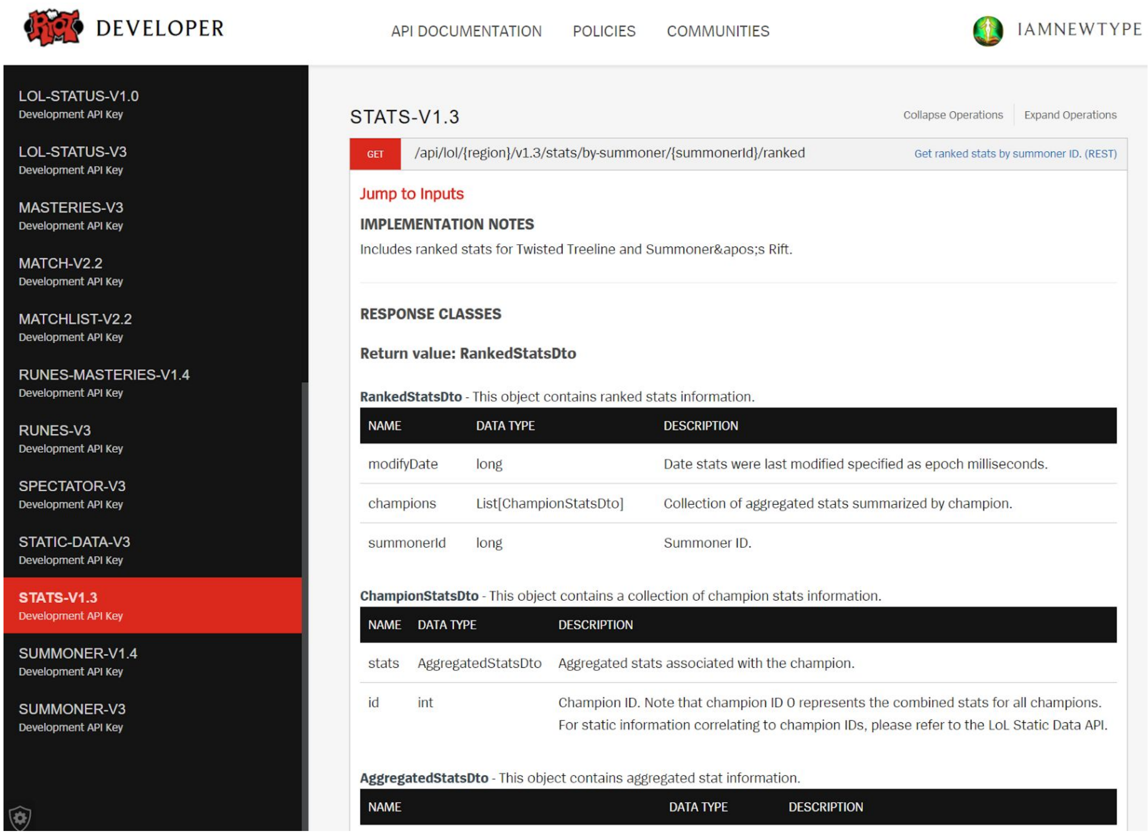
Project Journal

Proposal and background

For this project, I wanted to make a logistic regression model to see if I can predict if a player is ranked in Challenger or Master in the video game League of Legends. In this video game, all players were ranked in leagues in the order of bronze, silver, gold, platinum, diamond, and challenger. A year ago, the developers added a league between diamond and challenger called master. A lot of players were not happy because they felt like the ranking system got skewed and players rank got pushed down. Based on master and challenger players' ranked game stats, I would like to make a logistic model to classify if a player would be in the master league or the challenger league. I would be using the game's official developer API to obtain data. For data cleaning and shaping, I will be using R. For data storage and retrieval, I would be using a non relational database, MongoDB. Below is the journal of the process, and there is a final code run of everything at the end as well.

Data Acquisition

Originally I wanted to do a different video game, called Overwatch. However, there was no official API, so I had go around the internet and forums to see how the existing third party Overwatch stats websites were able to do it. I found an unofficial api on github, and I was able to retrieve individual player's stats from player's game tag id. However, I could not figure out a way to obtain a list of all the players' game tag id within the same rankings. I originally started this data acquisition in Python as well, however I ran into some problems retrieving data from the given api. After a couple hours of frustrations, I switched another game, League of Legend, which has an official API and is much easier to use and retrieve different sorts of data.



STATS-V1.3

GET /api/lol/{region}/v1.3/stats/by-summoner/{summonerId}/ranked — Get ranked stats by summoner ID. (REST)

IMPLEMENTATION NOTES

Includes ranked stats for Twisted Treeline and Summoner&apos;s Rift.

RESPONSE CLASSES

Return value: RankedStatsDto

RankedStatsDto - This object contains ranked stats information.

NAME	DATA TYPE	DESCRIPTION
modifyDate	long	Date stats were last modified specified as epoch milliseconds.
champions	List[ChampionStatsDto]	Collection of aggregated stats summarized by champion.
summonerId	long	Summoner ID.

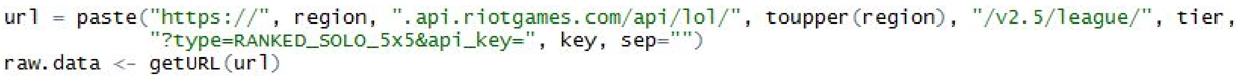
ChampionStatsDto - This object contains a collection of champion stats information.

NAME	DATA TYPE	DESCRIPTION
stats	AggregatedStatsDto	Aggregated stats associated with the champion.
id	int	Champion ID. Note that champion ID 0 represents the combined stats for all champions. For static information correlating to champion IDs, please refer to the LoL Static Data API.

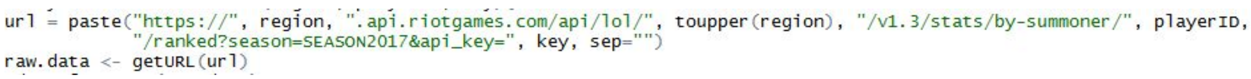
AggregatedStatsDto - This object contains aggregated stat information.

NAME	DATA TYPE	DESCRIPTION

The API also allows you to execute requests in the browser and see what the data return is like. I played around on the browser and tried retrieve all the different data to find what I wanted. After some digging around, I found the GET url for retrieving an individual’s ranked stats summary and the GET url for retrieving a list of player id within a league. The next step was retrieving the data in R. I spent some time trying to figure out how to retrieve the data in R and finally I installed the RCurl package.



```
url = paste("https://", region, ".api.riotgames.com/api/lol/", toupper(region), "/v2.5/league/", tier,
            "?type=RANKED_SOLO_5x5&api_key=", key, sep="")
raw.data <- getURL(url)
```

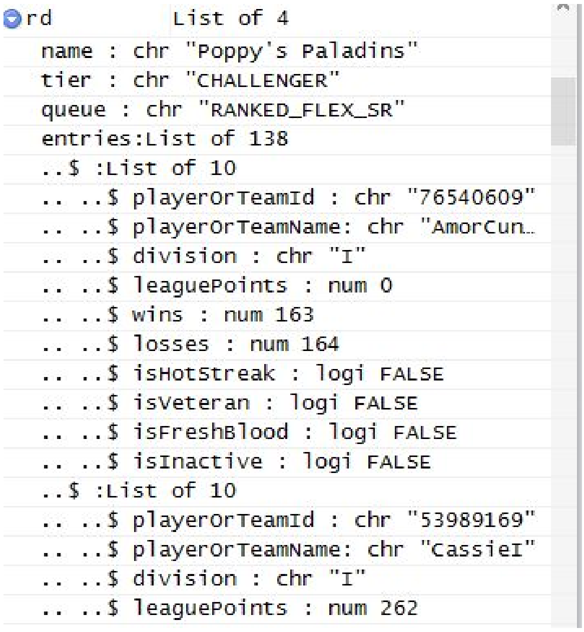


```
url = paste("https://", region, ".api.riotgames.com/api/lol/", toupper(region), "/v1.3/stats/by-summoner/", playerID,
          "/ranked?season=SEASON2017&api_key=", key, sep="")
raw.data <- getURL(url)
```

With RCurl, I was able to use getURL() to retrieve data from the API url, and parameterize the region, the rank tier, the API key and the player id. However, I spent a lot of time trying to figure out why this was not working when I tried to parameterize it. When I directly copied and paste the entire url, I was able to retrieve the data successfully. I thought it was something wrong with my functions, and I just could not figure it out. Eventually I realized that when I used the paste() function to add string and stored string variables, it created spaces in between and getURL does not get rid of the string by itself, so it was able to retrieve the data. I found out that I had to put sep = “” in paste to get rid of the spaces.

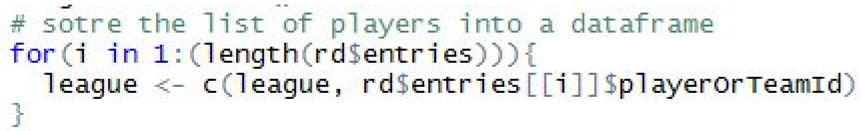
Data Cleaning & Shaping

After retrieving the raw data from the API, I was really confused about how to get the specific informations I wanted because the raw data format was in text characters. I then wasted a lot of time trying to find a pattern to parse the characters to store them into a dataframe without realizing it is all in JSON format. After realizing the character is in JSON format, I found the rjson package to parse the character and turn it into an organized list. Using the fromJSON() function, I was able to convert the data successfully. In the list format, I was able to navigate down the list and obtain what data I wanted much easier.



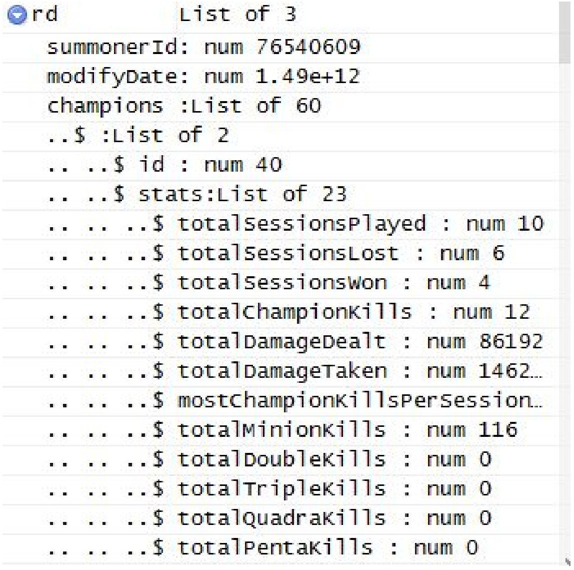
```
rd          List of 4
   name : chr "Poppy's Paladins"
   tier : chr "CHALLENGER"
   queue : chr "RANKED_FLEX_SR"
   entries:List of 138
   ..$ :List of 10
   .. ..$ playerOrTeamId : chr "76540609"
   .. ..$ playerOrTeamName: chr "AmorCun…
   .. ..$ division : chr "I"
   .. ..$ leaguePoints : num 0
   .. ..$ wins : num 163
   .. ..$ losses : num 164
   .. ..$ isHotStreak : logi FALSE
   .. ..$ isVeteran : logi FALSE
   .. ..$ isFreshBlood : logi FALSE
   .. ..$ isInactive : logi FALSE
   ..$ :List of 10
   .. ..$ playerOrTeamId : chr "53989169"
   .. ..$ playerOrTeamName: chr "CassieI"
   .. ..$ division : chr "I"
   .. ..$ leaguePoints : num 262
```

The data was stored in list of lists and after some trial and error, I figured out how to retrieve all the information needed from the list and storing it into a data frame using a for loop. This worked to retrieve the list of player id.



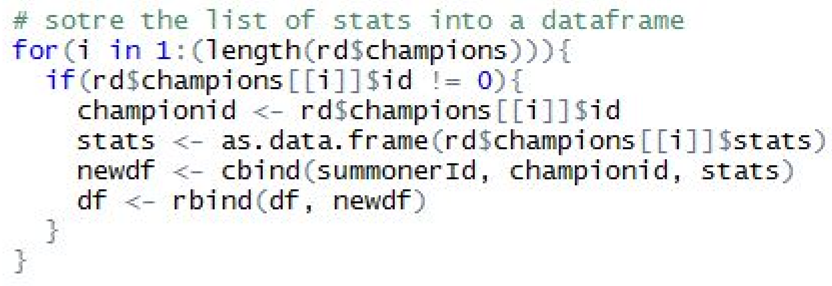
```
# sotre the list of players into a dataframe
for(i in 1:(length(rd$entries))){
  league <- c(league, rd$entries[[i]]$playerOrTeamId)
}
```

However, I ran into more problems when trying to do the same to retrieve each individual's player's stats because the list is more complicated than the league of players. For each player,



```
rd            List of 3
   summonerId: num 76540609
   modifyDate: num 1.49e+12
   champions :List of 60
   ..$ :List of 2
   .. ..$ id : num 40
   .. ..$ stats:List of 23
   .. .. ..$ totalSessionsPlayed : num 10
   .. .. ..$ totalSessionsLost : num 6
   .. .. ..$ totalSessionsWon : num 4
   .. .. ..$ totalChampionKills : num 12
   .. .. ..$ totalDamageDealt : num 86192
   .. .. ..$ totalDamageTaken : num 1462…
   .. .. ..$ mostChampionKillsPerSession…
   .. .. ..$ totalMinionKills : num 116
   .. .. ..$ totalDoubleKills : num 0
   .. .. ..$ totalTripleKills : num 0
   .. .. ..$ totalQuadraKills : num 0
   .. .. ..$ totalPentaKills : num 0
```

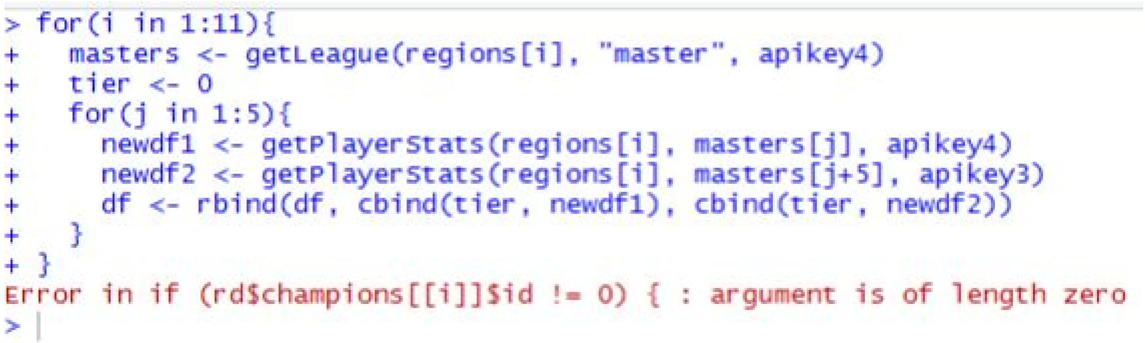
their stats is made of different entries based on different champions (in game characters the players use). The stats list is made of different stats based on champions and each entry has a champion id. Each character has dramatically different roles and play style, therefore might result dramatically different stats. I decided to include all the entries even they are all from the same player because I believed that the different champions used affect the stats as well and is an important variable. I then used a for loop to get all the stats and put them into a dataframe using r bind in a for loop. However, after scratching my head I could not figure out why this for loop would fail. I tried using the rbind() function to individually store the stats into a dataframe, and it seemed to be working. I thought the index was going out of bound but it was not the case either. I spent a lot of time trying to debug this and eventually found that in the stats list there is one entry with a champion id of 0 is causing the problem. All the entries had the same stats field, so I was able to rbind them after converting them into a dataframe without any problem. The problem occurs when the loop reaches the entry with the champion id of 0, rbind will not work becuase this entry has different stats fields than all of the other ones. This resulted an error of rbind() since the function can only combine two dataframes that have the same columns. I found out that champion id 0 is the total aggregate stats amongst all of the champions used. I decided to not use this because the different champions stats will make a difference and that is what I wanted only. So I added an if statement to make sure that it will not do an rbind() with the aggregate stats.



```
# sotre the list of stats into a dataframe
for(i in 1:(length(rd$champions))){
  if(rd$champions[[i]]$id != 0){
    championid <- rd$champions[[i]]$id
    stats <- as.data.frame(rd$champions[[i]]$stats)
    newdf <- cbind(summonerId, championid, stats)
    df <- rbind(df, newdf)
  }
}
```

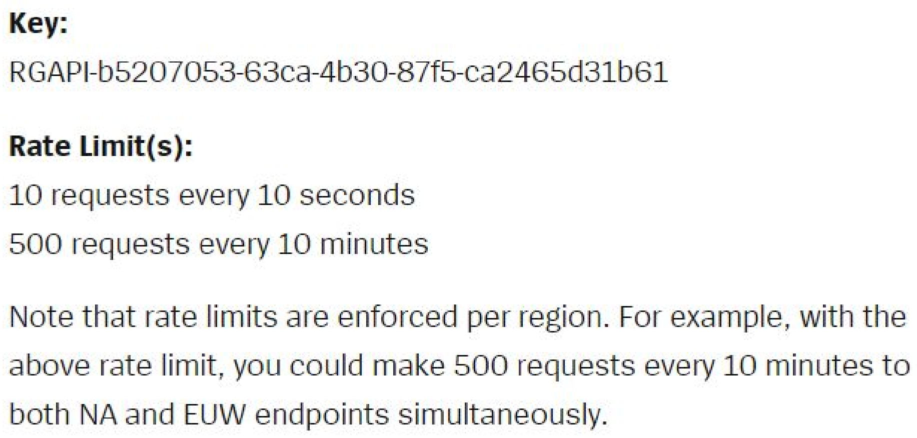
More on Data Acquisition, Cleaning and Shaping

The next step is to after being able to get a list of the players' ids within a league and an individual player ranked stats is to get all the player's stats on that list. I instinctively wrote a for loop to retrieve all the players' stats but it kept breaking on me. So I tried to debug it by trying line by line and found that instead of returning the players stats data, I was recieving error 404. On the API documentation, error 404 means that no stats found, which I had no idea why because my for loop seems perfect. I then found that this was caused by the GET url from the api. It turns out in the URL, you could parameterize the season too. The list of players I was retrieving was only in the most recent season, however, the individual player stats you have to select the season. I changed the season to the most recent season and the mismatch disappeared. The next problem I ran into gave me the most headache and I could not, for the life of me, figure out what was wrong. I tried doing different things and spent hours trying to identify the problem, I could not find any mistakes in the code. I just kept getting the error running the for loop.



```
> for(i in 1:11){
+   masters <- getLeague(regions[i], "master", apikey4)
+   tier <- 0
+   for(j in 1:5){
+     newdf1 <- getPlayerStats(regions[i], masters[j], apikey4)
+     newdf2 <- getPlayerStats(regions[i], masters[j+5], apikey3)
+     df <- rbind(df, cbind(tier, newdf1), cbind(tier, newdf2))
+   }
+ }
Error in if (rd$champions[[i]]$id != 0) { : argument is of length zero
> |
```

After spending hours of wondering what “Error in if (rd$champions[[i]]$id != 0) { : argument is of length zero” meant, I realized that the problem lies in the API key. Turns out there is a rate limit on requests per API key.

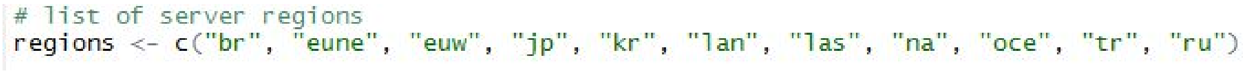


Key:
RGAPI-b5207053-63ca-4b30-87f5-ca2465d31b61

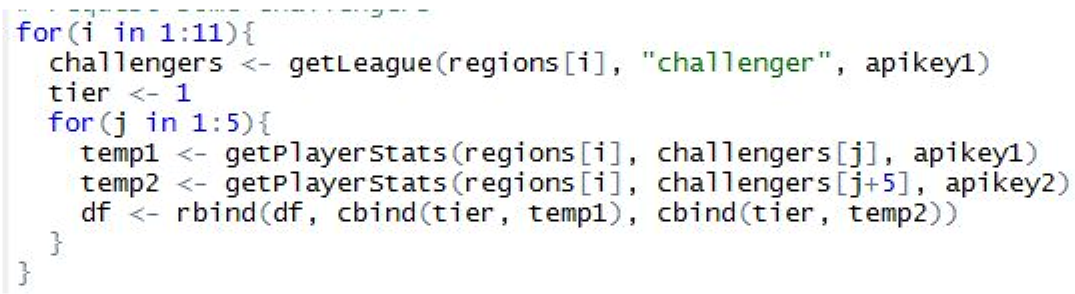
Rate Limit(s):
10 requests every 10 seconds
500 requests every 10 minutes

Note that rate limits are enforced per region. For example, with the above rate limit, you could make 500 requests every 10 minutes to both NA and EUW endpoints simultaneously.

To handles this problem, I first asked my friends accounts and got a couple different API keys to use in the for loop to get more data. And to maximize these keys I also made a nested for loop to go through each regions to get data because the rate limit applies to each regional endpoints simultaneously. I listed all of the different regional endpoints in the world.



```r
# list of server regions
regions <- c("br", "eune", "euw", "jp", "kr", "lan", "las", "na", "oce", "tr", "ru")
```

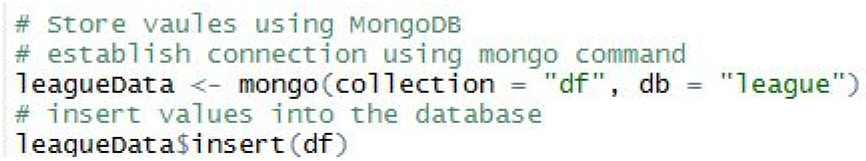


```r
for(i in 1:11){
  challengers <- getLeague(regions[i], "challenger", apikey1)
  tier <- 1
  for(j in 1:5){
    temp1 <- getPlayerStats(regions[i], challengers[j], apikey1)
    temp2 <- getPlayerStats(regions[i], challengers[j+5], apikey2)
    df <- rbind(df, cbind(tier, temp1), cbind(tier, temp2))
  }
}
```

I used a nested for loop to get players' data from all the regions and using different apikeys and it seemed to work the first time. However, it seems to not work from time to time and I still can not figure out the problem. I suspect that it might have something to do with the code getting the requests more than 10 requests in 10 seconds due to the variation in each player's data size.

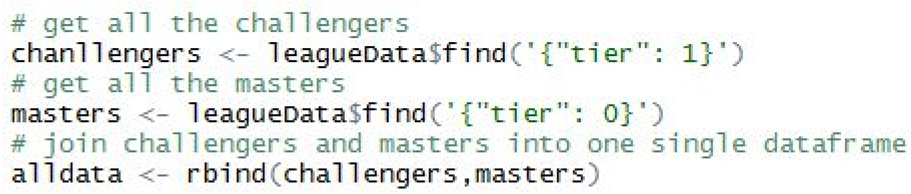
Data Storage and Retrieval

This step was relatively simple and straightforward. For nonrelational database, I downloaded MongoDB from its website following the tutorial. After running MongoDB, I installed mongolite package in R to establish connection in R.

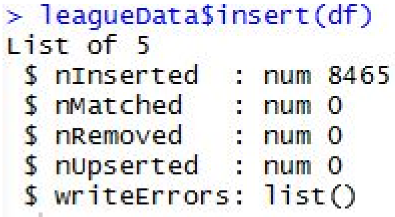


```r
# Store vaules using MongoDB
# establish connection using mongo command
leagueData <- mongo(collection = "df", db = "league")
# insert values into the database
leagueData$insert(df)
```

Using the mogo() function and insert(), I was able to establish connection and insert values successfully. Then to retrieve data, I retrieved all the players within master and challenger separately into two dataframes. I then combined them into a single data frame.



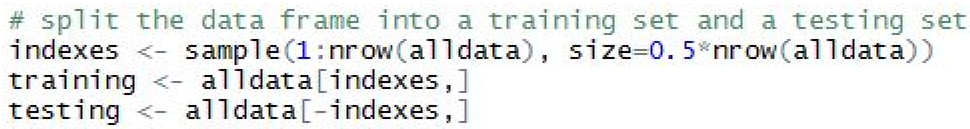
```
# get all the challengers
chanllengers <- leagueData$find('{"tier": 1}')
# get all the masters
masters <- leagueData$find('{"tier": 0}')
# join challengers and masters into one single dataframe
alldata <- rbind(challengers,masters)
```



```
> leagueData$insert(df)
List of 5
 $ nInserted  : num 8465
 $ nMatched   : num 0
 $ nRemoved   : num 0
 $ nUpserted  : num 0
 $ writeErrors: list()
```

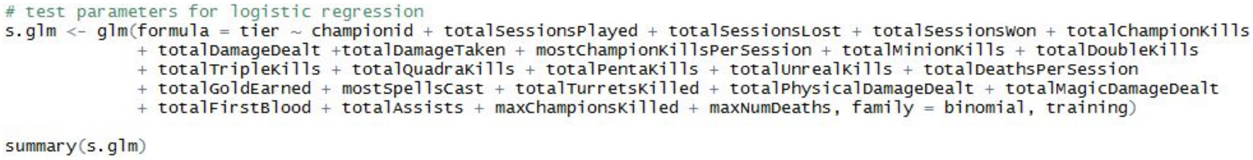
Predictive Model Construction

For constructing my logistic model, I first split the dataframe into two randomly for training and testing. After different attempts, I found simply doing it by index was the easiest.



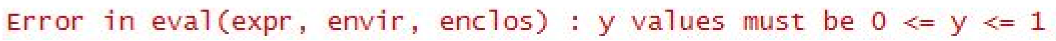
```
# split the data frame into a training set and a testing set
indexes <- sample(1:nrow(alldata), size=0.5*nrow(alldata))
training <- alldata[indexes,]
testing <- alldata[-indexes,]
```

I constructed a model with glm and started by including all the variables and started to test parameters for logistic regression.



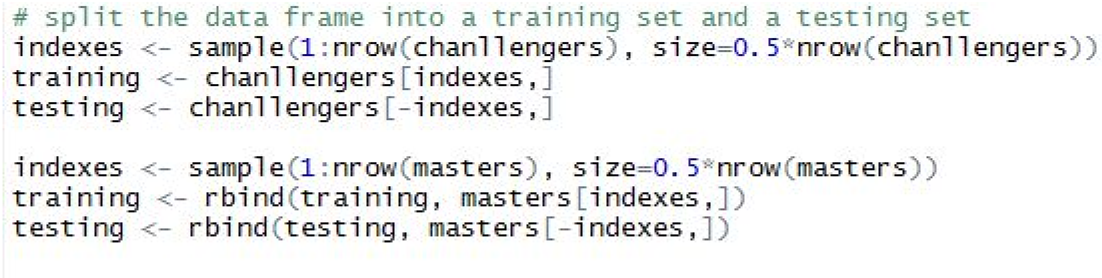
```
# test parameters for logistic regression
s.glm <- glm(formula = tier ~ championid + totalSessionsPlayed + totalSessionsLost + totalSessionsWon + totalChampionKills
             + totalDamageDealt +totalDamageTaken + mostChampionKillsPerSession + totalMinionKills + totalDoubleKills
             + totalTripleKills + totalQuadraKills + totalPentaKills + totalUnrealKills + totalDeathsPerSession
             + totalGoldEarned + mostSpellsCast + totalTurretsKilled + totalPhysicalDamageDealt + totalMagicDamageDealt
             + totalFirstBlood + totalAssists + maxChampionsKilled + maxNumDeaths, family = binomial, training)

summary(s.glm)
```

However, I got the following error.



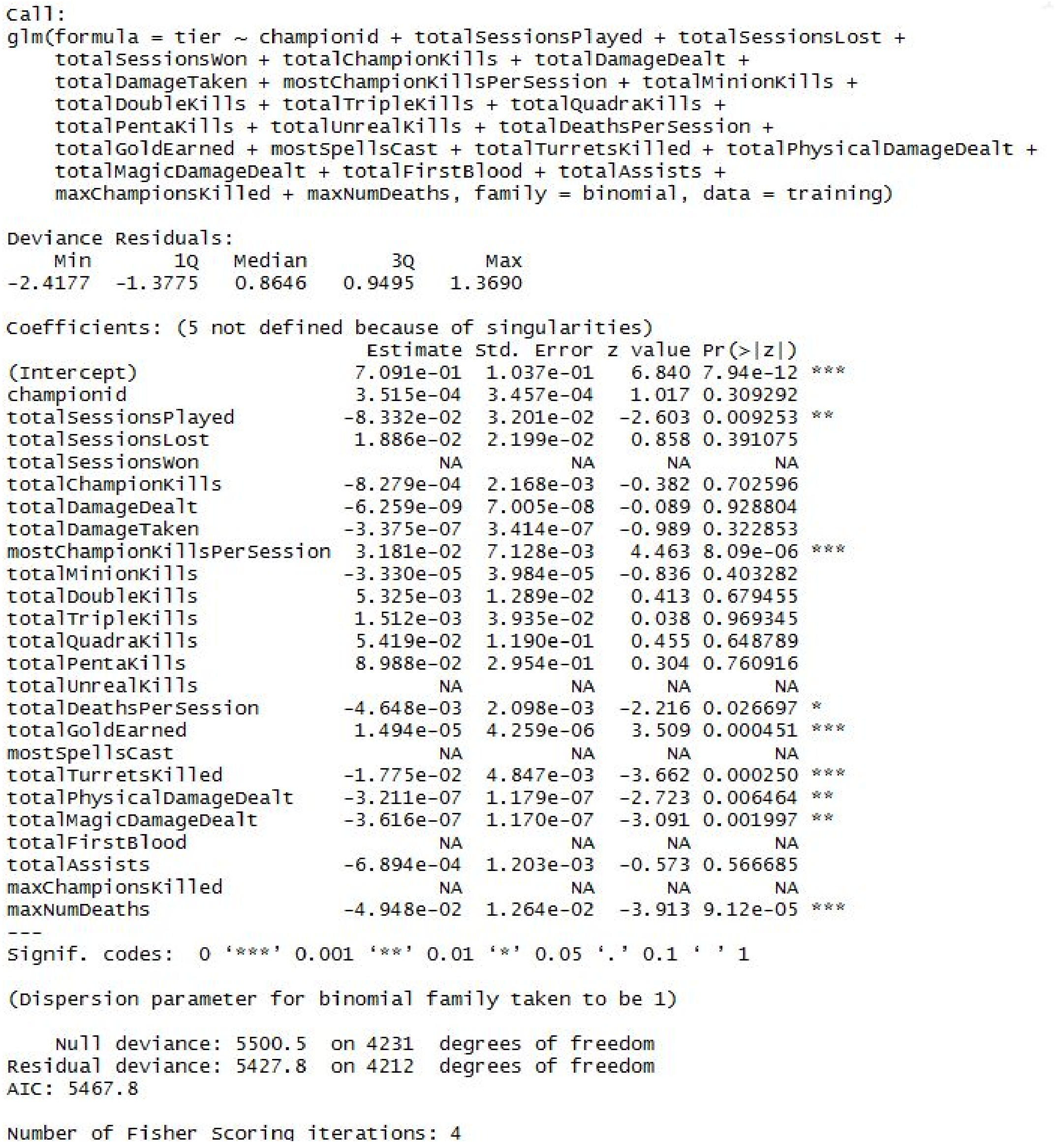
```
Error in eval(expr, envir, enclos) : y values must be 0 <= y <= 1
```

After searching online, I could not find a very good answer for my case so I started to fiddle around with the training data set. I tried putting the whole data set in or only half of them, and I suspected I am getting this error because of the split of the training data set does not have enough variance in the tier due to the way the data is arranged in the data frame. All the tier 1 is the first half and all the tier 0 is the second half, so the sample() could not get enough of 1 or 0. So instead I split my data separately and it finally worked.



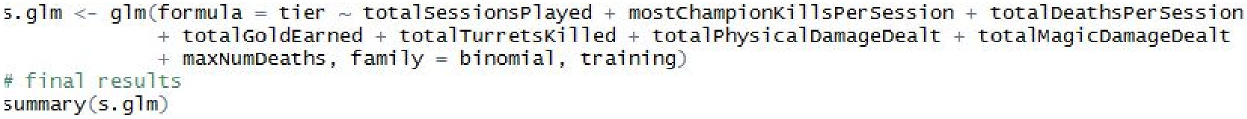
```
# split the data frame into a training set and a testing set
indexes <- sample(1:nrow(chanllengers), size=0.5*nrow(chanllengers))
training <- chanllengers[indexes,]
testing <- chanllengers[-indexes,]

indexes <- sample(1:nrow(masters), size=0.5*nrow(masters))
training <- rbind(training, masters[indexes,])
testing <- rbind(testing, masters[-indexes,])
```

After getting the summary of s.glm, something interesting comes up as shown below.



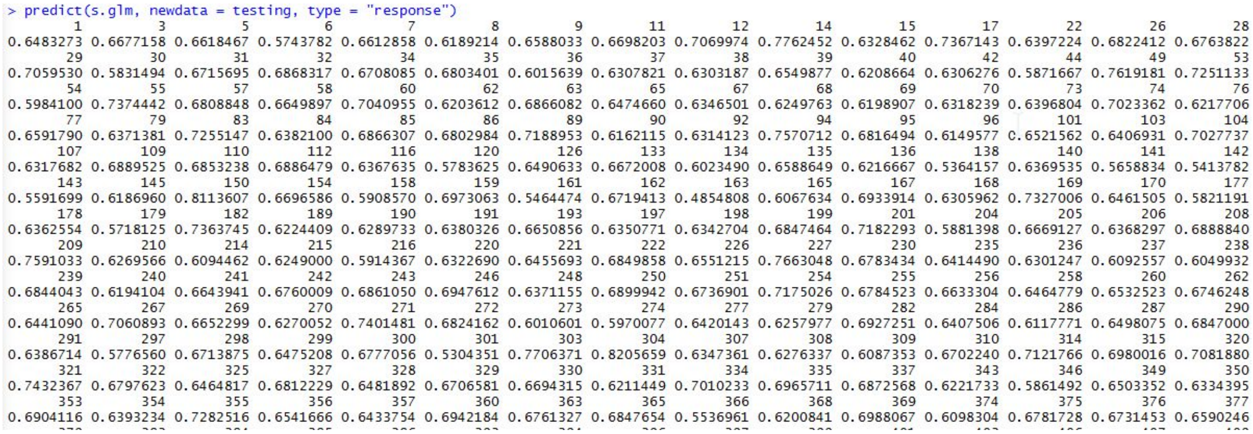
```
Call:
glm(formula = tier ~ championid + totalSessionsPlayed + totalSessionsLost + 
    totalSessionsWon + totalChampionKills + totalDamageDealt + 
    totalDamageTaken + mostChampionKillsPerSession + totalMinionKills + 
    totalDoubleKills + totalTripleKills + totalQuadraKills + 
    totalPentaKills + totalUnrealKills + totalDeathsPerSession + 
    totalGoldEarned + mostSpellsCast + totalTurretsKilled + totalPhysicalDamageDealt + 
    totalMagicDamageDealt + totalFirstBlood + totalAssists + 
    maxChampionsKilled + maxNumDeaths, family = binomial, data = training)

Deviance Residuals: 
    Min       1Q   Median       3Q      Max  
-2.4177  -1.3775   0.8646   0.9495   1.3690  

Coefficients: (5 not defined because of singularities)
                              Estimate Std. Error z value Pr(>|z|)    
(Intercept)                  7.091e-01  1.037e-01   6.840 7.94e-12 ***
championid                   3.515e-04  3.457e-04   1.017 0.309292    
totalSessionsPlayed         -8.332e-02  3.201e-02  -2.603 0.009253 ** 
totalSessionsLost            1.886e-02  2.199e-02   0.858 0.391075    
totalSessionsWon                    NA         NA      NA       NA    
totalChampionKills          -8.279e-04  2.168e-03  -0.382 0.702596    
totalDamageDealt            -6.259e-09  7.005e-08  -0.089 0.928804    
totalDamageTaken            -3.375e-07  3.414e-07  -0.989 0.322853    
mostChampionKillsPerSession  3.181e-02  7.128e-03   4.463 8.09e-06 ***
totalMinionKills            -3.330e-05  3.984e-05  -0.836 0.403282    
totalDoubleKills             5.325e-03  1.289e-02   0.413 0.679455    
totalTripleKills             1.512e-03  3.935e-02   0.038 0.969345    
totalQuadraKills             5.419e-02  1.190e-01   0.455 0.648789    
totalPentaKills              8.988e-02  2.954e-01   0.304 0.760916    
totalUnrealKills                    NA         NA      NA       NA    
totalDeathsPerSession       -4.648e-03  2.098e-03  -2.216 0.026697 *  
totalGoldEarned              1.494e-05  4.259e-06   3.509 0.000451 ***
mostSpellsCast                      NA         NA      NA       NA    
totalTurretsKilled          -1.775e-02  4.847e-03  -3.662 0.000250 ***
totalPhysicalDamageDealt    -3.211e-07  1.179e-07  -2.723 0.006464 ** 
totalMagicDamageDealt       -3.616e-07  1.170e-07  -3.091 0.001997 ** 
totalFirstBlood                     NA         NA      NA       NA    
totalAssists                -6.894e-04  1.203e-03  -0.573 0.566685    
maxChampionsKilled                  NA         NA      NA       NA    
maxNumDeaths                -4.948e-02  1.264e-02  -3.913 9.12e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 5500.5  on 4231  degrees of freedom
Residual deviance: 5427.8  on 4212  degrees of freedom
AIC: 5467.8

Number of Fisher Scoring iterations: 4
```

It is telling me that five of coefficients are not defined because of singularities. I did some searching online, and people suggested that there is multicollinearity in the data. I attempted to find ways to solve this problem but it does not seem so simple. Normally fo linear regression, you can omit the coefficients one by one to see which one to keep, however, with logistic regression, it is more complicated. I tried removing them one by one and nothing seemed to change. After consulting professor Schedlbauer, it seems like other models such as Bayesian reference or decision tree model is more suitable for this. However, those models are beyond my ability and the scope of this course, so I will instead continue this logistic regression model and omit the five undefined coefficients. Then I started to omit the coefficients with the biggest P value amongst all the ones that are higher than 0.05. 11 coefficients omitted were omitted one by one and the results are shown below.



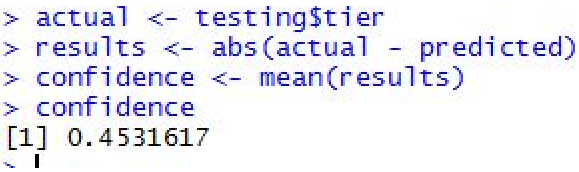
```
s.glm <- glm(formula = tier ~ totalSessionsPlayed + mostChampionKillsPerSession + totalDeathsPerSession
             + totalGoldEarned + totalTurretsKilled + totalPhysicalDamageDealt + totalMagicDamageDealt
             + maxNumDeaths, family = binomial, training)
# final results
summary(s.glm)
```

I then ran test my model with the testing data set.



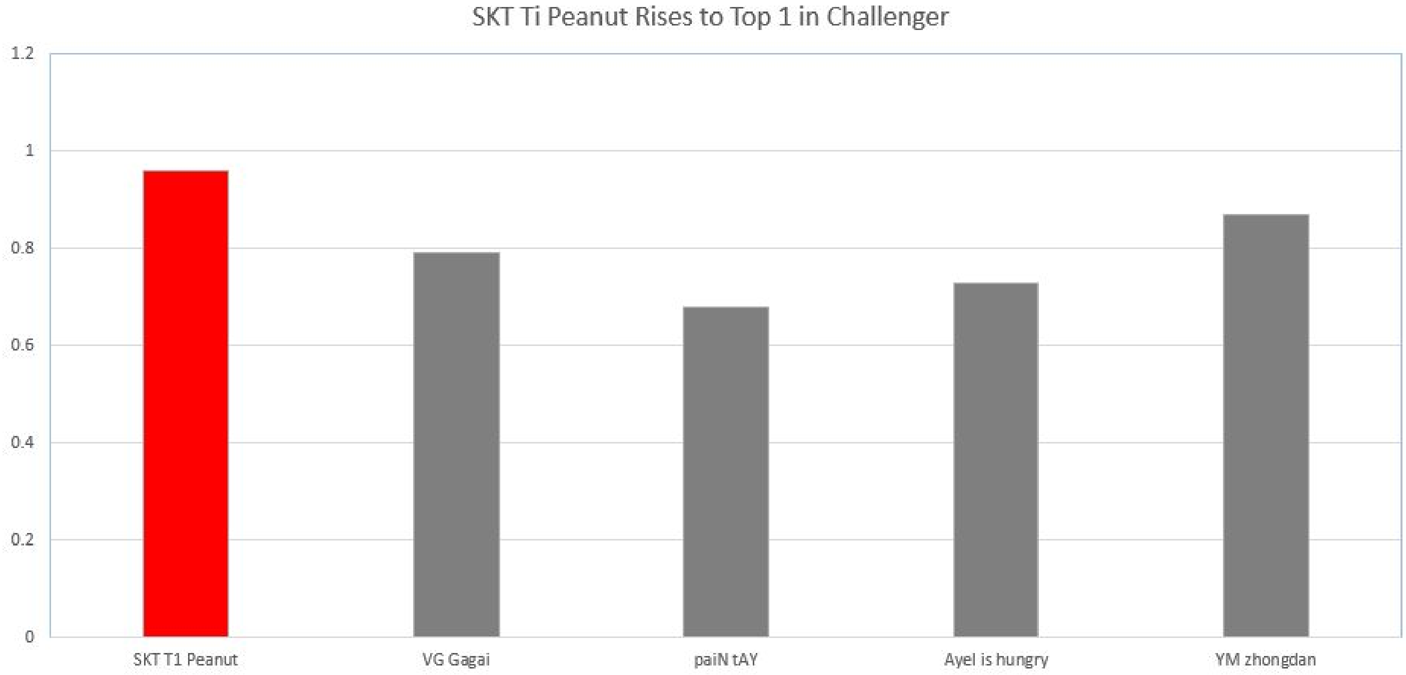
```
> predict(s.glm, newdata = testing, type = "response")
        1         3         5         6         7         8         9        11        12        14        15        17        22        26        28
0.6483273 0.6677158 0.6618467 0.5743782 0.6612858 0.6189214 0.6588033 0.6698203 0.7069974 0.7762452 0.6328462 0.7367143 0.6397224 0.6822412 0.6763822
       29        30        31        32        34        35        36        37        38        39        40        42        44        49        53
0.7059530 0.5831494 0.6715695 0.6868317 0.6708085 0.6803401 0.6015639 0.6307821 0.6303187 0.6549877 0.6208664 0.6306276 0.5871667 0.7619181 0.7251133
       54        55        57        58        60        62        63        65        67        68        69        70        73        74        76
0.5984100 0.7374442 0.6808848 0.6649897 0.7040955 0.6203612 0.6866082 0.6474660 0.6346501 0.6249763 0.6198907 0.6318239 0.6396804 0.7023362 0.6217706
       77        79        83        84        85        86        89        90        92        94        95        96       101       103       104
0.6591790 0.6371381 0.7255147 0.6382100 0.6866307 0.6802984 0.7188953 0.6162115 0.6314123 0.7570712 0.6816494 0.6149577 0.6521562 0.6406931 0.7027737
      107       109       110       112       116       120       126       133       134       135       136       138       140       141       142
0.6317682 0.6889525 0.6853238 0.6886479 0.6367635 0.5783625 0.6490633 0.6672008 0.6023490 0.6588649 0.6216667 0.5364157 0.6369535 0.5658834 0.5413782
      143       145       150       154       158       159       161       162       163       165       167       168       169       170       177
0.5591699 0.6186960 0.8113607 0.6696586 0.5908570 0.6973063 0.5464474 0.6719413 0.4854808 0.6067634 0.6933914 0.6305962 0.7327006 0.6461505 0.5821191
      178       179       182       189       190       191       193       197       198       199       201       204       205       206       208
0.6362554 0.5718125 0.7363745 0.6224409 0.6289733 0.6380326 0.6650856 0.6350771 0.6342704 0.6847464 0.7182293 0.5881398 0.6669127 0.6368297 0.6888840
      209       210       214       215       216       220       221       222       226       227       230       235       236       237       238
0.7591033 0.6269566 0.6094462 0.6249000 0.5914367 0.6322690 0.6455693 0.6849858 0.6551215 0.7663048 0.6783434 0.6414490 0.6301247 0.6092557 0.6049932
      239       240       241       242       243       246       248       250       251       254       255       256       258       260       262
0.6844043 0.6194104 0.6643941 0.6760009 0.6861050 0.6947612 0.6371155 0.6899942 0.6736901 0.7175026 0.6784523 0.6633304 0.6464779 0.6532523 0.6746248
      265       267       269       270       271       272       273       274       277       279       282       284       286       287       290
0.6441090 0.7060893 0.6652299 0.6270052 0.7401481 0.6824162 0.6010601 0.5970077 0.6420143 0.6257977 0.6927251 0.6407506 0.6117771 0.6498075 0.6847000
      291       297       298       299       300       301       303       304       307       308       309       310       314       315       320
0.6386714 0.5776560 0.6713875 0.6475208 0.6777056 0.5304351 0.7706371 0.8205659 0.6347361 0.6276337 0.6087353 0.6702240 0.7121766 0.6980016 0.7081880
      321       322       325       327       328       329       330       331       334       335       337       343       346       349       350
0.7432367 0.6797623 0.6464817 0.6812229 0.6481892 0.6706581 0.6694315 0.6211449 0.7010233 0.6965711 0.6872568 0.6221733 0.5861492 0.6503352 0.6334395
      353       354       355       356       357       360       363       365       366       368       369       374       375       376       377
0.6904116 0.6393234 0.7282516 0.6541666 0.6433754 0.6942184 0.6761327 0.6847654 0.5536961 0.6200841 0.6988067 0.6098304 0.6781728 0.6731453 0.6590246
```

This model can predict if a player is challenger or platinum value if the response value assuming the response value is rounded up if it is higher than 0.5. I also calculated the confidence level of this model.



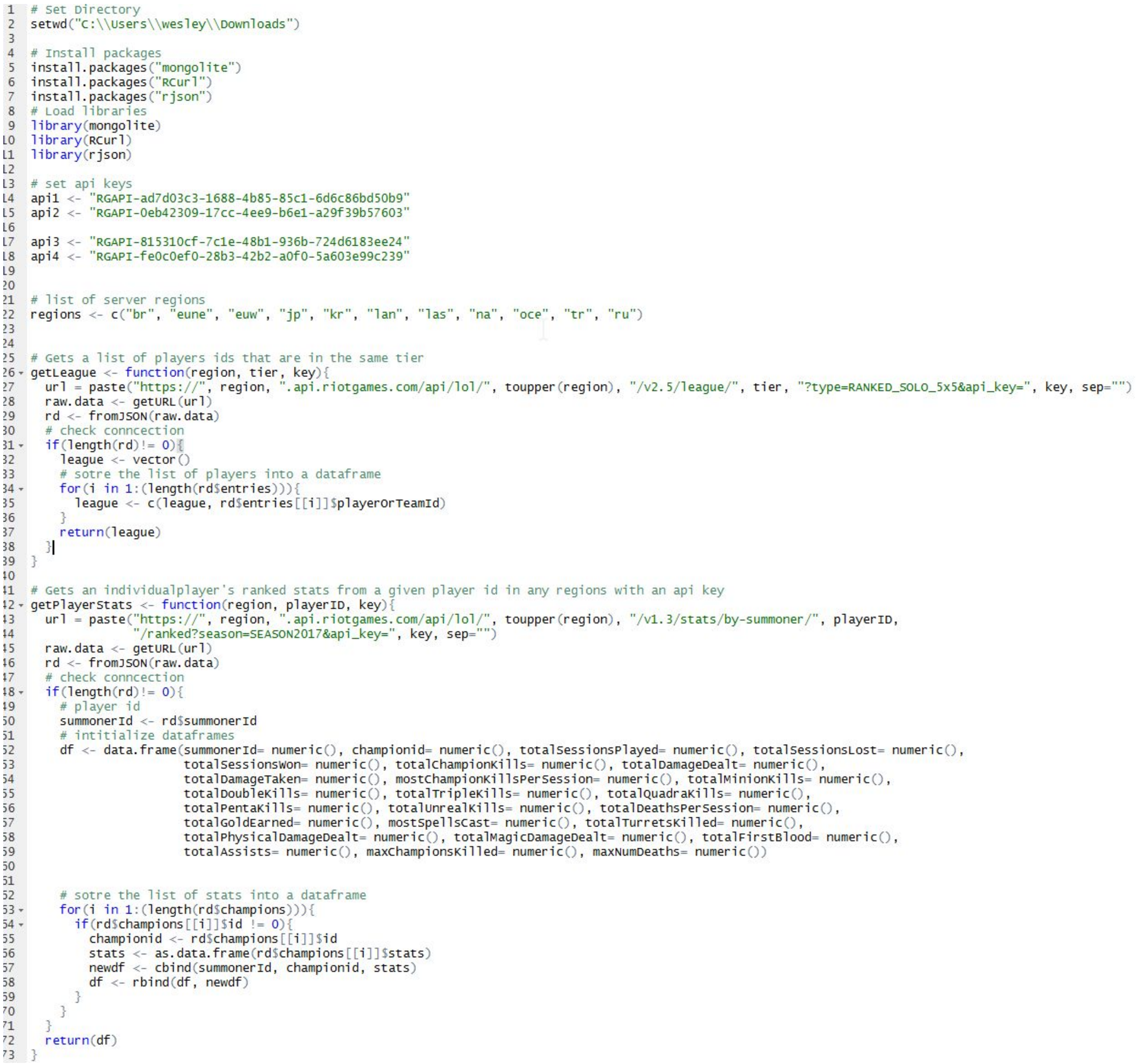
```
> actual <- testing$tier
> results <- abs(actual - predicted)
> confidence <- mean(results)
> confidence
[1] 0.4531617
>
```

Explanatory Visualization

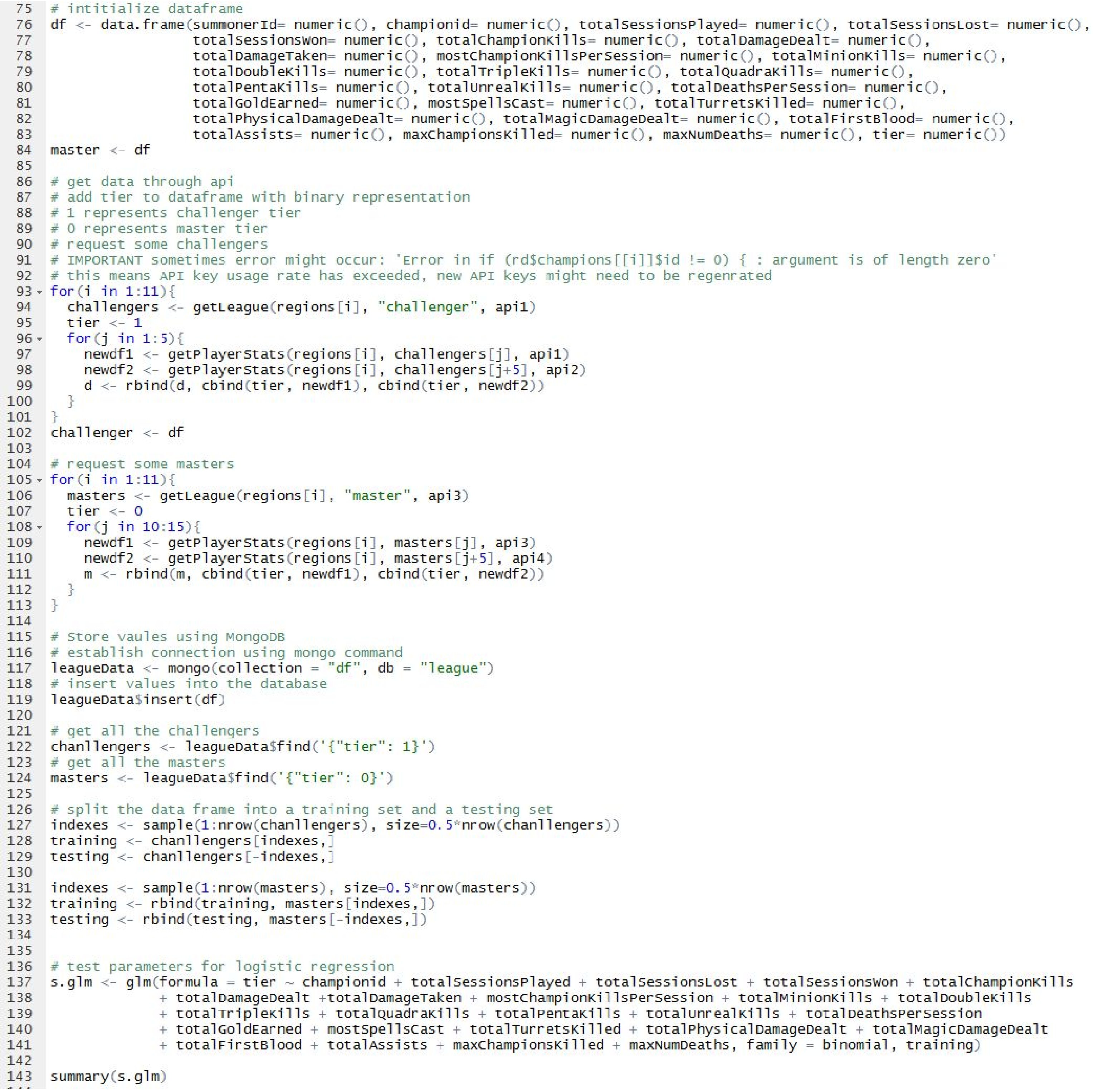


For this visualization, I selected the top 5 players of all the regions and ran the logistic model on them. (ranked solely by their elo scores, instead of other stats) Since they are the top players amongst the millions, the model should naturally return values close to 1. Interestingly, the number one ranked player by ELO also has the predicted value from the model closest to 1.

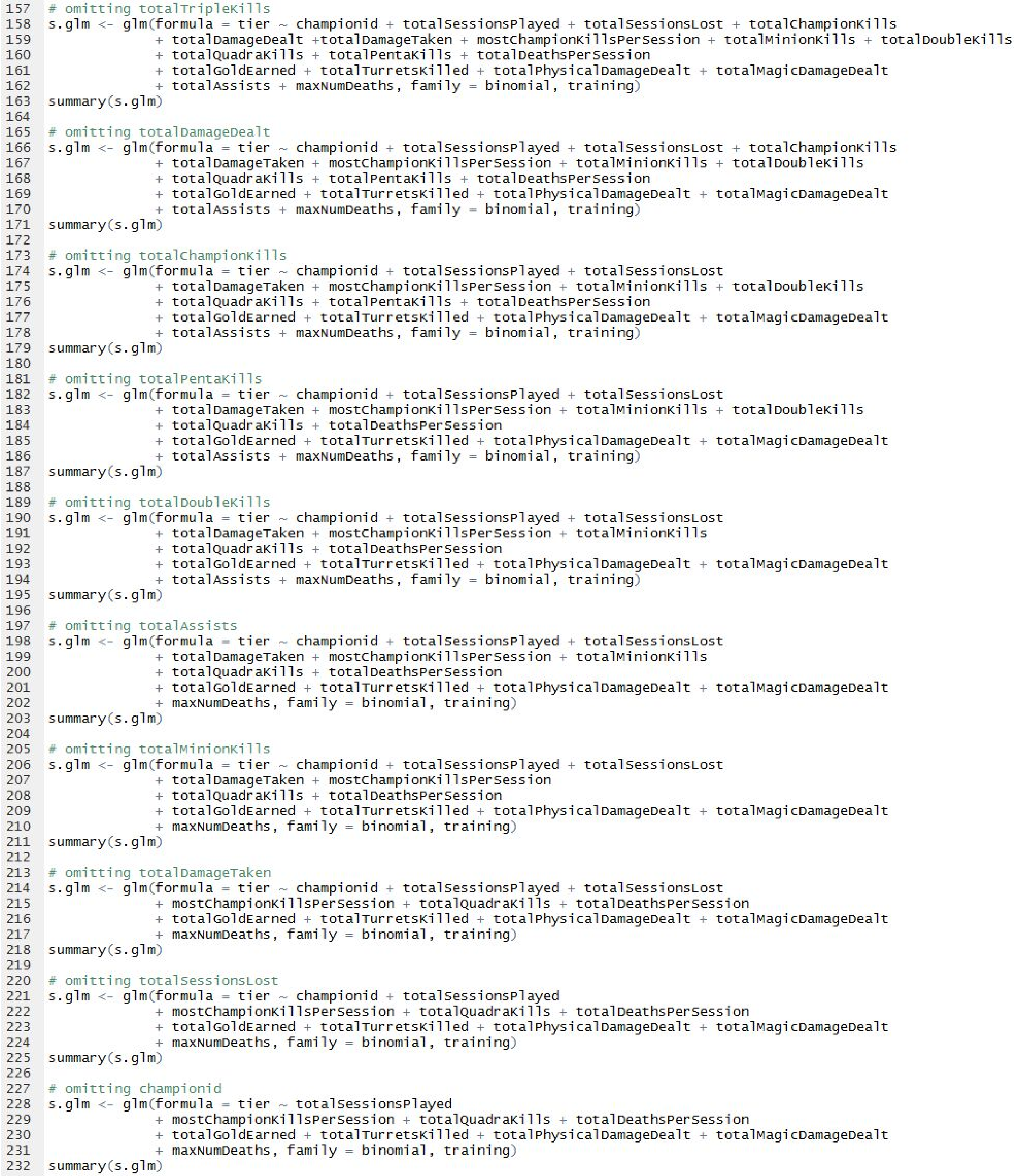
Walk Through of the final code:



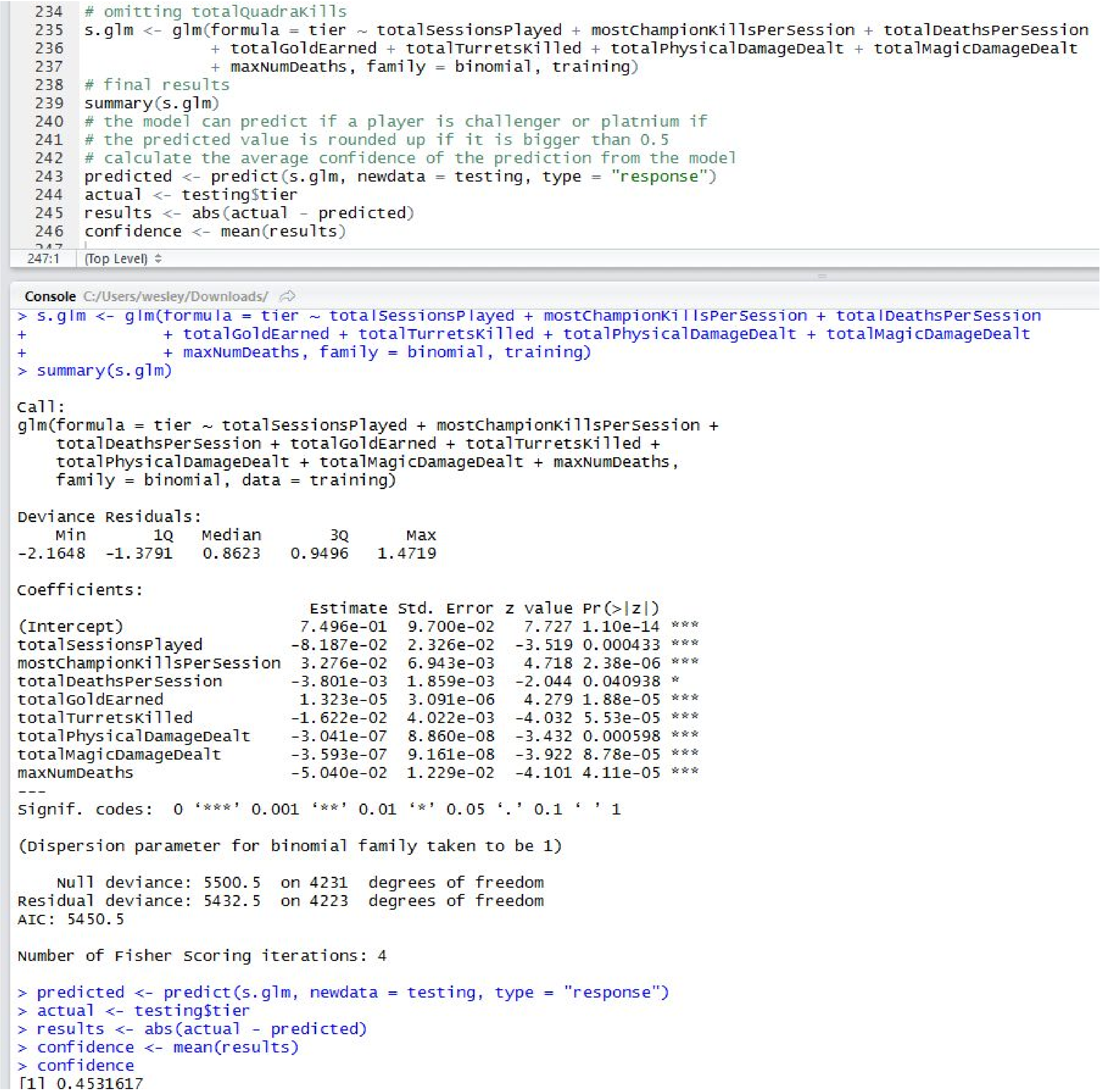
```r
# Set Directory
setwd("C:\\Users\\wesley\\Downloads")

# Install packages
install.packages("mongolite")
install.packages("RCurl")
install.packages("rjson")
# Load libraries
library(mongolite)
library(RCurl)
library(rjson)

# set api keys
api1 <- "RGAPI-ad7d03c3-1688-4b85-85c1-6d6c86bd50b9"
api2 <- "RGAPI-0eb42309-17cc-4ee9-b6e1-a29f39b57603"

api3 <- "RGAPI-815310cf-7c1e-48b1-936b-724d6183ee24"
api4 <- "RGAPI-fe0c0ef0-28b3-42b2-a0f0-5a603e99c239"


# list of server regions
regions <- c("br", "eune", "euw", "jp", "kr", "lan", "las", "na", "oce", "tr", "ru")


# Gets a list of players ids that are in the same tier
getLeague <- function(region, tier, key){
  url = paste("https://", region, ".api.riotgames.com/api/lol/", toupper(region), "/v2.5/league/", tier, "?type=RANKED_SOLO_5x5&api_key=", key, sep="")
  raw.data <- getURL(url)
  rd <- fromJSON(raw.data)
  # check conncection
  if(length(rd)!= 0){
    league <- vector()
    # sotre the list of players into a dataframe
    for(i in 1:(length(rd$entries))){
      league <- c(league, rd$entries[[i]]$playerOrTeamId)
    }
    return(league)
  }
}

# Gets an individualplayer's ranked stats from a given player id in any regions with an api key
getPlayerStats <- function(region, playerID, key){
  url = paste("https://", region, ".api.riotgames.com/api/lol/", toupper(region), "/v1.3/stats/by-summoner/", playerID,
              "/ranked?season=SEASON2017&api_key=", key, sep="")
  raw.data <- getURL(url)
  rd <- fromJSON(raw.data)
  # check conncection
  if(length(rd)!= 0){
    # player id
    summonerId <- rd$summonerId
    # intitialize dataframes
    df <- data.frame(summonerId= numeric(), championid= numeric(), totalSessionsPlayed= numeric(), totalSessionsLost= numeric(),
                     totalSessionsWon= numeric(), totalChampionKills= numeric(), totalDamageDealt= numeric(),
                     totalDamageTaken= numeric(), mostChampionKillsPerSession= numeric(), totalMinionKills= numeric(),
                     totalDoubleKills= numeric(), totalTripleKills= numeric(), totalQuadraKills= numeric(),
                     totalPentaKills= numeric(), totalUnrealKills= numeric(), totalDeathsPerSession= numeric(),
                     totalGoldEarned= numeric(), mostSpellsCast= numeric(), totalTurretsKilled= numeric(),
                     totalPhysicalDamageDealt= numeric(), totalMagicDamageDealt= numeric(), totalFirstBlood= numeric(),
                     totalAssists= numeric(), maxChampionsKilled= numeric(), maxNumDeaths= numeric())


    # sotre the list of stats into a dataframe
    for(i in 1:(length(rd$champions))){
      if(rd$champions[[i]]$id != 0){
        championid <- rd$champions[[i]]$id
        stats <- as.data.frame(rd$champions[[i]]$stats)
        newdf <- cbind(summonerId, championid, stats)
        df <- rbind(df, newdf)
      }
    }
  }
  return(df)
}
```



```r
# intitialize dataframe
df <- data.frame(summonerId= numeric(), championid= numeric(), totalSessionsPlayed= numeric(), totalSessionsLost= numeric(),
                 totalSessionsWon= numeric(), totalChampionKills= numeric(), totalDamageDealt= numeric(),
                 totalDamageTaken= numeric(), mostChampionKillsPerSession= numeric(), totalMinionKills= numeric(),
                 totalDoubleKills= numeric(), totalTripleKills= numeric(), totalQuadraKills= numeric(),
                 totalPentaKills= numeric(), totalUnrealKills= numeric(), totalDeathsPerSession= numeric(),
                 totalGoldEarned= numeric(), mostSpellsCast= numeric(), totalTurretsKilled= numeric(),
                 totalPhysicalDamageDealt= numeric(), totalMagicDamageDealt= numeric(), totalFirstBlood= numeric(),
                 totalAssists= numeric(), maxChampionsKilled= numeric(), maxNumDeaths= numeric(), tier= numeric())
master <- df

# get data through api
# add tier to dataframe with binary representation
# 1 represents challenger tier
# 0 represents master tier
# request some challengers
# IMPORTANT sometimes error might occur: 'Error in if (rd$champions[[i]]$id != 0) { : argument is of length zero'
# this means API key usage rate has exceeded, new API keys might need to be regenrated
for(i in 1:11){
  challengers <- getLeague(regions[i], "challenger", api1)
  tier <- 1
  for(j in 1:5){
    newdf1 <- getPlayerStats(regions[i], challengers[j], api1)
    newdf2 <- getPlayerStats(regions[i], challengers[j+5], api2)
    d <- rbind(d, cbind(tier, newdf1), cbind(tier, newdf2))
  }
}
challenger <- df

# request some masters
for(i in 1:11){
  masters <- getLeague(regions[i], "master", api3)
  tier <- 0
  for(j in 10:15){
    newdf1 <- getPlayerStats(regions[i], masters[j], api3)
    newdf2 <- getPlayerStats(regions[i], masters[j+5], api4)
    m <- rbind(m, cbind(tier, newdf1), cbind(tier, newdf2))
  }
}

# Store vaules using MongoDB
# establish connection using mongo command
leagueData <- mongo(collection = "df", db = "league")
# insert values into the database
leagueData$insert(df)

# get all the challengers
chanllengers <- leagueData$find('{"tier": 1}')
# get all the masters
masters <- leagueData$find('{"tier": 0}')

# split the data frame into a training set and a testing set
indexes <- sample(1:nrow(chanllengers), size=0.5*nrow(chanllengers))
training <- chanllengers[indexes,]
testing <- chanllengers[-indexes,]

indexes <- sample(1:nrow(masters), size=0.5*nrow(masters))
training <- rbind(training, masters[indexes,])
testing <- rbind(testing, masters[-indexes,])


# test parameters for logistic regression
s.glm <- glm(formula = tier ~ championid + totalSessionsPlayed + totalSessionsLost + totalSessionsWon + totalChampionKills
             + totalDamageDealt +totalDamageTaken + mostChampionKillsPerSession + totalMinionKills + totalDoubleKills
             + totalTripleKills + totalQuadraKills + totalPentaKills + totalUnrealKills + totalDeathsPerSession
             + totalGoldEarned + mostSpellsCast + totalTurretsKilled + totalPhysicalDamageDealt + totalMagicDamageDealt
             + totalFirstBlood + totalAssists + maxChampionsKilled + maxNumDeaths, family = binomial, training)

summary(s.glm)
```



```
# omitting totalTripleKills
s.glm <- glm(formula = tier ~ championid + totalSessionsPlayed + totalSessionsLost + totalChampionKills
             + totalDamageDealt +totalDamageTaken + mostChampionKillsPerSession + totalMinionKills + totalDoubleKills
             + totalQuadraKills + totalPentaKills + totalDeathsPerSession
             + totalGoldEarned + totalTurretsKilled + totalPhysicalDamageDealt + totalMagicDamageDealt
             + totalAssists + maxNumDeaths, family = binomial, training)
summary(s.glm)

# omitting totalDamageDealt
s.glm <- glm(formula = tier ~ championid + totalSessionsPlayed + totalSessionsLost + totalChampionKills
             + totalDamageTaken + mostChampionKillsPerSession + totalMinionKills + totalDoubleKills
             + totalQuadraKills + totalPentaKills + totalDeathsPerSession
             + totalGoldEarned + totalTurretsKilled + totalPhysicalDamageDealt + totalMagicDamageDealt
             + totalAssists + maxNumDeaths, family = binomial, training)
summary(s.glm)

# omitting totalChampionKills
s.glm <- glm(formula = tier ~ championid + totalSessionsPlayed + totalSessionsLost
             + totalDamageTaken + mostChampionKillsPerSession + totalMinionKills + totalDoubleKills
             + totalQuadraKills + totalPentaKills + totalDeathsPerSession
             + totalGoldEarned + totalTurretsKilled + totalPhysicalDamageDealt + totalMagicDamageDealt
             + totalAssists + maxNumDeaths, family = binomial, training)
summary(s.glm)

# omitting totalPentaKills
s.glm <- glm(formula = tier ~ championid + totalSessionsPlayed + totalSessionsLost
             + totalDamageTaken + mostChampionKillsPerSession + totalMinionKills + totalDoubleKills
             + totalQuadraKills + totalDeathsPerSession
             + totalGoldEarned + totalTurretsKilled + totalPhysicalDamageDealt + totalMagicDamageDealt
             + totalAssists + maxNumDeaths, family = binomial, training)
summary(s.glm)

# omitting totalDoubleKills
s.glm <- glm(formula = tier ~ championid + totalSessionsPlayed + totalSessionsLost
             + totalDamageTaken + mostChampionKillsPerSession + totalMinionKills
             + totalQuadraKills + totalDeathsPerSession
             + totalGoldEarned + totalTurretsKilled + totalPhysicalDamageDealt + totalMagicDamageDealt
             + totalAssists + maxNumDeaths, family = binomial, training)
summary(s.glm)

# omitting totalAssists
s.glm <- glm(formula = tier ~ championid + totalSessionsPlayed + totalSessionsLost
             + totalDamageTaken + mostChampionKillsPerSession + totalMinionKills
             + totalQuadraKills + totalDeathsPerSession
             + totalGoldEarned + totalTurretsKilled + totalPhysicalDamageDealt + totalMagicDamageDealt
             + maxNumDeaths, family = binomial, training)
summary(s.glm)

# omitting totalMinionKills
s.glm <- glm(formula = tier ~ championid + totalSessionsPlayed + totalSessionsLost
             + totalDamageTaken + mostChampionKillsPerSession
             + totalQuadraKills + totalDeathsPerSession
             + totalGoldEarned + totalTurretsKilled + totalPhysicalDamageDealt + totalMagicDamageDealt
             + maxNumDeaths, family = binomial, training)
summary(s.glm)

# omitting totalDamageTaken
s.glm <- glm(formula = tier ~ championid + totalSessionsPlayed + totalSessionsLost
             + mostChampionKillsPerSession + totalQuadraKills + totalDeathsPerSession
             + totalGoldEarned + totalTurretsKilled + totalPhysicalDamageDealt + totalMagicDamageDealt
             + maxNumDeaths, family = binomial, training)
summary(s.glm)

# omitting totalSessionsLost
s.glm <- glm(formula = tier ~ championid + totalSessionsPlayed
             + mostChampionKillsPerSession + totalQuadraKills + totalDeathsPerSession
             + totalGoldEarned + totalTurretsKilled + totalPhysicalDamageDealt + totalMagicDamageDealt
             + maxNumDeaths, family = binomial, training)
summary(s.glm)

# omitting championid
s.glm <- glm(formula = tier ~ totalSessionsPlayed
             + mostChampionKillsPerSession + totalQuadraKills + totalDeathsPerSession
             + totalGoldEarned + totalTurretsKilled + totalPhysicalDamageDealt + totalMagicDamageDealt
             + maxNumDeaths, family = binomial, training)
summary(s.glm)
```



```
# omitting totalQuadraKills
s.glm <- glm(formula = tier ~ totalSessionsPlayed + mostChampionKillsPerSession + totalDeathsPerSession
             + totalGoldEarned + totalTurretsKilled + totalPhysicalDamageDealt + totalMagicDamageDealt
             + maxNumDeaths, family = binomial, training)
# final results
summary(s.glm)
# the model can predict if a player is challenger or platnium if
# the predicted value is rounded up if it is bigger than 0.5
# calculate the average confidence of the prediction from the model
predicted <- predict(s.glm, newdata = testing, type = "response")
actual <- testing$tier
results <- abs(actual - predicted)
confidence <- mean(results)
```

```
> s.glm <- glm(formula = tier ~ totalSessionsPlayed + mostChampionKillsPerSession + totalDeathsPerSession
+              + totalGoldEarned + totalTurretsKilled + totalPhysicalDamageDealt + totalMagicDamageDealt
+              + maxNumDeaths, family = binomial, training)
> summary(s.glm)

Call:
glm(formula = tier ~ totalSessionsPlayed + mostChampionKillsPerSession + 
    totalDeathsPerSession + totalGoldEarned + totalTurretsKilled + 
    totalPhysicalDamageDealt + totalMagicDamageDealt + maxNumDeaths, 
    family = binomial, data = training)

Deviance Residuals: 
    Min       1Q   Median       3Q      Max  
-2.1648  -1.3791   0.8623   0.9496   1.4719  

Coefficients:
                              Estimate Std. Error z value Pr(>|z|)    
(Intercept)                  7.496e-01  9.700e-02   7.727 1.10e-14 ***
totalSessionsPlayed         -8.187e-02  2.326e-02  -3.519 0.000433 ***
mostChampionKillsPerSession  3.276e-02  6.943e-03   4.718 2.38e-06 ***
totalDeathsPerSession       -3.801e-03  1.859e-03  -2.044 0.040938 *  
totalGoldEarned              1.323e-05  3.091e-06   4.279 1.88e-05 ***
totalTurretsKilled          -1.622e-02  4.022e-03  -4.032 5.53e-05 ***
totalPhysicalDamageDealt    -3.041e-07  8.860e-08  -3.432 0.000598 ***
totalMagicDamageDealt       -3.593e-07  9.161e-08  -3.922 8.78e-05 ***
maxNumDeaths                -5.040e-02  1.229e-02  -4.101 4.11e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 5500.5  on 4231  degrees of freedom
Residual deviance: 5432.5  on 4223  degrees of freedom
AIC: 5450.5

Number of Fisher Scoring iterations: 4

> predicted <- predict(s.glm, newdata = testing, type = "response")
> actual <- testing$tier
> results <- abs(actual - predicted)
> confidence <- mean(results)
> confidence
[1] 0.4531617
```