  # Print the results
  print(lm)
  print(summary(lm))
}
```

```r
# Loop over numerical variables and perform linear regression
for (i in c("white_rating", "black_rating")) {
  print(i)
  linearity_test(chess[[i]], chess$winner)
}
```

```
## [1] "white_rating"
##
## Call:
## lm(formula = x ~ y)
##
## Coefficients:
## (Intercept)            y
##     1543.35        81.29
##
##
## Call:
## lm(formula = x ~ y)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -768.65 -203.35  -37.35  181.35 1077.65
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1543.353      4.698  328.52   <2e-16 ***
## y             81.295      6.476   12.55   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 297.1 on 8437 degrees of freedom
## Multiple R-squared:  0.01833,    Adjusted R-squared:  0.01822
## F-statistic: 157.6 on 1 and 8437 DF,  p-value: < 2.2e-16
##
## [1] "black_rating"
##
## Call:
## lm(formula = x ~ y)
##
## Coefficients:
## (Intercept)            y
##      1627.8       -106.1
##
##
## Call:
## lm(formula = x ~ y)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -836.8 -203.7  -21.7  182.2 1201.3
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1627.757      4.634   351.2   <2e-16 ***
## y           -106.060      6.388   -16.6   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 293 on 8437 degrees of freedom
## Multiple R-squared:  0.03164,    Adjusted R-squared:  0.03152
## F-statistic: 275.6 on 1 and 8437 DF,  p-value: < 2.2e-16
```

```r
# 2. Normality
# Let's check the normality of the numerical variables
# Create a function to check the normality of the numerical variables
normality_test <- function(x) {
  # Perform Shapiro-Wilk test
    shapiro <-  shapiro.test(x)
    print(shapiro)
}
```

```r
# Loop over numerical variables and perform Shapiro-Wilk test
for (i in c("white_rating", "black_rating")) {
  print(i)
#   sample 5000 rows and perform the test
    normality_test(sample(chess[[i]], 5000))
}
```

```
## [1] "white_rating"
##
##  Shapiro-Wilk normality test
##
## data:  x
## W = 0.98783, p-value < 2.2e-16
##
```

```
## [1] "black_rating"
##
##  Shapiro-Wilk normality test
##
## data:  x
## W = 0.99141, p-value < 2.2e-16
```

b. Model training

```r
# Model training
train_model <- function(data,verbose = FALSE) {
  # Split the data into train and test
  train_index <- createDataPartition(data$winner, p = 0.7, list = FALSE)
  train <- data[train_index, ]
  test <- data[-train_index, ]
  # Train the model
  model <- glm(winner ~ ., data = train, family = "binomial")
  # Predict the test data
  pred <- predict(model, test)
  if (verbose) {
    print(model)
    print(summary(model))
    # Print the confusion matrix
    print(table(test$winner, pred > 0.5))
    # Print the accuracy
    print(mean(test$winner == (pred > 0.5)))

  }
  return(model)
}
```