PSTAT131 Final Project

Yifei Zhang

2022-05-11

Contents

Introduction 2

About This Project 2

About This Dataset 2

Begin tidying 3

Loading Packages and Data 3

Setting Seed and Data Spliting 5

Exploratory Data Analysis 5

Check Fairness of Data 5

Narrow Down Variables 6

Visualization 9

Modeling 15

Conclusion 15

Introduction

About This Project

If you recognize characters from the picture above, you can probably tell we are going to explore the E sport/ video games field in this project. We are looking more specifically into the determining factors for winning a match in a game with the notoriously toxic gaming community, League of Legends. There are many types of people in this world, and a lot of us fall into two categories, the The League of Legends player category, and the victims of the first category, the player's friend who get forced to watch them playing knowing they are going to lose and even have a temper tantrum afterwards sometimes category. I am a part of the latter group, and to avoid spending extra half an hour watching/playing a game that I know is going to lose, which will lead us to a bad mental state, our friendships on the edge, I want to make a model that can predict the game result as accurately as possible given the first ten minutes game play statistics. Or at least get to know what the most important factors in winning a match are.

About This Dataset

The League of Legends Diamond Ranked Games dataset includes the first ten minutes statistics of approximately ten thousands ranked League of Legends matches (solo queue) ranging from diamond to master ranking. For background information, League of Legends is a multiplayer online battle arena (MOBA) game where there are 3 lanes, a jungle, and 5 player roles each for the 2 teams (blue and red). The first one to take down the enemy Nexus wins the game.

Here is some basic information about the The League of Legends Diamond Ranked Games dataset. The data is obtained from user MICHEL'S FANBOI who seems to have changed their username pretty frequently, on Kaggle, and their source is Riot Games, the developer of League of Legends, API. You can find the dataset following the link here https://www.kaggle.com/datasets/bobbyscience/league-of-legends-diamond-ranked-games-10-min?resource=download. In this dataset there are 9879 observations, and 38 predictors in total.

Begin tidying

Loading Packages and Data

```
library(ggplot2)
library(tidyverse)
library(tidymodels)
library(corrplot)
library(ggthemes)
library(discrim)
library(poissonreg)
library(corrr)
library(klaR)
library(ISLR)
library(ISLR2)
library(purrr)
library(janitor)
tidymodels_prefer()
```

```
loldata <- read_csv("DATA/high_diamond_ranked_10min.csv")
```

Before tidying we want to make sure we are not working with a significant amount of missing data. If we do, we want to make sure to tidy the records with significant amount of missing entries out for better accuracy.

```
is.null(loldata)
```

```
## [1] FALSE
```

Since we do not have any null entries, wee can directly move on to the next part, normally we would need to deselect some rows or fill in the null with zeros before moving on.

Although from a glance the variable names look pretty unique and not problem causing, we want to use clean the names to avoid potential problems in the future, such as forgetting to capitalize certain letters in the variable name.

```
# save the cleaned data
lol <- clean_names(loldata)

# print the new names
# for later variable selection purpose, also makes life easier
colnames(lol)
```

```
##  [1] "game_id"                          "blue_wins"
##  [3] "blue_wards_placed"                "blue_wards_destroyed"
##  [5] "blue_first_blood"                 "blue_kills"
##  [7] "blue_deaths"                      "blue_assists"
##  [9] "blue_elite_monsters"              "blue_dragons"
## [11] "blue_heralds"                     "blue_towers_destroyed"
## [13] "blue_total_gold"                  "blue_avg_level"
## [15] "blue_total_experience"            "blue_total_minions_killed"
## [17] "blue_total_jungle_minions_killed" "blue_gold_diff"
## [19] "blue_experience_diff"            "blue_cs_per_min"
## [21] "blue_gold_per_min"               "red_wards_placed"
## [23] "red_wards_destroyed"             "red_first_blood"
## [25] "red_kills"                       "red_deaths"
## [27] "red_assists"                     "red_elite_monsters"
## [29] "red_dragons"                     "red_heralds"
## [31] "red_towers_destroyed"            "red_total_gold"
## [33] "red_avg_level"                   "red_total_experience"
## [35] "red_total_minions_killed"        "red_total_jungle_minions_killed"
## [37] "red_gold_diff"                   "red_experience_diff"
## [39] "red_cs_per_min"                  "red_gold_per_min"
```

Since there are only two teams, Blue and Red, and many of the variables are coded in 1 and 0 that represents either blue or red got it just in the opposite way, a lot of them are repetitive to look at. For example, there are if the entry for our blue_wins is 0, then we know the corresponding entry for red_wins is 1. So we want to deselect some repetitive variables from our data set. It does not matter if blue or red wins, if blue loses then obviously red wins. In this case, we will work on classifying if team blue wins or not.

```
lol_blue <-  lol[ , 0:21]
colnames(lol_blue) # check if we have the right columns
```

```
##  [1] "game_id"                         "blue_wins"
##  [3] "blue_wards_placed"               "blue_wards_destroyed"
##  [5] "blue_first_blood"                "blue_kills"
##  [7] "blue_deaths"                     "blue_assists"
##  [9] "blue_elite_monsters"             "blue_dragons"
## [11] "blue_heralds"                    "blue_towers_destroyed"
## [13] "blue_total_gold"                 "blue_avg_level"
## [15] "blue_total_experience"           "blue_total_minions_killed"
## [17] "blue_total_jungle_minions_killed" "blue_gold_diff"
## [19] "blue_experience_diff"            "blue_cs_per_min"
## [21] "blue_gold_per_min"
```

For this part please take a look at the Exploratory Data Analysis section first. We want to extract the variables that appears to be significantly correlated with blue_wins.

```
important <- lol_blue[c("game_id", "blue_wins", "blue_first_blood",
                   "blue_kills",  "blue_deaths",
                   "blue_assists","blue_elite_monsters",
                   "blue_dragons", "blue_total_gold",
                   "blue_avg_level", "blue_total_experience",
                   "blue_gold_diff", "blue_experience_diff",
                   "blue_total_minions_killed", "blue_cs_per_min",
                   "blue_gold_per_min")]
```

```
important
```

```
## # A tibble: 9,879 x 16
##       game_id blue_wins blue_first_blood blue_kills blue_deaths blue_assists
##         <dbl>     <dbl>            <dbl>      <dbl>       <dbl>        <dbl>
##   1 4519157822         0                1          9           6           11
##   2 4523371949         0                0          5           5            5
##  3 4521474530         0                0          7          11             4
##  4 4524384067         0                0          4           5             5
##  5 4436033771         0                0          6           6             6
##  6 4475365709         1                0          5           3             6
##  7 4493010632         1                1          7           6             7
##  8 4496759358         0                0          5          13             3
##  9 4443048030         0                0          7           7             8
## 10 4509433346         1                1          4           5             5
## # ... with 9,869 more rows, and 10 more variables: blue_elite_monsters <dbl>,
## #   blue_dragons <dbl>, blue_total_gold <dbl>, blue_avg_level <dbl>,
## #   blue_total_experience <dbl>, blue_gold_diff <dbl>,
## #   blue_experience_diff <dbl>, blue_total_minions_killed <dbl>,
## #   blue_cs_per_min <dbl>, blue_gold_per_min <dbl>
```

Setting Seed and Data Spliting

The year is 2022 so I am setting the seed to be 2022. It is easy to remember, and it is not too small.

The data is split with a 75% training, 25% testing split. Stratified with blue_wins.

```
set.seed(2022)

lol_split <- lol %>%
  initial_split(strata = blue_wins, prop = 0.75)
lol_train <- training(lol_split)
lol_test <- testing(lol_split)

dim(lol_train) # check if we have the right proportion
```

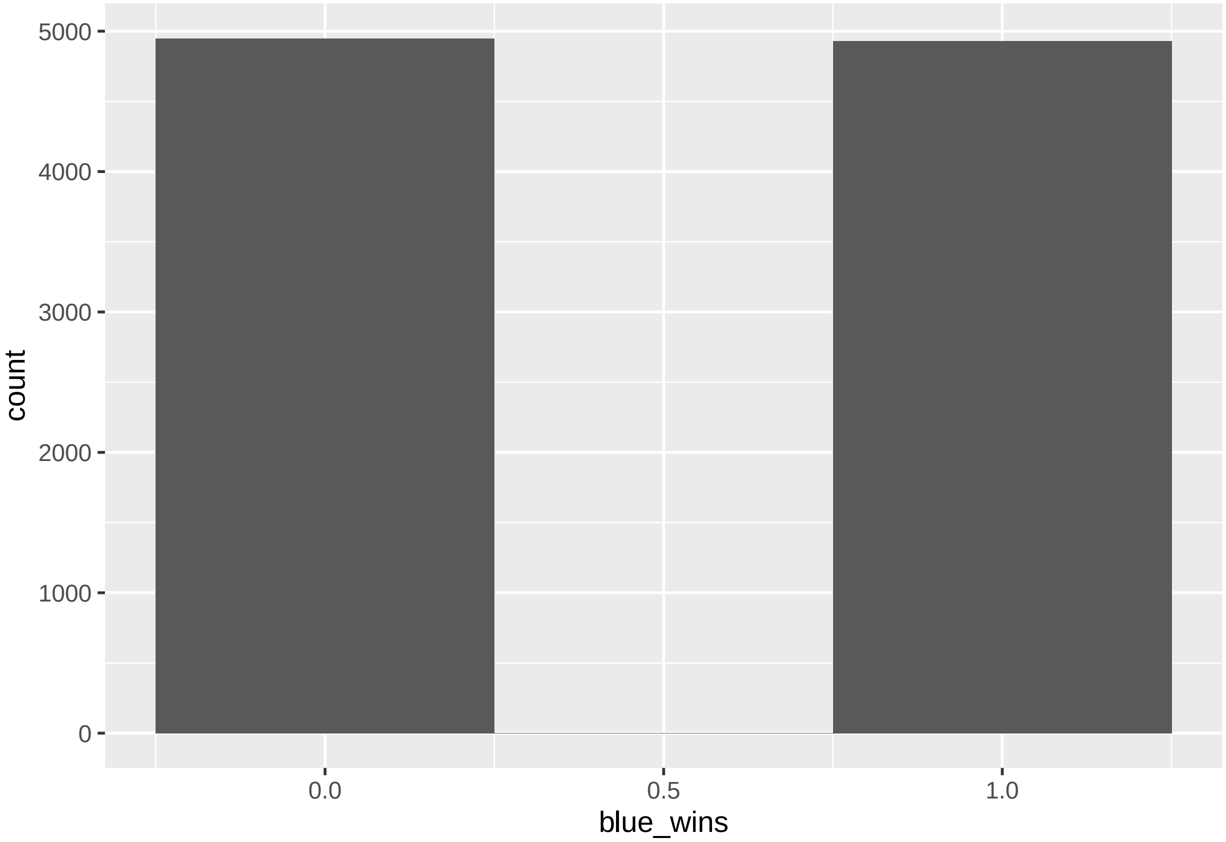
```
## [1] 7408   40
```

Exploratory Data Analysis

Check Fairness of Data

From the result we can see there is a very slight difference(insignificant) between the number of blue win and lose

```
lol %>%
  ggplot(aes(x = blue_wins)) +
  geom_histogram(bins = 3)
```

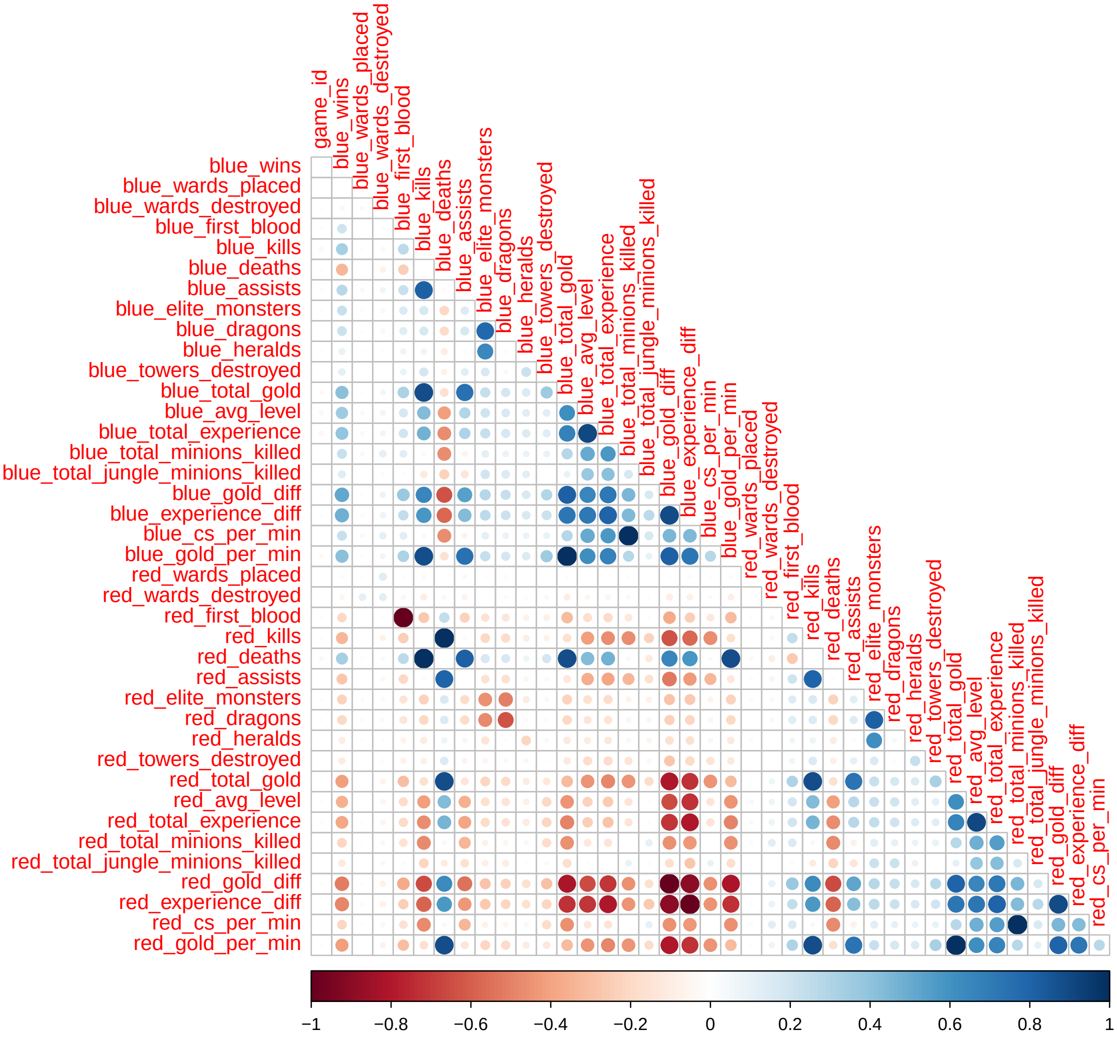


Narrow Down Variables

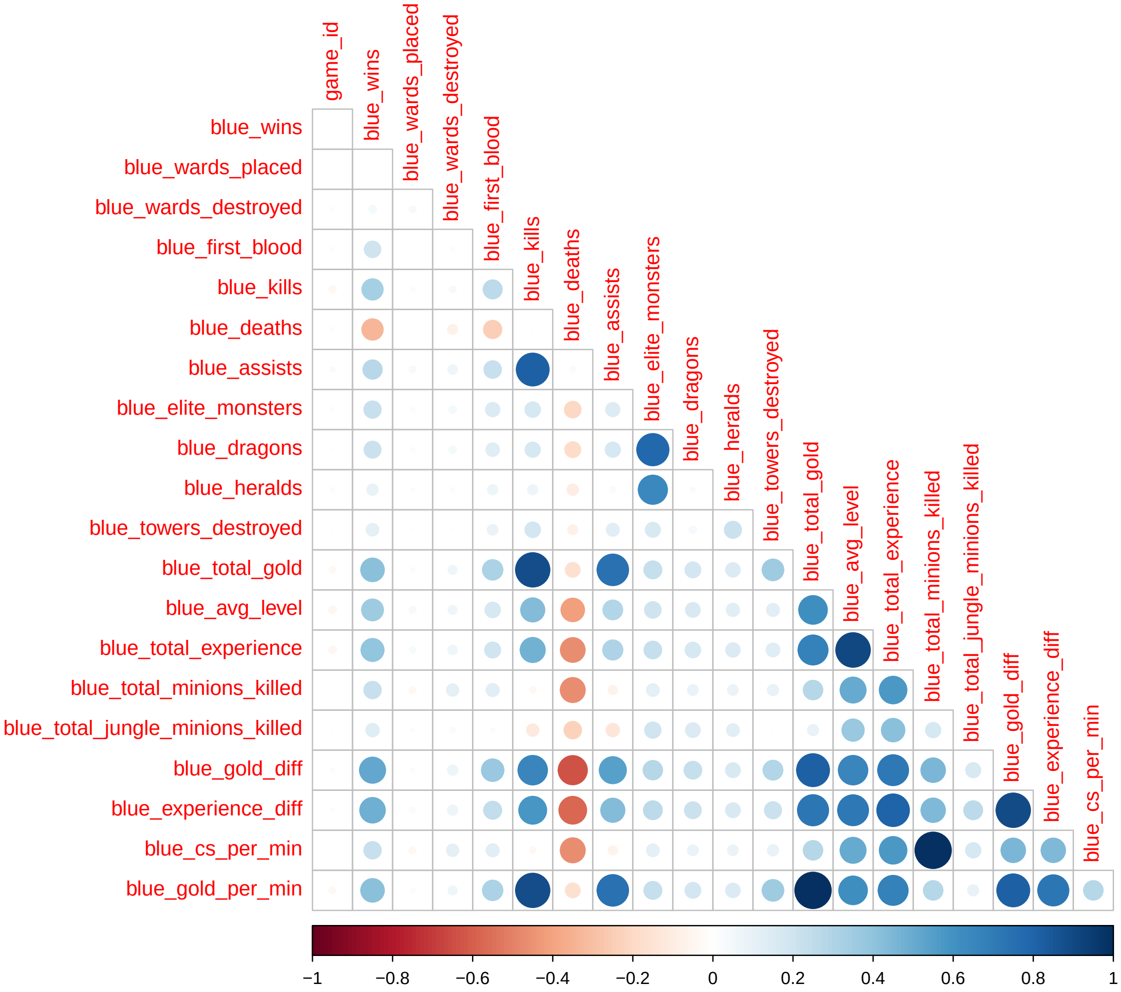
Here we are checking the correlation between all the variables, but we specifically need to pay more attention to what is correlated with blue_wins.

From the result of this auto-plot, we can see there are variables that correlate with blue_wins at about the same scale but in totally opposite ways, which proves our assumption that there are repetitive variables to be correct.

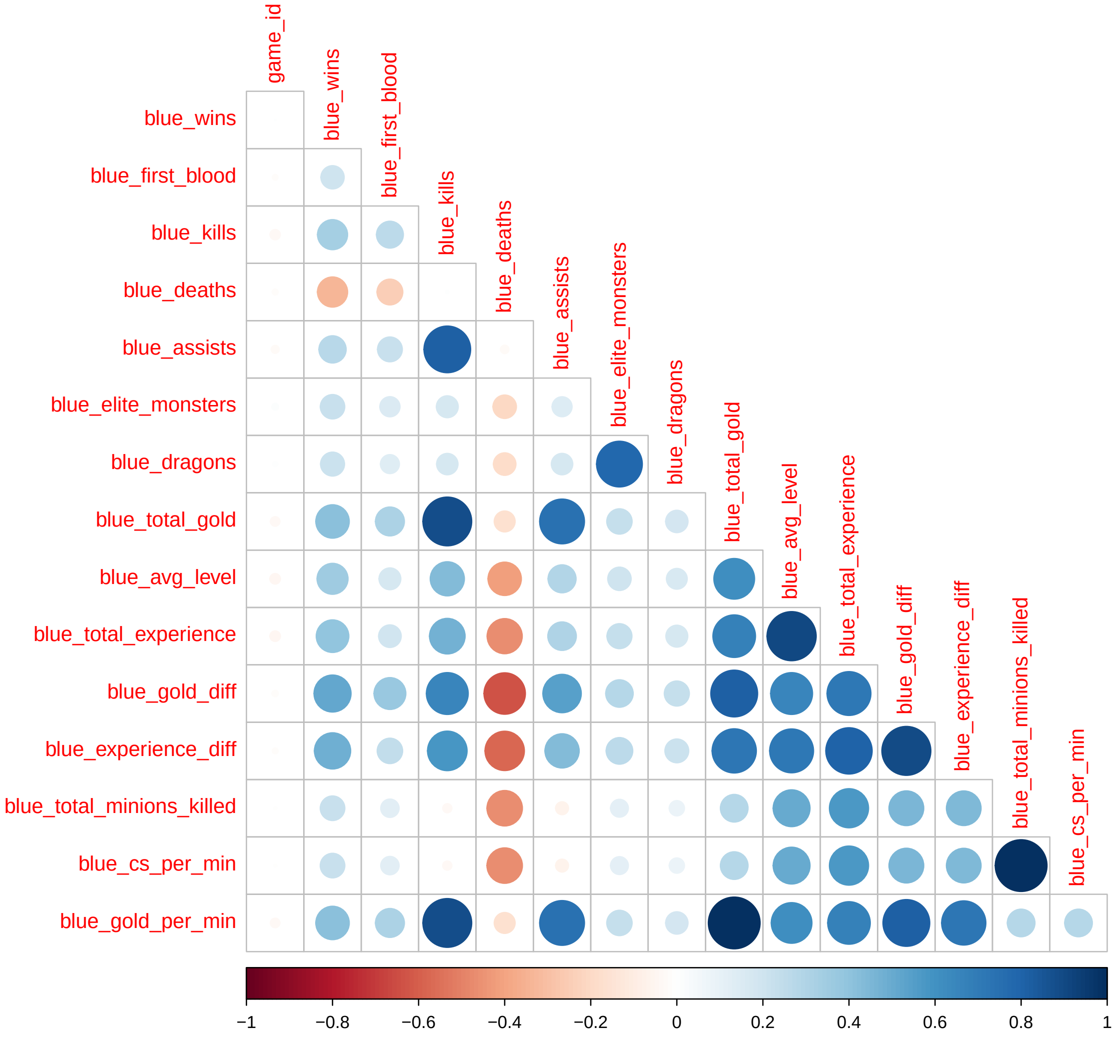
```
lol %>%
  select(is.numeric) %>%
  cor(use = "complete.obs") %>%
  corrplot(type = "lower", diag = FALSE)
```



```
lol_blue %>%
  select(is.numeric) %>%
  cor(use = "complete.obs") %>%
  corrplot(type = "lower", diag = FALSE)
```



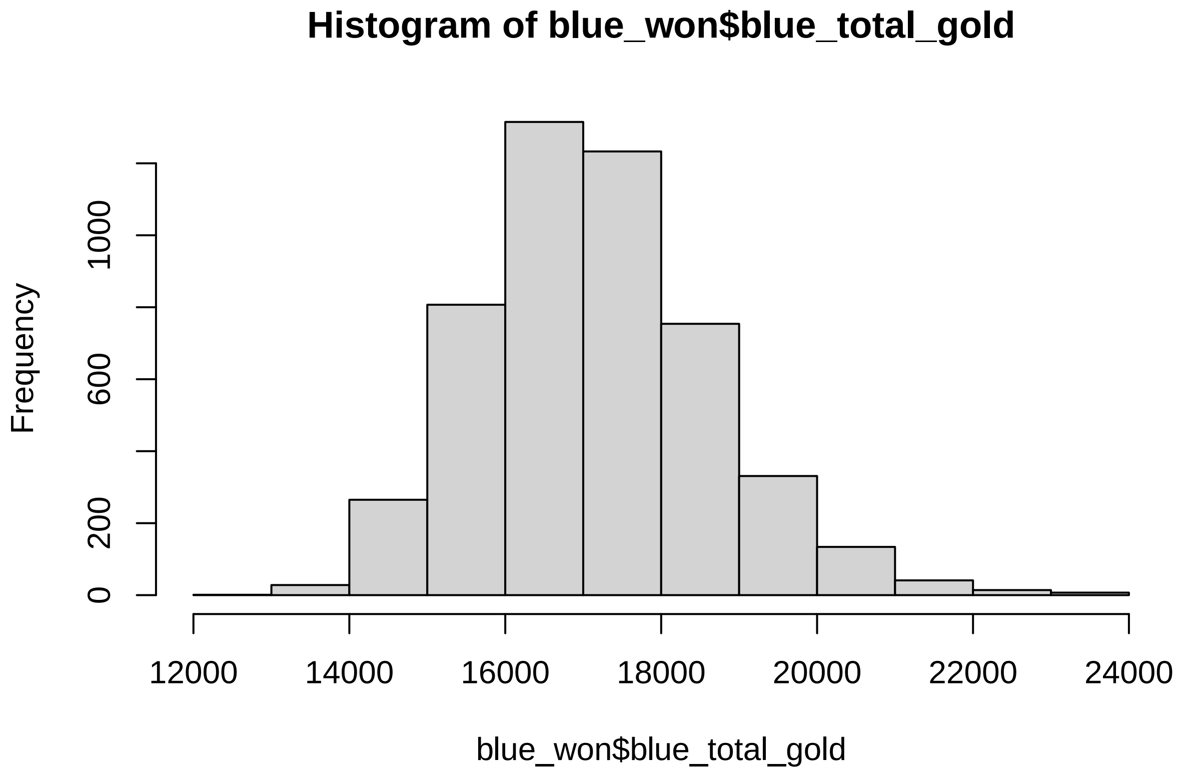
```
important %>%
  select(is.numeric) %>%
  cor(use = "complete.obs") %>%
  corrplot(type = "lower", diag = FALSE)
```



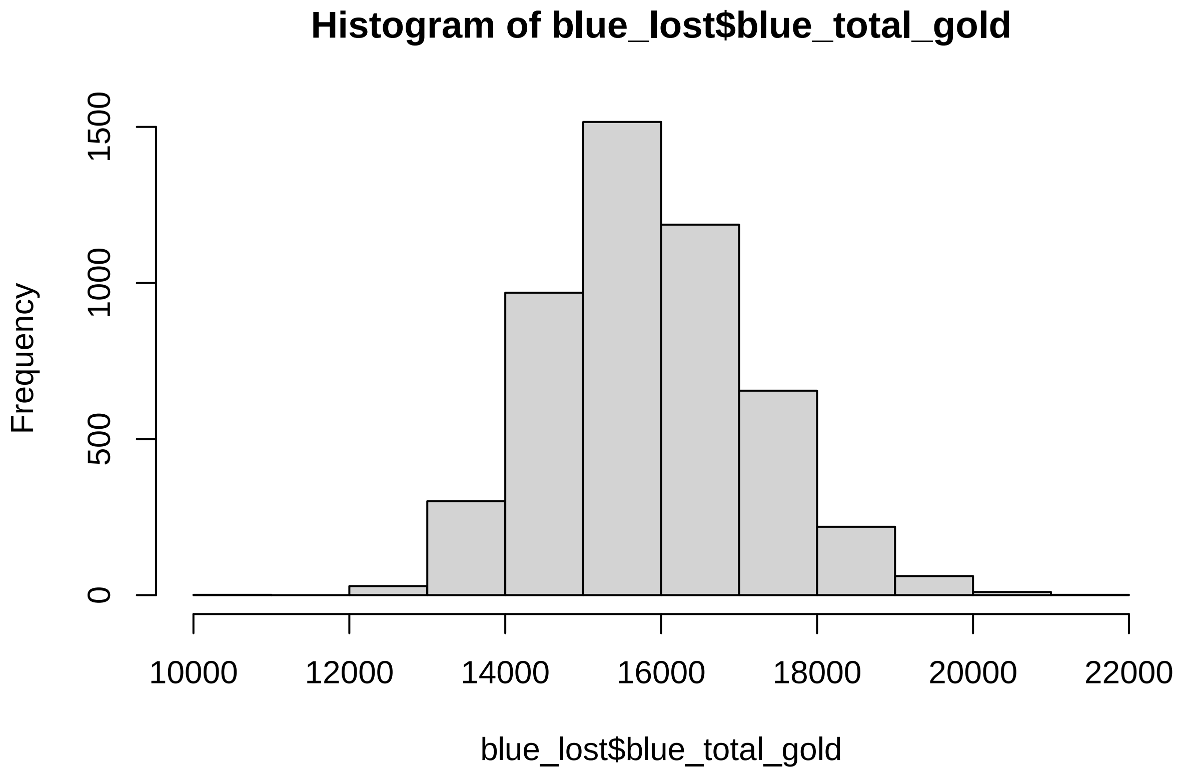
Visualization

From the Previous EDA results we know two of the most important factors that are highly correlated with *blue_wins* are *blue_total_gold* and *blue_total_experience*. We want to visualize the distribution difference between winning and losing. For the following plots, red represents red wins/ blue loses, blue represents blue wins/red loses.

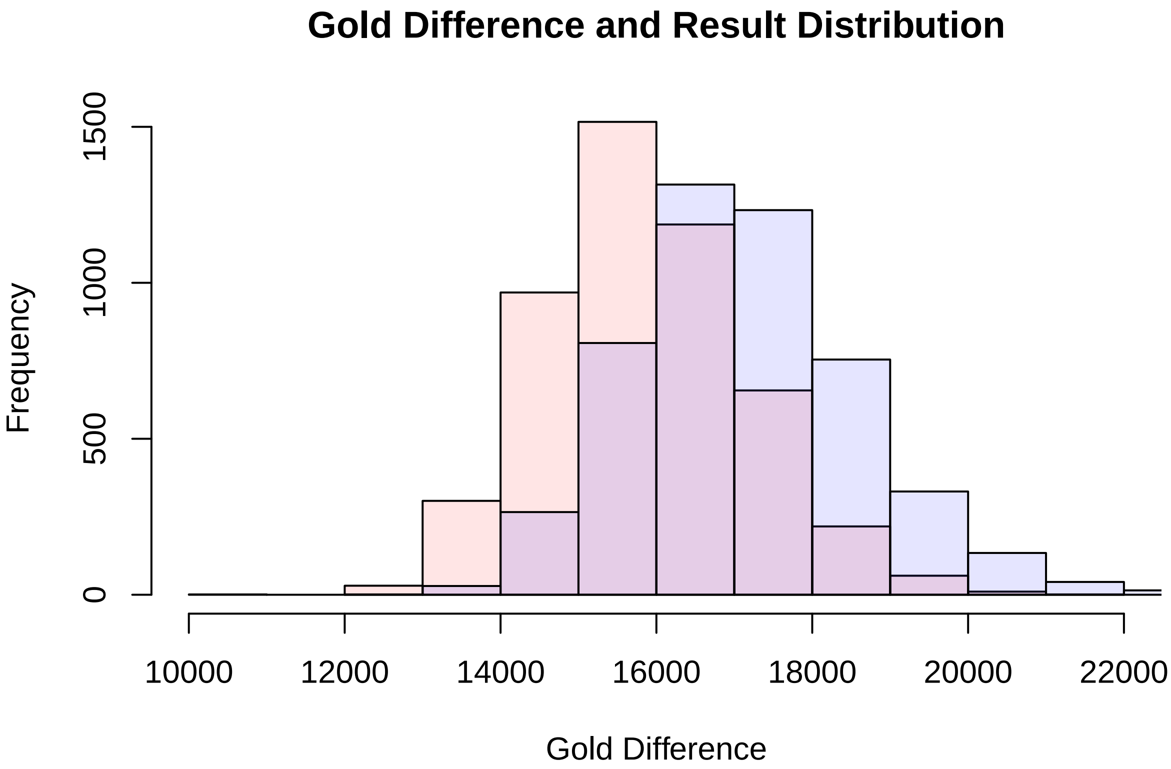
```
# separate the data into two
blue_won <- lol[lol $ blue_wins == '1', ]
blue_lost <- lol[lol $ blue_wins == '0', ]

#begin plotting
won_gold <- hist(blue_won $ blue_total_gold )
```



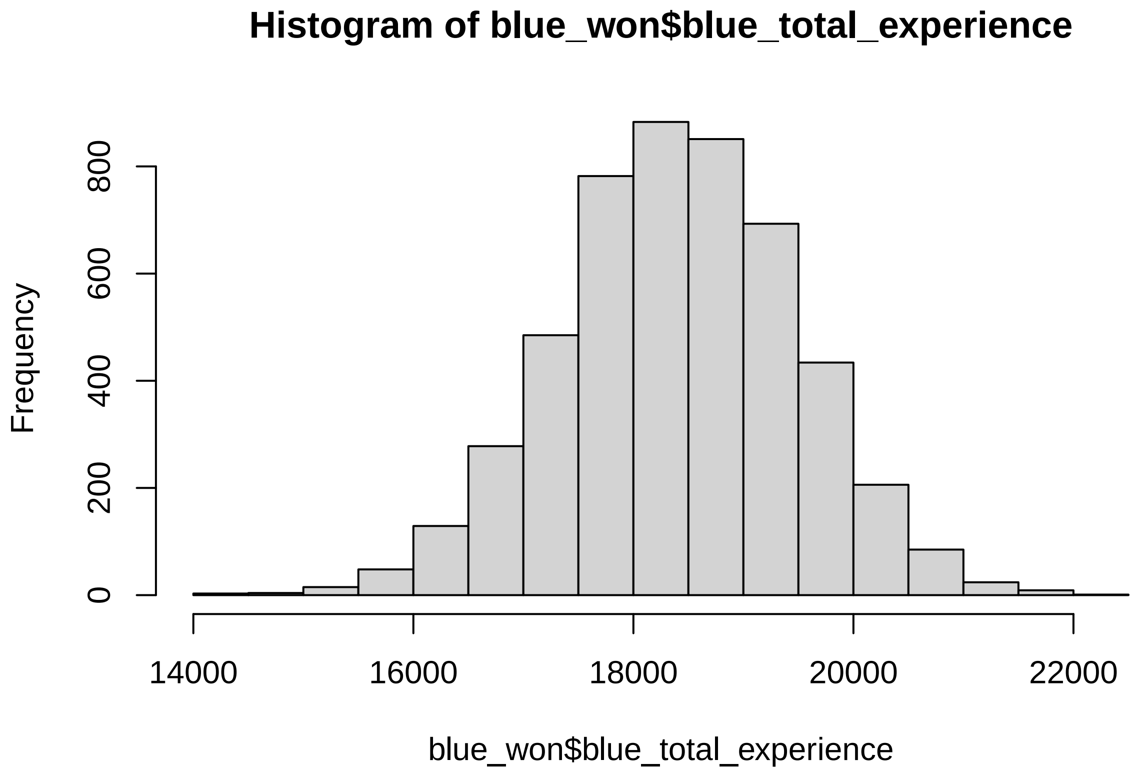
```
lost_gold <- hist(blue_lost $ blue_total_gold)
```



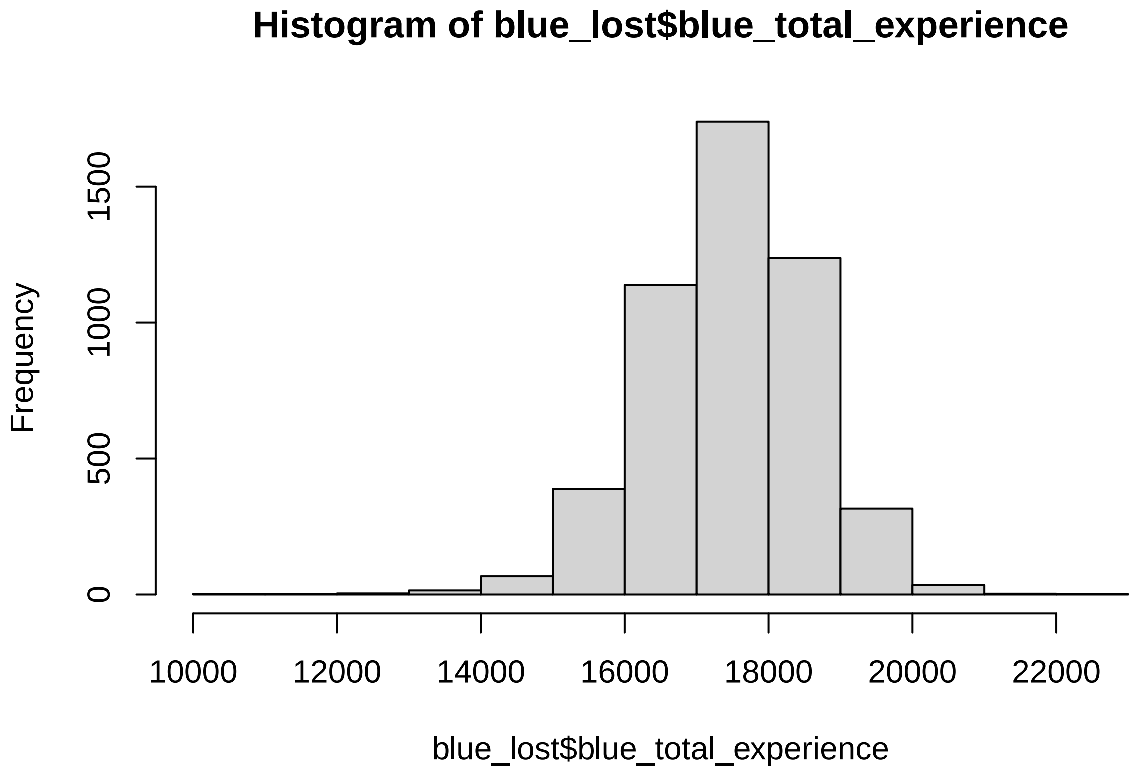
```
#put them together
plot(lost_gold, col = rgb(1, 0, 0, 0.1),
     main = "Gold Difference and Result Distribution",
     xlab = "Gold Difference")
plot(won_gold, col = rgb(0, 0, 1, 0.1), add = TRUE)
```



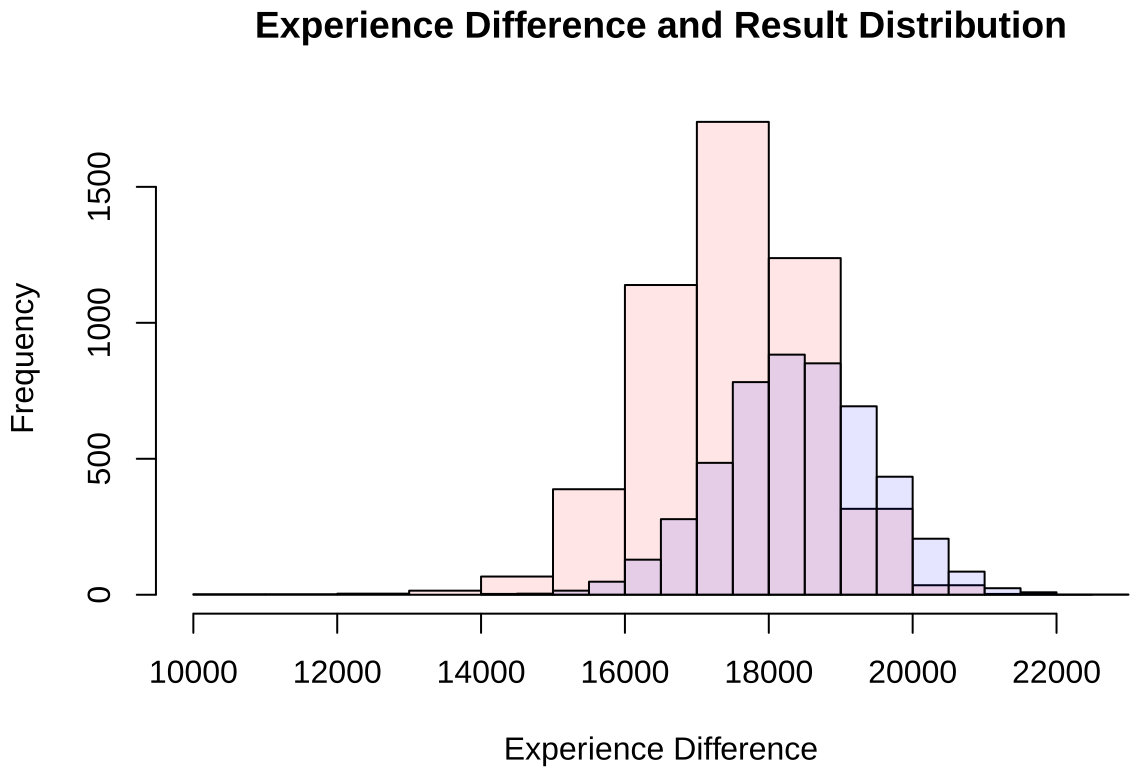
```
#begin plotting
won_exp <- hist(blue_won $ blue_total_experience)
```



```
lost_exp <- hist(blue_lost $ blue_total_experience)
```



```
#put them together
plot(lost_exp, col = rgb(1, 0, 0, 0.1),
     main = "Experience Difference and Result Distribution",
     xlab = "Experience Difference")
plot(won_exp, col = rgb(0, 0, 1, 0.1), add = TRUE)
```



Modeling

```r
lol_folds <- vfold_cv(lol_train, v = 5, strata = 'blue_wins')
lol_folds
```

```
## #  5-fold cross-validation using stratification
## # A tibble: 5 x 2
##   splits              id
##   <list>              <chr>
## 1 <split [5925/1483]> Fold1
## 2 <split [5926/1482]> Fold2
## 3 <split [5927/1481]> Fold3
## 4 <split [5927/1481]> Fold4
## 5 <split [5927/1481]> Fold5
```

Conclusion