Project 1 - Proposal and Data Assembly Take 2

Andrew Kerr

2023-10-15

# Data Collection and Cleaning

Using my program, I provide the tier I desire, and my program acquires 205 randomly selected players from the tier and adds their 5 most recently played matches to my database. For each match I collect information on all 8 players and how they performed in the match. For the highest tier (Challenger), I was able to collect the 5 most recent matches of all players in this tier (n = 250).

The variables I have decided to include are:

* placement: a measure of how well a player performed in a match compared to other players in the match
* level: a measure of how well strong a player was in a match
* last\_round: what round a player is eliminated from a match
* gold\_left: amount of gold a player had when the were eliminated from a match
* augment\_1-3: game modifier selected by a player during the match
* units: units a player was using when they were eliminated
* game\_length: how long (in seconds) a match lasted
* is\_challenger: True if the player is Challenger tier, False if not

## Augments

To start I need to explain a bit more about the augment variables. All players select 3 augments throughout each match, and there are two types of augments: normal augments and legend augments. Legend augments are tailored towards different play styles, and are chosen by a player prior to starting a match. Riot’s API provides us the name of each augment a player chooses, unless it was a legend augment, in which case the API returns NA. However, there is a small chance that no augments are offered in a given match, and in this case the API returns NA for all augments for all players in the match.

A new feature I have decided to create is the number of legend augments a player chose during their match. Since I can not distinguish legend augments from a no augment match, I have dropped matches where all players have NA for all three augments (this signifies that either everyone chose a legend augment 3 times, or no augments were offered that match, the former being more unlikely than the latter; -8 observations).

remove\_matches <- df\_TFT %>%  
 group\_by(match\_id) %>%  
 filter(is.na(augment\_1), is.na(augment\_2), is.na(augment\_3)) %>%  
 count() %>%  
 filter(n == 8) %>%  
 select(match\_id)  
  
# remove\_matches %>% nrow()  
  
df\_TFT\_clean <- df\_TFT %>%  
 filter(!(match\_id %in% remove\_matches)) %>%  
 group\_by(match\_id, name) %>%  
 mutate(n\_legend = sum(  
 if\_else(last\_round >= 5, is.na(augment\_1), 0)  
 + if\_else(last\_round >= 13, is.na(augment\_2), 0)  
 + if\_else(last\_round >= 20, is.na(augment\_3), 0)  
 ))

## Units

Each unit a player has on their board has a cost of 1, 2, 3, 4, or 5 and can be level 1, 2, or 3. If you have 3 of the same level 1 units, it becomes a level 2, and if you have 3 of the same level 2 units it becomes level 3 (e.g. the cost of a level 2, 3 cost, unit is 6 gold). Two ways to measure how strong a players board is is by looking at average unit level and total board cost. The following aggregates the units feature (currently a json) to be the average unit level and total board cost the player ended the match with.

Additionally, since there is great variability in amount of resources (gold, free units, etc.) given to players from match to match, I have created features for average unit level and average board cost of all players in a match (level 2 features).

df\_units <- lapply(df\_TFT\_clean$units, fromJSON)  
  
avg\_units <- sapply(df\_units, function(units) {  
 if (length(units) > 0) {  
 return(mean(units$tier))  
 }   
 return(0)  
})  
  
board\_costs <- sapply(df\_units, function(units) {  
 if (length(units) > 0) {  
 cost <- units %>%  
 filter(character\_id != 'TFT9\_HeimerdingerTurret') %>%  
 # The rarity (cost) is off from the API results...  
 mutate(costs = ifelse(rarity < 3, rarity + 1, ifelse(rarity == 6, rarity - 1, rarity)) \* tier)  
 return(sum(cost$costs))  
 }   
 return(0)  
})  
  
df\_TFT\_clean$avg\_unit\_level <- avg\_units # level 1  
df\_TFT\_clean$board\_cost <- board\_costs # level 1  
df\_TFT\_clean <- df\_TFT\_clean %>%   
 group\_by(match\_id) %>%  
 mutate(match\_avg\_unit\_level = mean(avg\_unit\_level),  
 match\_avg\_board\_cost = mean(board\_cost)) # 2 level 2 variables  
  
df\_TFT\_clean <- df\_TFT\_clean %>% select(-units)

## Independence

Since, within each match, knowing the results of one player can affect the probability of seeing certain results of another player (e.g. placement), I have decided to randomly sample from my random sample. I will only keep 3 random players per match.

set.seed(43)  
  
# select 2 Challenger players and 2 Non-Challenger players from each match  
# if 2 of one target group does not exist in a match, remove it  
df\_TFT\_clean\_sample <- df\_TFT\_clean %>%  
 group\_by(match\_id) %>%  
 slice\_sample(n = 3, replace = FALSE) %>%  
 group\_by(match\_id)  
   
df\_TFT\_clean\_sample %>%  
 group\_by(is\_challenger) %>%  
 select(is\_challenger, match\_id) %>%  
 count()

## # A tibble: 2 × 2  
## # Groups: is\_challenger [2]  
## is\_challenger n  
## <lgl> <int>  
## 1 FALSE 2656  
## 2 TRUE 2100

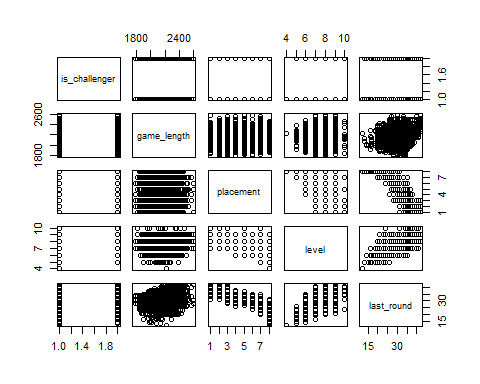
nrow(df\_TFT\_clean\_sample)

## [1] 4756

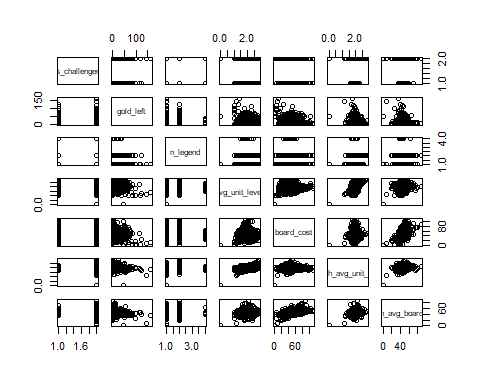
df\_TFT\_clean\_sample <- df\_TFT\_clean\_sample %>%  
 select(-name, -tft\_game\_type, -traits, -augment\_1, -augment\_2, -augment\_3) %>%  
 mutate(  
 game\_length = as.integer(game\_length),  
 level = as\_factor(level),  
 placement = as\_factor(placement),  
 n\_legend = as\_factor(n\_legend),  
 match\_id = as\_factor(match\_id),  
 is\_challenger = as\_factor(is\_challenger)  
 )  
  
df\_TFT\_clean\_sample$is\_challenger <- factor(df\_TFT\_clean\_sample$is\_challenger, levels=c(TRUE, FALSE), labels=c('Challenger', 'Master/Grandmaster'))  
df\_TFT\_clean\_sample$level <- factor(df\_TFT\_clean\_sample$level, levels=c('1', '2', '3', '4', '5', '6', '7', '8', '9', '10'))  
df\_TFT\_clean\_sample$placement <- factor(df\_TFT\_clean\_sample$placement, levels=c('1', '2', '3', '4', '5', '6', '7', '8'))  
df\_TFT\_clean\_sample$n\_legend <- factor(df\_TFT\_clean\_sample$n\_legend, levels=c('0', '1', '2', '3'))

# Exploration

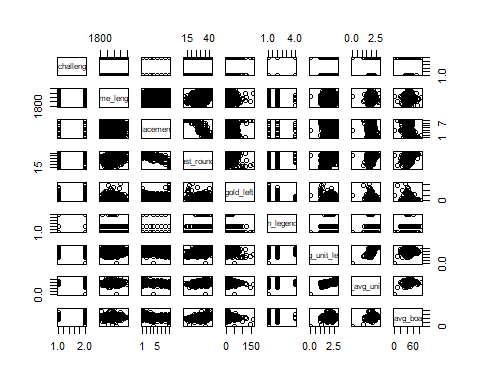
plot(df\_TFT\_clean\_sample[c(1, 3, 4, 5, 6)])



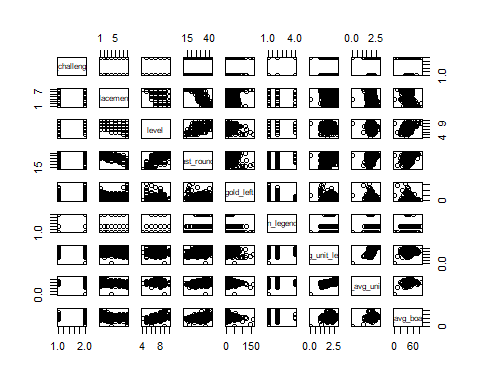
plot(df\_TFT\_clean\_sample[c(1, 7, 8, 9, 10, 11, 12)])



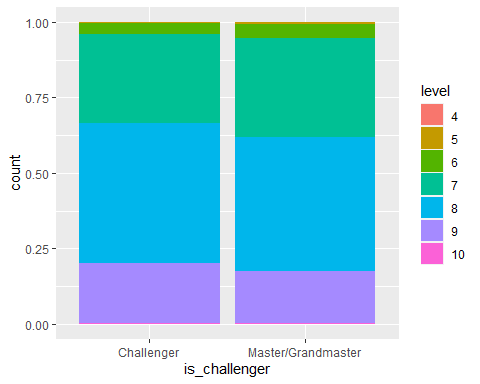
plot(df\_TFT\_clean\_sample[c(1, 3, 4, 6, 7, 8, 9, 11, 12)])



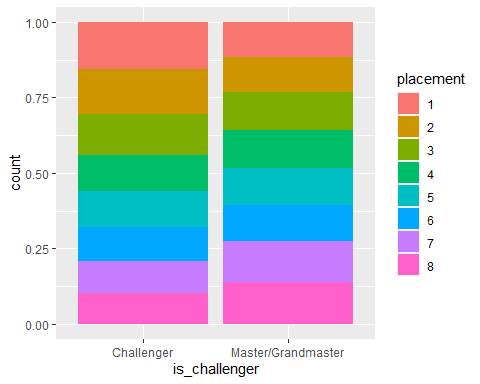
plot(df\_TFT\_clean\_sample[c(1, 4, 5, 6, 7, 8, 9, 11, 12)])



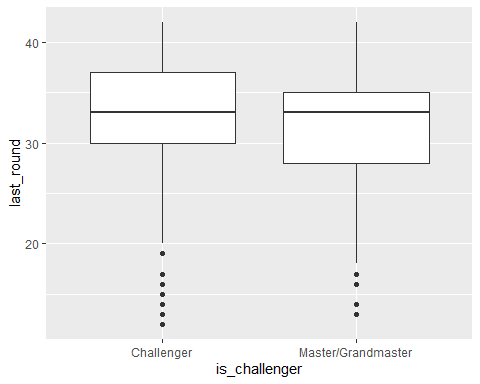
df\_TFT\_clean\_sample %>%  
 ggplot() +  
 geom\_bar(aes(x=is\_challenger, fill=level), position='fill')



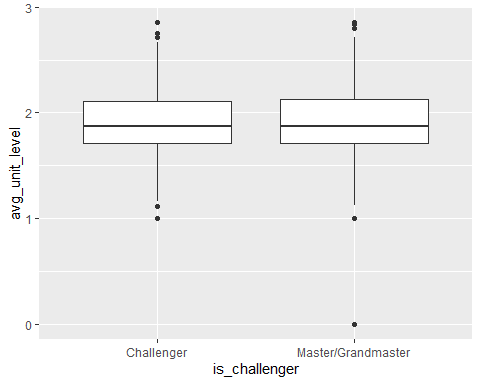
df\_TFT\_clean\_sample %>%  
 ggplot() +  
 geom\_bar(aes(x=is\_challenger, fill=placement), position='fill')



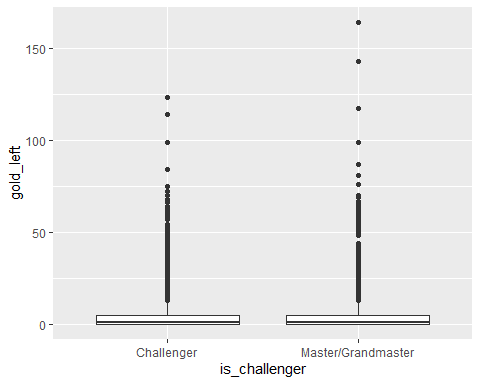
df\_TFT\_clean\_sample %>%  
 ggplot() +  
 geom\_boxplot(aes(x=is\_challenger, y=last\_round))



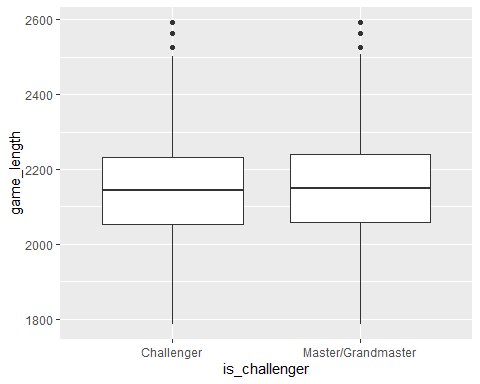
df\_TFT\_clean\_sample %>%  
 ggplot() +  
 geom\_boxplot(aes(x=is\_challenger, y=avg\_unit\_level))



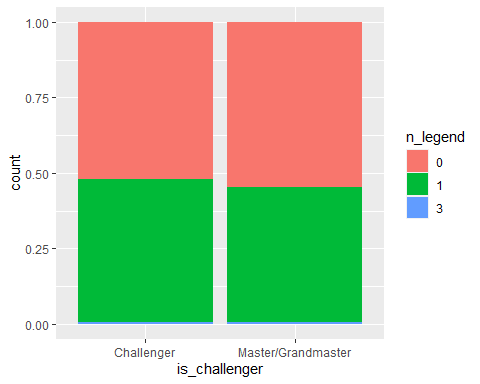
df\_TFT\_clean\_sample %>%  
 ggplot() +  
 geom\_boxplot(aes(x=is\_challenger, y=gold\_left))



df\_TFT\_clean\_sample %>%  
 ggplot() +  
 geom\_boxplot(aes(x=is\_challenger, y=game\_length))



df\_TFT\_clean\_sample %>%  
 ggplot() +  
 geom\_bar(aes(x=is\_challenger, fill=n\_legend), position='fill')



write.csv(df\_TFT\_clean\_sample, "TFT\_data.csv", row.names = FALSE)