Overwatch League Exploratory Analysis

Mark Davison

```r
library(tidyr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(pander)
library(car)
```

```
## Loading required package: carData

##
## Attaching package: 'car'

## The following object is masked from 'package:dplyr':
##
##     recode
```

```r
library(cowplot)
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```
library(ggplot2)
```

The purpose of this project is exploratory analysis of the Overwatch League. My aim is to find which variables are significant predictors of winning matches. This study with inform the variables I use building the final model of predicting the outcome of Overwatch League matches in another project.

Necessary Domain Knowledge:

The Overwatch league is an international Esports league ran and owned by Blizzard. The league is comprised of 19 city-based teams. The matches are formatted in best of 5 games. With each game being played on a unique map. The maps range from 1 to 4 rounds. In this project, I am only interested in the winners of maps and matches, not rounds.

Each team consists of 5 players, each having unique roles. A tank, two damage dealing players, and two healing/support players.

Data

I will be using two sets of data. "phs-2023/2022" and "watch_map_stats". The "phs" dataset are player statitistics, while "watch_map_stats" are the map statistics. The payer statistics include the predictor variables of the players performance, while the map statistics include the response variable (what I aim on predicting) the outcome of each match. The data is all sourced from the Overwatch League statistics page.

(Note, I am only using 2022 and 2023, as overwatch 2 was released in 2022, which came with significant balance changes, most notably changing the number of players in each team from 6 to 5, rendering all data pre 2022 useless here).

##Abstract The exploratory analysis found that variables 'Deaths,' 'Defensive_Assists,' 'Eliminations,' 'Hero_Damage_Done,' 'Objective_Time,' 'Recon_Assists,' 'Time_Alive', were the most significant of the predictors in the dataset at predicting the outcome of matches. This will be used to inform the building of another model that uses these variables in the context of a team vs team model, rather than player vs player.

I ran into some issues with multi-collinearity, as many variables record very similar things, primarily time based. After removing said variables, the predictive power of the model decreased very marginally, but the complexity was reduced significantly. My future study of this subject will be focused on the effect of these variables when modelled as a team.

```
#loading player data and merging years
owlplayer2023 <- read.csv("phs-2023.csv")
owlplayer2022 <- read.csv("phs-2022.csv")
owlplayermerge <- rbind(owlplayer2023, owlplayer2022)
```

```r
#Loading map data
owlmap <- read.csv("match_map_stats_000000000000.csv")
```

```r
owlplayermerge[290:300,]
```

```
##                  start_time esports_match_id tournament_title map_type map_name
## 290 2023-03-23 20:11:00 UTC            41215           Pro-Am  control    Nepal
## 291 2023-03-23 20:11:00 UTC            41215           Pro-Am  control    Nepal
## 292 2023-03-23 20:11:00 UTC            41215           Pro-Am  control    Nepal
## 293 2023-03-23 20:11:00 UTC            41215           Pro-Am  control    Nepal
## 294 2023-03-23 20:11:00 UTC            41215           Pro-Am  control    Nepal
## 295 2023-03-23 20:11:00 UTC            41215           Pro-Am  control    Nepal
## 296 2023-03-23 20:11:00 UTC            41215           Pro-Am  control    Nepal
## 297 2023-03-23 20:11:00 UTC            41215           Pro-Am  control    Nepal
## 298 2023-03-23 20:11:00 UTC            41215           Pro-Am  control    Nepal
## 299 2023-03-23 20:11:00 UTC            41215           Pro-Am  control    Nepal
## 300 2023-03-23 20:11:00 UTC            41215           Pro-Am  control    Nepal
##     player_name      team_name             stat_name  hero_name       amount
## 290       MER1T Florida Mayhem       All Damage Done All Heroes 14806.654358
## 291       MER1T Florida Mayhem               Assists All Heroes    10.000000
## 292       MER1T Florida Mayhem     Average Time Alive All Heroes    63.224814
## 293       MER1T Florida Mayhem   Barrier Damage Done All Heroes  1670.999992
## 294       MER1T Florida Mayhem           Damage Done All Heroes 13022.254560
## 295       MER1T Florida Mayhem          Damage Taken All Heroes  6763.294514
## 296       MER1T Florida Mayhem                Deaths All Heroes     6.000000
## 297       MER1T Florida Mayhem          Eliminations All Heroes    17.000000
## 298       MER1T Florida Mayhem           Final Blows All Heroes     7.000000
## 299       MER1T Florida Mayhem Games Played Plus Won All Heroes     2.000001
## 300       MER1T Florida Mayhem             Games Won All Heroes     1.000001
```

Looking at the player data, there are some glaring issues that need to be solved to make this data model ready. Specifically:
- The dataframe is currently in long format, meaning the variables we are interested in stat_name are in rows, while they need to be in columns. Also note that there are statistcs recorded for each individual character played. We will not be used individual character statistics, as it introduces complexity that is not likely to be significant, and it introduces more issues such as many character not being played often etc. I am therefore going to limit the statistics to all heroes only.

```r
#filtering out characters statistics, as I am only interested in 'all heroes' statistics
owlplayer1 <- owlplayermerge[owlplayermerge$hero_name == "All Heroes",]
```

```r
#pivoting the dataframe from long to wide format
owlpivot <- pivot_wider(owlplayer1, names_from=stat_name, values_from=amount)
owlpivot[20:30]
```

```
## # A tibble: 14,640 x 11
##    Eliminations `Final Blows` `Games Played Plus Won` `Games Won` `Healing Done`
##    <list>       <list>        <list>                  <list>      <list>
##  1 <dbl [1]>    <dbl [1]>     <dbl [1]>               <dbl [2]>   <dbl [1]>
##  2 <dbl [1]>    <dbl [1]>     <dbl [1]>               <dbl [2]>   <NULL>
##  3 <dbl [1]>    <dbl [1]>     <dbl [1]>               <dbl [2]>   <NULL>
##  4 <dbl [1]>    <dbl [1]>     <dbl [1]>               <dbl [2]>   <dbl [1]>
##  5 <dbl [1]>    <dbl [1]>     <dbl [1]>                <dbl [2]>   <NULL>
##  6 <dbl [1]>    <dbl [1]>     <dbl [1]>                <NULL>      <dbl [1]>
##  7 <dbl [1]>    <dbl [1]>     <dbl [1]>                <NULL>      <dbl [1]>
##  8 <dbl [1]>    <dbl [1]>     <dbl [1]>                <NULL>      <dbl [1]>
##  9 <dbl [1]>    <dbl [1]>     <dbl [1]>                <NULL>      <NULL>
## 10 <dbl [1]>    <dbl [1]>     <dbl [1]>                <NULL>      <dbl [1]>
## # i 14,630 more rows
## # i 6 more variables: `Hero Damage Done` <list>, `Hero Wins` <list>,
## #   `Knockback Kills` <list>, `Objective Contest Time` <list>,
## #   `Objective Contest Time - Avg per 10 Min` <list>,
## #   `Objective Contest Time - Most in Game` <list>
```

This has successfully pivoted the dataframe, however many observations have recorded multiple values for each player for one map, and caused the columns to be formatted as lists.

The 'Assist' column is the only variable that is required that has these multiple recorded values. We will select the first assist recorded for the variable:

```r
owlpivot$Assists[10:20]
```

```
## [[1]]
## [1] 11 12
##
## [[2]]
## [1] 16 11
##
## [[3]]
## [1] 17
##
## [[4]]
## [1] 5
##
## [[5]]
## [1]  8 19
##
## [[6]]
## [1]  3 10
##
## [[7]]
## [1] 19  7
##
## [[8]]
## [1] 11
##
## [[9]]
## [1] 15  7
##
## [[10]]
## [1] 9 3
##
## [[11]]
## [1] 18 11
```

As shown in this output, some observations have two values, while some only have one.

```
#Loop that makes the list equal to the first value in the list.
for(i in 1:nrow(owlpivot)){
  if(length(owlpivot$Assists[[i]])>1){
   owlpivot$Assists[[i]] <- owlpivot$Assists[[i]][1]
  }
}
```

```
owlpivot$Assists[10:20]
```

```
## [[1]]
## [1] 11
##
## [[2]]
## [1] 16
##
## [[3]]
## [1] 17
##
## [[4]]
## [1] 5
##
## [[5]]
## [1] 8
##
## [[6]]
## [1] 3
##
## [[7]]
## [1] 19
##
## [[8]]
## [1] 11
##
## [[9]]
## [1] 15
##
## [[10]]
## [1] 9
##
## [[11]]
## [1] 18
```

- There are a number of redundant variables that need to be removed: ("Damage_Done" and "Hero_Damage_Done" are the exact same, damage_done will be removed. The same for "Objective_Contest_Time_Most_in_Game" and "Objective_Contest_Time")

```
owlpivot[,c("Hero Wins", "Games Played Plus Won", "Games Won", "Assists - Most in Game", "Assists  - Avg
head(owlpivot)
```

```
## # A tibble: 6 x 46
##   start_time     esports_match_id tournament_title map_type map_name player_name
##   <chr>                     <int> <chr>            <chr>    <chr>    <chr>
## 1 2023-03-23 20~           41215 Pro-Am          control  Nepal    CHORONG
## 2 2023-03-23 20~           41215 Pro-Am          control  Nepal    Checkmate
## 3 2023-03-23 20~           41215 Pro-Am          control  Nepal    MER1T
## 4 2023-03-23 20~           41215 Pro-Am          control  Nepal    Rupal
## 5 2023-03-23 20~           41215 Pro-Am          control  Nepal    Someone
## 6 2023-03-23 20~           41215 Pro-Am          control  Nepal    FiNN
## # i 40 more variables: team_name <chr>, `All Damage Done` <list>,
## #   Assists <list>, `Average Time Alive` <list>, `Barrier Damage Done` <list>,
## #   `Damage - Quick Melee` <list>, `Damage Done` <list>, `Damage Taken` <list>,
## #   Deaths <list>, `Defensive Assists` <list>, Eliminations <list>,
## #   `Final Blows` <list>, `Healing Done` <list>, `Hero Damage Done` <list>,
## #   `Knockback Kills` <list>, `Objective Contest Time` <list>,
## #   `Objective Contest Time - Avg per 10 Min` <list>, ...
```

- The nested list columns need to be un-nested, and the NULL values need to be replaced with 0s

```
owlplayer <- owlpivot %>%
  unnest(everything()) %>%
   mutate_all(~replace_na(., 0))

#Also renaming the 'esports_match_id' variable to 'match_id' for simplicity
names(owlplayer)[2] <- "match_id"
head(owlplayer)
```

```
## # A tibble: 6 x 46
##   start_time   match_id tournament_title map_type map_name player_name team_name
##   <chr>           <int> <chr>            <chr>    <chr>    <chr>       <chr>
## 1 2023-03-23 ~    41215 Pro-Am           control  Nepal    CHORONG     Florida ~
## 2 2023-03-23 ~    41215 Pro-Am           control  Nepal    Checkmate   Florida ~
## 3 2023-03-23 ~    41215 Pro-Am           control  Nepal    MER1T       Florida ~
## 4 2023-03-23 ~    41215 Pro-Am           control  Nepal    Rupal       Florida ~
## 5 2023-03-23 ~    41215 Pro-Am           control  Nepal    Someone     Florida ~
## 6 2023-03-23 ~    41215 Pro-Am           control  Nepal    FiNN        San Fran~
## # i 39 more variables: `All Damage Done` <dbl>, Assists <dbl>,
## #   `Average Time Alive` <dbl>, `Barrier Damage Done` <dbl>,
## #   `Damage - Quick Melee` <dbl>, `Damage Done` <dbl>, `Damage Taken` <dbl>,
## #   Deaths <dbl>, `Defensive Assists` <dbl>, Eliminations <dbl>,
## #   `Final Blows` <dbl>, `Healing Done` <dbl>, `Hero Damage Done` <dbl>,
## #   `Knockback Kills` <dbl>, `Objective Contest Time` <dbl>,
## #   `Objective Contest Time - Avg per 10 Min` <dbl>, ...
```

The *owlfinal* dataframe (which contains data of player statistics for each map) needs to be combined with the map dataframe (which contains data of each map played, including the winning team(our response variable)).

First, I need to clean the map dataset:
- There are variables in this dataset that I am not interested in

```
owlmap <- owlmap[,names(owlmap)[1:17]]
head(owlmap)
```

```
##   round_start_time  round_end_time                  stage match_id game_number
## 1  6/04/2023 21:03 6/04/2023 21:06 2023: Spring Knockouts    41901           1
## 2  6/04/2023 21:07 6/04/2023 21:13 2023: Spring Knockouts    41901           1
## 3  6/04/2023 21:21 6/04/2023 21:27 2023: Spring Knockouts    41901           2
## 4  6/04/2023 21:29 6/04/2023 21:32 2023: Spring Knockouts    41901           2
## 5  6/04/2023 21:39 6/04/2023 21:48 2023: Spring Knockouts    41901           3
## 6  6/04/2023 21:49 6/04/2023 21:54 2023: Spring Knockouts    41901           3
##     match_winner             map_winner             map_loser
## 1 Florida Mayhem         Florida Mayhem Los Angeles Gladiators
## 2 Florida Mayhem         Florida Mayhem Los Angeles Gladiators
## 3 Florida Mayhem         Florida Mayhem Los Angeles Gladiators
## 4 Florida Mayhem         Florida Mayhem Los Angeles Gladiators
## 5 Florida Mayhem Los Angeles Gladiators         Florida Mayhem
## 6 Florida Mayhem Los Angeles Gladiators         Florida Mayhem
##             map_name map_round winning_team_final_map_score
## 1              Oasis         1                            2
## 2              Oasis         2                            2
## 3     Blizzard World         1                            2
## 4     Blizzard World         2                            2
## 5 Shambali Monastery         1                            2
## 6 Shambali Monastery         2                            2
##   losing_team_final_map_score control_round_name               Attacker
## 1                           2            Gardens         Florida Mayhem
## 2                           2        City Center         Florida Mayhem
## 3                           2                    Los Angeles Gladiators
## 4                           2                            Florida Mayhem
## 5                           0                    Los Angeles Gladiators
## 6                           0                            Florida Mayhem
##                 Defender  team_one_name          team_two_name
## 1 Los Angeles Gladiators Florida Mayhem Los Angeles Gladiators
## 2 Los Angeles Gladiators Florida Mayhem Los Angeles Gladiators
## 3         Florida Mayhem Florida Mayhem Los Angeles Gladiators
## 4 Los Angeles Gladiators Florida Mayhem Los Angeles Gladiators
## 5         Florida Mayhem Florida Mayhem Los Angeles Gladiators
## 6 Los Angeles Gladiators Florida Mayhem Los Angeles Gladiators
```

The map dataframe now contains information for each round, for each map in each match.

However we do not want observations for each round for each map, I am interested in the outcomes of each map, not each round. Meaning I need to group the round observations by map.

```
#This creates a dataframe of the maximum round of each map for each map
owl_max_round<- owlmap %>%
  group_by(match_id, map_name) %>%
  summarize(
    max_map_round = max(map_round)
)
```

```
## `summarise()` has grouped output by 'match_id'. You can override using the
## `.groups` argument.
```

```
#This will then be used to inner join with the original map dataframe, resulting in only the last round
names(owl_max_round)[3] <- "map_round"
owlmap <- inner_join(owlmap, owl_max_round, names(owl_max_round))
head(owlmap)
```

```
##   round_start_time  round_end_time                   stage match_id game_number
## 1  6/04/2023 21:07 6/04/2023 21:13 2023: Spring Knockouts    41901           1
## 2  6/04/2023 21:29 6/04/2023 21:32 2023: Spring Knockouts    41901           2
## 3  6/04/2023 21:49 6/04/2023 21:54 2023: Spring Knockouts    41901           3
## 4  6/04/2023 22:02 6/04/2023 22:12 2023: Spring Knockouts    41901           4
## 5  6/04/2023 22:24 6/04/2023 22:30 2023: Spring Knockouts    41901           5
## 6  4/09/2023 21:30 4/09/2023 21:34           2023: Pro-Am    41351           1
##             match_winner             map_winner              map_loser
## 1          Florida Mayhem         Florida Mayhem Los Angeles Gladiators
## 2          Florida Mayhem         Florida Mayhem Los Angeles Gladiators
## 3          Florida Mayhem Los Angeles Gladiators         Florida Mayhem
## 4          Florida Mayhem Los Angeles Gladiators         Florida Mayhem
## 5          Florida Mayhem         Florida Mayhem Los Angeles Gladiators
## 6 Los Angeles Gladiators Los Angeles Gladiators        Houston Outlaws
##              map_name map_round winning_team_final_map_score
## 1               Oasis         2                            2
## 2       Blizzard World         2                            2
## 3  Shambali Monastery         2                            2
## 4           EsperanÃ§a         1                            1
## 5 Antarctic Peninsula         2                            2
## 6               Ilios         3                            2
##   losing_team_final_map_score control_round_name               Attacker
## 1                           2        City Center         Florida Mayhem
## 2                           2                            Florida Mayhem
## 3                           0                            Florida Mayhem
## 4                           0                    Los Angeles Gladiators
## 5                           2               Labs         Florida Mayhem
## 6                           2              Ruins Los Angeles Gladiators
##                  Defender          team_one_name          team_two_name
## 1 Los Angeles Gladiators         Florida Mayhem Los Angeles Gladiators
## 2 Los Angeles Gladiators         Florida Mayhem Los Angeles Gladiators
## 3 Los Angeles Gladiators         Florida Mayhem Los Angeles Gladiators
## 4          Florida Mayhem         Florida Mayhem Los Angeles Gladiators
## 5 Los Angeles Gladiators         Florida Mayhem Los Angeles Gladiators
## 6        Houston Outlaws Los Angeles Gladiators        Houston Outlaws
```

Now the two dataframes need to be joined.

```
#Loading packages and creating database connection
library(DBI)
library(RSQLite)
conn <- dbConnect(SQLite(), "new_db.sqlite")
```

```
#Writing dataframes into tables in the database
write.table(owlplayer, file="owlplayer.csv", sep=",", row.names=FALSE, col.names=TRUE)
write.table(owlmap, file="owlmap.csv", sep=",", row.names=FALSE, col.names=TRUE)

dbWriteTable(conn, "Players", owlplayer, overwrite=TRUE)
dbWriteTable(conn, "Maps", owlmap, overwrite=TRUE)
```

```
#Joining tables with sql query and storing it in an R object
owl <- dbGetQuery(conn, "SELECT *
           FROM Players p JOIN Maps m
           ON p.match_id = m.match_id AND p.map_name = m.map_name")
```

```
dbDisconnect(conn)
head(owl)
```

```
##                start_time match_id tournament_title map_type map_name
## 1 2023-03-23 20:11:00 UTC    41215           Pro-Am  control    Nepal
## 2 2023-03-23 20:11:00 UTC    41215           Pro-Am  control    Nepal
## 3 2023-03-23 20:11:00 UTC    41215           Pro-Am  control    Nepal
## 4 2023-03-23 20:11:00 UTC    41215           Pro-Am  control    Nepal
## 5 2023-03-23 20:11:00 UTC    41215           Pro-Am  control    Nepal
## 6 2023-03-23 20:11:00 UTC    41215           Pro-Am  control    Nepal
##   player_name           team_name All Damage Done Assists Average Time Alive
## 1     CHORONG      Florida Mayhem        4618.946      23           82.42819
## 2   Checkmate      Florida Mayhem       12551.992      11          189.47101
## 3       MER1T      Florida Mayhem       14806.654      10           63.22481
## 4       Rupal      Florida Mayhem        4176.770      18          117.82980
## 5     Someone      Florida Mayhem       12399.901       7          162.30850
## 6        FiNN San Francisco Shock        6569.945      15           72.48205
##   Barrier Damage Done Damage - Quick Melee Damage Done Damage Taken Deaths
## 1             565.000              30.0000    3983.946     4547.275      8
## 2            1050.500             172.2399   11449.787     4733.221      3
## 3            1671.000               0.0000   13022.255     6763.295      6
## 4            1135.000               0.0000    3001.770     4440.840      2
## 5            1316.160             372.1397   10736.507    13262.493      3
## 6            2255.374               0.0000    4314.571     4122.527      7
##   Defensive Assists Eliminations Final Blows Healing Done Hero Damage Done
## 1                21           12           2     8377.252         3983.946
## 2                 0           21          10        0.000        11449.787
## 3                 0           17           7        0.000        13022.255
## 4                20            7           3    13784.699         3001.770
## 5                 0           27          10        0.000        10736.507
## 6                15           10           2    11525.995         4314.571
##   Knockback Kills Objective Contest Time
## 1               3                 49.385
## 2               0                 46.543
## 3               0                 21.175
## 4               0                 40.754
## 5               1                 61.859
## 6               0                 42.784
##   Objective Contest Time - Avg per 10 Min Objective Contest Time - Most in Game
## 1                              0.06274022                                49.385
## 2                              0.05912966                                46.543
## 3                              0.02690137                                21.175
## 4                              0.05177514                                40.754
## 5                              0.07858758                                61.859
## 6                              0.05435411                                42.784
##   Objective Kills Objective Time Offensive Assists Shots Fired Time Alive
## 1               5         63.728                14         931    697.257
## 2               8         68.992                 0        6350    761.168
## 3               3         29.407                 0         602    721.321
## 4               2         58.247                 5         387    764.323
## 5              10         76.503                 1         191    754.305
## 6               5         67.067                 2         556    717.338
##   Time Building Ultimate Time Elapsed per Ultimate Earned Time Holding Ultimate
## 1              697.081                        135.65918              58.820
## 2              458.114                         84.59154             328.152
## 3              614.430                         99.09087             151.438
## 4              704.980                        103.16755              41.097
## 5              670.750                        127.25773              88.374
## 6              595.631                         99.59567             153.659
##   Time Played Ultimates Earned - Fractional Ultimates Used Weapon Accuracy
## 1    787.1346                     5.138473              4       0.1504113
## 2    787.1346                     5.415601              5       0.3562128
## 3    787.1346                     6.200672              5       0.3920553
## 4    787.1346                     6.833350              4       0.2225705
## 5    787.1346                     5.270800              4       0.4627660
## 6    787.1346                     5.980491              4       0.1891892
##   Melee Final Blows Melee Percentage of Final Blows Solo Kills Damage Blocked
## 1                 0                             0.0          0           0.00
## 2                 1                             0.1          2           0.00
## 3                 0                             0.0          0           0.00
## 4                 0                             0.0          0           0.00
## 5                 2                             0.2          0       14737.62
## 6                 0                             0.0          0           0.00
##   Environmental Kills Environmental Deaths Multikills Recon Assists
## 1                   0                    0          0             0
## 2                   0                    0          0             0
## 3                   0                    0          0             0
## 4                   0                    0          0             0
## 5                   1                    0          0             0
## 6                   0                    0          0             0
##   Turrets Destroyed Teleporter Pads Destroyed round_start_time round_end_time
## 1                 0                         0   03/23/23 20:22 03/23/23 20:26
## 2                 0                         0   03/23/23 20:22 03/23/23 20:26
## 3                 0                         0   03/23/23 20:22 03/23/23 20:26
## 4                 0                         0   03/23/23 20:22 03/23/23 20:26
## 5                 0                         0   03/23/23 20:22 03/23/23 20:26
## 6                 0                         0   03/23/23 20:22 03/23/23 20:26
##          stage match_id game_number   match_winner     map_winner
## 1 2023: Pro-Am    41215           1 Florida Mayhem Florida Mayhem
## 2 2023: Pro-Am    41215           1 Florida Mayhem Florida Mayhem
## 3 2023: Pro-Am    41215           1 Florida Mayhem Florida Mayhem
## 4 2023: Pro-Am    41215           1 Florida Mayhem Florida Mayhem
## 5 2023: Pro-Am    41215           1 Florida Mayhem Florida Mayhem
## 6 2023: Pro-Am    41215           1 Florida Mayhem Florida Mayhem
##             map_loser map_name map_round winning_team_final_map_score
## 1 San Francisco Shock    Nepal         3                            2
## 2 San Francisco Shock    Nepal         3                            2
## 3 San Francisco Shock    Nepal         3                            2
## 4 San Francisco Shock    Nepal         3                            2
## 5 San Francisco Shock    Nepal         3                            2
## 6 San Francisco Shock    Nepal         3                            2
##   losing_team_final_map_score control_round_name       Attacker
## 1                           2            Sanctum Florida Mayhem
## 2                           2            Sanctum Florida Mayhem
## 3                           2            Sanctum Florida Mayhem
## 4                           2            Sanctum Florida Mayhem
## 5                           2            Sanctum Florida Mayhem
## 6                           2            Sanctum Florida Mayhem
##             Defender  team_one_name       team_two_name
## 1 San Francisco Shock Florida Mayhem San Francisco Shock
## 2 San Francisco Shock Florida Mayhem San Francisco Shock
## 3 San Francisco Shock Florida Mayhem San Francisco Shock
## 4 San Francisco Shock Florida Mayhem San Francisco Shock
## 5 San Francisco Shock Florida Mayhem San Francisco Shock
## 6 San Francisco Shock Florida Mayhem San Francisco Shock
```

- As stated in the brief, I am only interested in data after 05/05/2022 (start of Overwatch 2). Meaning I will need to format the dates into the correct format, as they are currently in the 'character' format.

The dates in this data set are formatted in two different ways for the 'round_end_time' variable, m/d/y and m/d/Y. I will convert both and then replace the correct dates with an index

```
#index of dates that are m/d/y (not m/d/Y) using regular expression
timeindex <- grepl("^\\d{2}/\\d{2}/\\d{2} \\d{2}:\\d{2}$", owl$round_end_time)

timechange1 <- as.POSIXct(owl$round_end_time, format="%m/%d/%y %H:%M", tz="UTC")
timechange2 <- as.POSIXct(owl$round_end_time, format="%m/%d/%Y %H:%M", tz="UTC")
timechange1 <- na.omit(timechange1)
timechange2 <- na.omit(timechange2)

#Replacing the m/d/Y first, then m/d/y.
owl$round_end_time <- timechange2
owl$round_end_time[timeindex] <- timechange1
```

Now converting 'round_start_time' to a date:

```
owl$start_time <- as.POSIXct(owl$start_time, format = "%Y-%m-%d %H:%M:%S", tz = "UTC")
```

```
#Filtering out observations before 05/05/2022
owl <- na.omit(owl[(owl$round_end_time > as.POSIXct("2022/05/05 01:00", tz="UTC")),])
min(owl$round_end_time)
```

```
## [1] "2022-05-05 19:25:00 UTC"
```

The 'round_start_time' variable from *owl_map* is now redundant, as it is not the start time of the game, but the final round of that game. Instead I will use the start time variable from the player data when the dataframes are joined.

```
owl[,"round_start_time"] <- list(NULL)
```

The player dataframe now contains all the information variables and most predictor variables I need to do analysis. But I still need to create a response variable that will be predicted by the model.

```
#Loop that iterates through each row, and stores a value for the outcome of each map
for(i in 1:nrow(owl)){
  if(owl$map_winner[i]==owl$team_name[i]){
    owl$win_map[i] <- 1
  }else if(owl$map_winner[i]=="draw"){
```

```r
    owl$win_map[i] <- 2
  }else{
    owl$win_map[i] <- 0
  }
}

#Loop that iterates through each row, and stores a value for the outcome of each match
for(i in 1:nrow(owl)){
  if(owl$match_winner[i]==owl$team_name[i]){
    owl$win_match[i] <- 1
  }else if(owl$match_winner[i]=="draw"){
    owl$win_match[i] <- 2
  }else{
    owl$win_match[i] <- 0
  }
}
```

Joining the two dataframes has created two duplicate columns:

```r
names(owl)[duplicated(names(owl))]
```

```
## [1] "match_id" "map_name"
```

Which will be removed:

```r
owl <- owl[, -c(49, 54)]
```

```r
any(duplicated(names(owl)))
```

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

Some variable names need to have underscores placed between words to ensure they work with glm() function:

```r
#Placing underscores in each variable name
owlrmv_ <- gsub(" ", "_", names(owl))
names(owl) <- owlrmv_
#Removing minus and underscore that some variables have
owlrmv_minus <- gsub("_-", "", names(owl))
names(owl) <- owlrmv_minus
```

To quickly check the validity of the data, I will check to make sure that there are 10 observations (one for each player) for a given map in a given match:

```r
nrow(owl[owl$match_id==41215 & owl$map_name=="Nepal",])
```

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

The data appears to have been cleaned correctly.

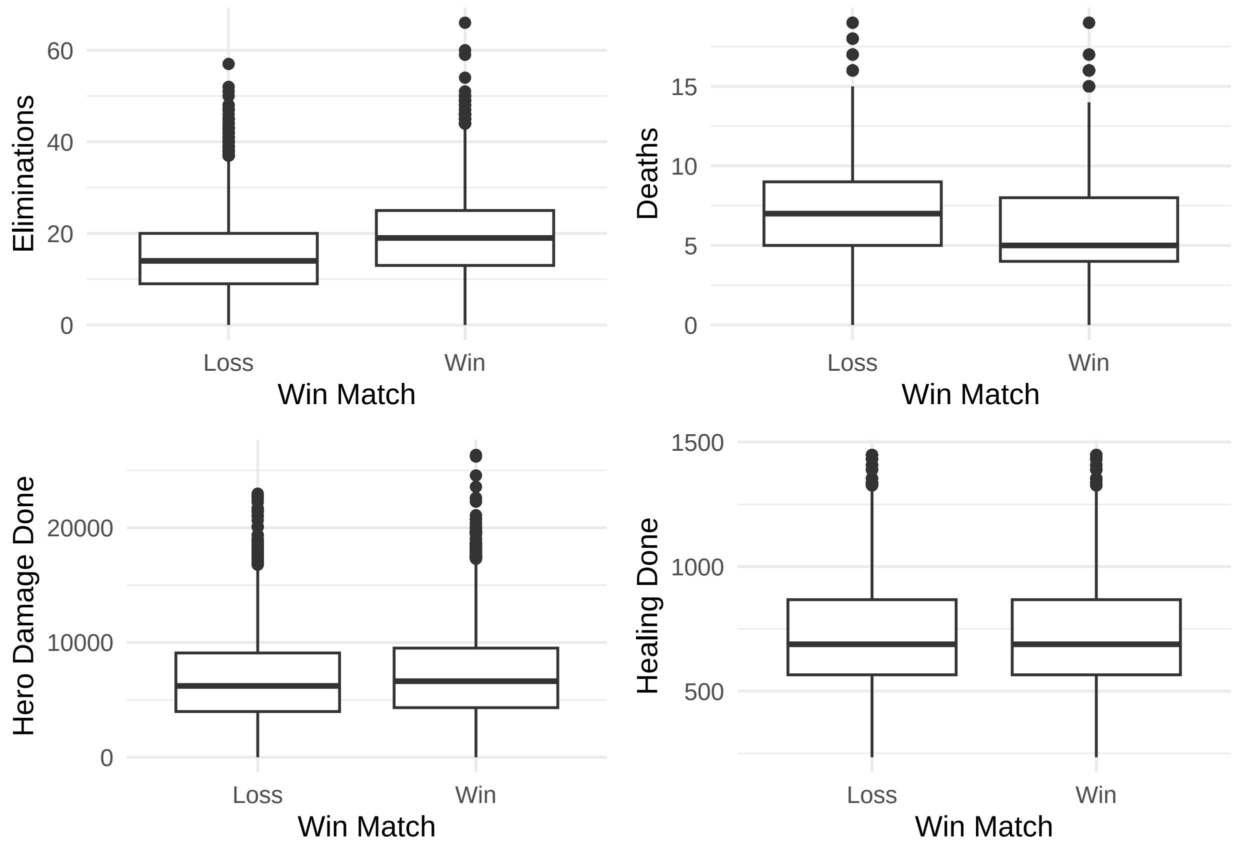
Reordering columns for simplicity:

```
owl <- owl[,c("start_time", "round_end_time", "match_id", "tournament_title", "map_type", "map_name", "
names(owl)[1] <- "round_start_time"
```

#Exploratory analysis:

```
elim_plot <- ggplot(owl, aes(x = factor(win_match), y = Eliminations)) +
  geom_boxplot() +
  labs(x = "Win Match",
       y = "Eliminations") +
  scale_x_discrete(labels = c("Loss", "Win")) +
  theme_minimal()
deaths_plot <- ggplot(owl, aes(x = factor(win_match), y = Deaths)) +
  geom_boxplot() +
  labs(x = "Win Match",
       y = "Deaths") +
  scale_x_discrete(labels = c("Loss", "Win")) +
  theme_minimal()
damage_plot <- ggplot(owl, aes(x = factor(win_match), y = Hero_Damage_Done)) +
  geom_boxplot() +
  labs(x = "Win Match",
       y = "Hero Damage Done") +
  scale_x_discrete(labels = c("Loss", "Win")) +
  theme_minimal()
healing_plot <- ggplot(owl, aes(x = factor(win_match), y = Time_Played)) +
  geom_boxplot() +
  labs(x = "Win Match",
       y = "Healing Done") +
  scale_x_discrete(labels = c("Loss", "Win")) +
  theme_minimal()

plot_grid(elim_plot, deaths_plot, damage_plot, healing_plot)
```

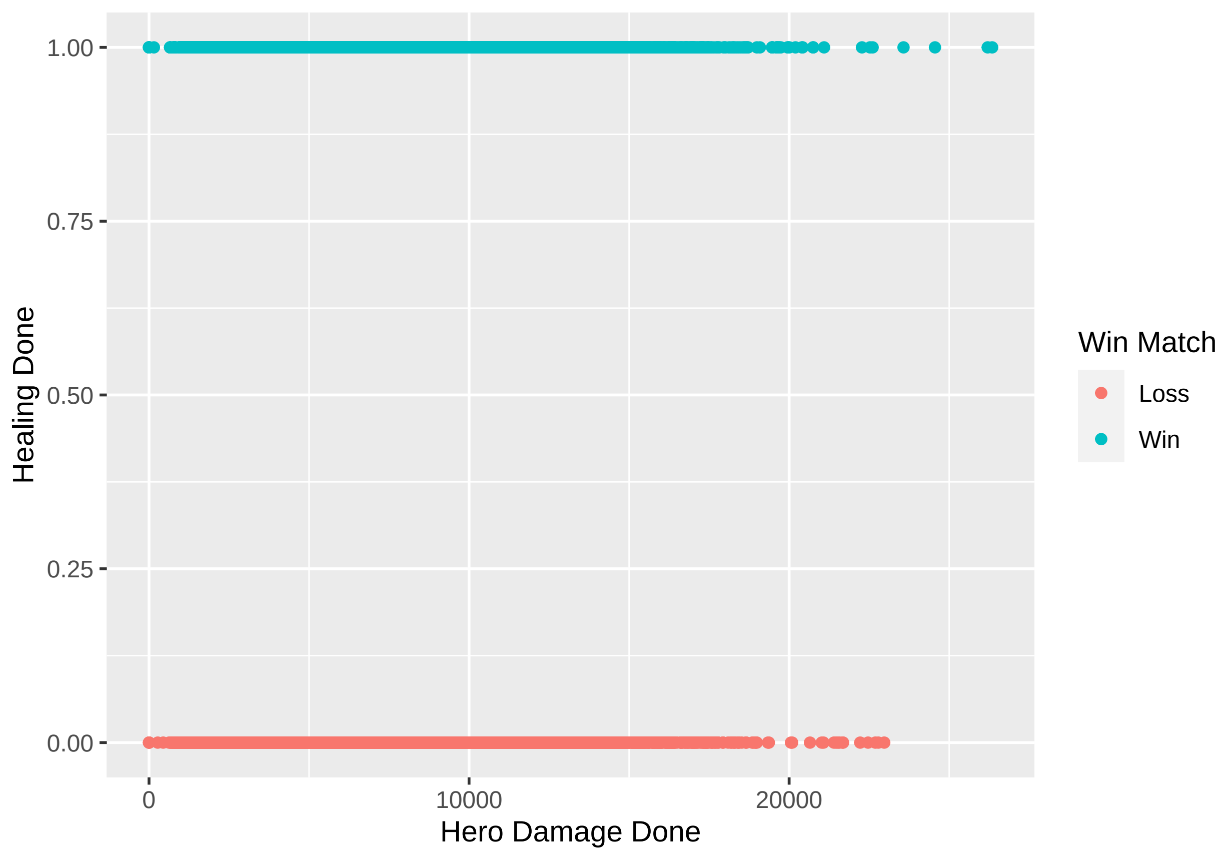


These boxplots illustrate some of the likely relationships in this dataset. Variable 'Eliminations' clealy has a positive correlation with winning. While 'Deaths' has a negative one.

Variable 'Hero_Damage_Done' appears to have a slight difference in win rate, while healing done doesn't appear to because of the significant amount of 0 values.

This scatter plot better illustrates the relationship between 'Hero_Damage_Done' and winning.

```r
ggplot(owl, aes(x = Hero_Damage_Done, y = win_match, color = factor(win_match))) +
  geom_point() +
  labs(y = "Healing Done",
       x = "Hero Damage Done",
       color = "Win Match") +
  scale_color_discrete(name = "Win Match",
                       labels = c("Loss", "Win"))
```



#Model selection and evaluation

splitting into test and train set:

```
set.seed(1)
train <- sample(nrow(owl), nrow(owl)*0.7, replace=FALSE)
train_owl <- owl[train,]
test_owl <- owl[-train,]
```

##fitting full model:

```
owl_logreg <- glm(win_match ~ All_Damage_Done + Assists + Average_Time_Alive + Barrier_Damage_Done + Dan
```

```
summary(owl_logreg)
```

```
##
## Call:
## glm(formula = win_match ~ All_Damage_Done + Assists + Average_Time_Alive +
##     Barrier_Damage_Done + Damage_Quick_Melee + Damage_Taken +
##     Deaths + Defensive_Assists + Eliminations + Final_Blows +
##     Healing_Done + Hero_Damage_Done + Knockback_Kills + Objective_Contest_Time +
##     Objective_Contest_Time_Avg_per_10_Min + Objective_Kills +
##     Objective_Time + Offensive_Assists + Shots_Fired + Time_Alive +
##     Time_Building_Ultimate + Time_Elapsed_per_Ultimate_Earned +
##     Time_Holding_Ultimate + Time_Played + Ultimates_Earned_Fractional +
##     Ultimates_Used + Weapon_Accuracy + Melee_Final_Blows + Melee_Percentage_of_Final_Blows +
##     Solo_Kills + Damage_Blocked + Environmental_Kills + Environmental_Deaths +
##     Multikills + Recon_Assists + Turrets_Destroyed + Teleporter_Pads_Destroyed,
##     family = "binomial", data = train_owl)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5641  -0.9614   0.3247   0.9698   2.6133
##
## Coefficients:
##                                        Estimate Std. Error z value Pr(>|z|)
## (Intercept)                           4.827e-01  3.362e-01   1.436 0.151117
## All_Damage_Done                      -2.592e-06  1.171e-05  -0.221 0.824812
## Assists                               2.476e-02  1.300e-02   1.904 0.056889
## Average_Time_Alive                   -1.113e-04  9.040e-04  -0.123 0.902020
## Barrier_Damage_Done                  -1.143e-06  1.799e-05  -0.064 0.949338
## Damage_Quick_Melee                    2.049e-05  8.776e-05   0.233 0.815395
## Damage_Taken                         -5.440e-06  9.949e-06  -0.547 0.584515
## Deaths                               -7.075e-01  4.996e-02 -14.161  < 2e-16
## Defensive_Assists                     1.913e-02  4.911e-03   3.895 9.83e-05
## Eliminations                          6.675e-02  1.421e-02   4.698 2.63e-06
## Final_Blows                           2.910e-02  1.665e-02   1.748 0.080420
## Healing_Done                         -1.149e-06  1.343e-05  -0.086 0.931841
## Hero_Damage_Done                     -6.726e-05  1.936e-05  -3.474 0.000512
## Knockback_Kills                      -1.569e-02  7.010e-03  -2.238 0.025211
## Objective_Contest_Time               -5.741e-04  5.631e-03  -0.102 0.918784
## Objective_Contest_Time_Avg_per_10_Min  9.434e-02  3.825e+00   0.025 0.980324
## Objective_Kills                       9.166e-03  8.763e-03   1.046 0.295559
## Objective_Time                       -2.307e-03  6.020e-04  -3.832 0.000127
## Offensive_Assists                     3.206e-03  4.778e-03   0.671 0.502195
## Shots_Fired                          -3.134e-06  1.449e-05  -0.216 0.828754
## Time_Alive                           -4.840e-02  5.063e-03  -9.559  < 2e-16
## Time_Building_Ultimate                4.397e-03  2.465e-03   1.784 0.074472
## Time_Elapsed_per_Ultimate_Earned     -4.632e-03  2.370e-03  -1.954 0.050681
## Time_Holding_Ultimate                 3.527e-03  2.311e-03   1.526 0.126931
## Time_Played                           4.553e-02  5.539e-03   8.219  < 2e-16
## Ultimates_Earned_Fractional          -1.158e-01  6.660e-02  -1.739 0.082047
## Ultimates_Used                        3.028e-02  3.593e-02   0.843 0.399308
## Weapon_Accuracy                       1.752e-01  2.296e-01   0.763 0.445299
## Melee_Final_Blows                     1.959e-02  3.697e-02   0.530 0.596140
## Melee_Percentage_of_Final_Blows      -4.805e-02  1.694e-01  -0.284 0.776700
## Solo_Kills                           -3.937e-02  3.012e-02  -1.307 0.191194
## Damage_Blocked                        1.473e-05  8.287e-06   1.778 0.075448
## Environmental_Kills                   3.129e-02  8.904e-02   0.351 0.725273
## Environmental_Deaths                  6.361e-02  7.845e-02   0.811 0.417437
## Multikills                            9.032e-02  5.391e-02   1.675 0.093855
## Recon_Assists                         1.518e-01  4.160e-02   3.649 0.000263
## Turrets_Destroyed                     9.547e-02  7.598e-02   1.256 0.208941
## Teleporter_Pads_Destroyed            -2.191e-01  2.139e-01  -1.024 0.305648
##
## (Intercept)
## All_Damage_Done
## Assists                                .
## Average_Time_Alive
## Barrier_Damage_Done
## Damage_Quick_Melee
## Damage_Taken
## Deaths                                 ***
## Defensive_Assists                      ***
## Eliminations                           ***
## Final_Blows                            .
## Healing_Done
## Hero_Damage_Done                       ***
## Knockback_Kills                        *
## Objective_Contest_Time
## Objective_Contest_Time_Avg_per_10_Min
## Objective_Kills
## Objective_Time                         ***
## Offensive_Assists
## Shots_Fired
## Time_Alive                             ***
## Time_Building_Ultimate                 .
## Time_Elapsed_per_Ultimate_Earned       .
## Time_Holding_Ultimate
## Time_Played                            ***
## Ultimates_Earned_Fractional            .
## Ultimates_Used
## Weapon_Accuracy
## Melee_Final_Blows
## Melee_Percentage_of_Final_Blows
## Solo_Kills
## Damage_Blocked                         .
## Environmental_Kills
## Environmental_Deaths
## Multikills                             .
## Recon_Assists                          ***
## Turrets_Destroyed
## Teleporter_Pads_Destroyed
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 14158  on 10212  degrees of freedom
## Residual deviance: 12076  on 10175  degrees of freedom
## AIC: 12152
##
## Number of Fisher Scoring iterations: 3
```

Evaluation metrics:

```
threshold <- 0.5
preds <- predict(owl_logreg, newdata = test_owl, type = "response")
pred_labels <- ifelse(preds >= threshold, 1, 0)

# AUC calculation
```

```
auc <- roc(test_owl$win_match, preds)$auc
```

```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```

```
# Classification accuracy
acc <- mean(test_owl$win_match == pred_labels)

# Precision
precision <- sum(preds[test_owl$win_match == 1] >= threshold) / sum(preds >= threshold)

# Recall
recall <- sum(preds[test_owl$win_match == 1] >= threshold) / sum(test_owl$win_match == 1)

cat(paste("AUC:", round(auc, 4), "\n"))
```

```
## AUC: 0.7535
```

```
cat(paste("Accuracy:", round(acc, 4), "\n"))
```

```
## Accuracy: 0.7016
```

```
cat(paste("Precision:", round(precision, 4), "\n"))
```

```
## Precision: 0.6982
```

```
cat(paste("Recall:", round(recall, 4), "\n"))
```

```
## Recall: 0.7033
```

The full model appears to predict somewhat well. This full model likely has some highly correlated variables:

```
pander(vif(owl_logreg))
```

Table 1: Table continues below

All_Damage_Done	Assists	Average_Time_Alive	Barrier_Damage_Done
7.935	11.22	2.354	2.67

Table 2: Table continues below

Damage_Quick_Melee	Damage_Taken	Deaths	Defensive_Assists	Eliminations
2.218	5.361	40.74	4.521	27.85

Table 3: Table continues below

Final_Blows	Healing_Done	Hero_Damage_Done	Knockback_Kills
11.29	10.14	10.23	1.904

Table 4: Table continues below

Objective_Contest_Time	Objective_Contest_Time_Avg_per_10_Min
25.14	24.97

Table 5: Table continues below

Objective_Kills	Objective_Time	Offensive_Assists	Shots_Fired
2.462	2.465	2.039	2.231

Table 6: Table continues below

Time_Alive	Time_Building_Ultimate	Time_Elapsed_per_Ultimate_Earned
1865	373	7.603

Table 7: Table continues below

Time_Holding_Ultimate	Time_Played	Ultimates_Earned_Fractional
62.5	2630	25.3

Table 8: Table continues below

Ultimates_Used	Weapon_Accuracy	Melee_Final_Blows
6.346	1.659	2.892

Table 9: Table continues below

Melee_Percentage_of_Final_Blows	Solo_Kills	Damage_Blocked
2.217	1.436	3.514

Table 10: Table continues below

Environmental_Kills	Environmental_Deaths	Multikills	Recon_Assists
1.148	1.045	1.178	1.226

Turrets_Destroyed	Teleporter_Pads_Destroyed
1.272	1.254

There are some very heavily correlated variables in this dataset. This is likely because there are variables recording very similar things, or the same thing in a different way. For example 'Time_Alive' and 'Time_Building_Ultimate' both have unreasonably high VIF scores of 1865 and 373 respectively. This is likely because ultimates are constantly being built while players are playing the game. I will remove the least important predictor, rather than using regularization or PCA. I will remove variables based on the relative significance and my domain knowledge.

Model with variables 'Time_Played', 'Objective_Contest_Time', 'Final_Blows', 'Time_Building_Ultimate', 'Ultimates_Earned_Fractional', 'All_Damage_Done' removed:

```
owl_logreg_reduced <- glm(win_match ~  Assists + Average_Time_Alive + Barrier_Damage_Done + Damage_Quic
```

```
pander(vif(owl_logreg_reduced))
```

Table 12: Table continues below

Assists	Average_Time_Alive	Barrier_Damage_Done	Damage_Quick_Melee
4.805	2.281	1.456	2.134

Table 13: Table continues below

Damage_Taken	Deaths	Defensive_Assists	Eliminations	Healing_Done
5.259	3.373	4.46	7.731	8.253

Table 14: Table continues below

Hero_Damage_Done	Knockback_Kills	Objective_Contest_Time_Avg_per_10_Min
7.025	1.861	1.096

Table 15: Table continues below

Objective_Time	Offensive_Assists	Shots_Fired
2.022	1.991	2.022

Table 16: Table continues below

Time_Elapsed_per_Ultimate_Earned	Time_Holding_Ultimate	Ultimates_Used
2.823	1.857	4.62

Table 17: Table continues below

Weapon_Accuracy	Melee_Final_Blows	Melee_Percentage_of_Final_Blows
1.641	2.765	2.158

Table 18: Table continues below

Solo_Kills	Damage_Blocked	Environmental_Kills	Time_Alive
1.387	3.305	1.12	7.568

Table 19: Table continues below

Environmental_Deaths	Multikills	Recon_Assists	Turrets_Destroyed
1.041	1.172	1.204	1.27

Teleporter_Pads_Destroyed
1.255

This signifcant correlations have been removed, as all variables are < 10.

```
summary(owl_logreg_reduced)
```

```
##
## Call:
## glm(formula = win_match ~ Assists + Average_Time_Alive + Barrier_Damage_Done +
##     Damage_Quick_Melee + Damage_Taken + Deaths + Defensive_Assists +
##     Eliminations + Healing_Done + Hero_Damage_Done + Knockback_Kills +
##     Objective_Contest_Time_Avg_per_10_Min + Objective_Time +
##     Offensive_Assists + Shots_Fired + Time_Elapsed_per_Ultimate_Earned +
##     Time_Holding_Ultimate + Ultimates_Used + Weapon_Accuracy +
##     Melee_Final_Blows + Melee_Percentage_of_Final_Blows + Solo_Kills +
##     Damage_Blocked + Environmental_Kills + Time_Alive + Environmental_Deaths +
##     Multikills + Recon_Assists + Turrets_Destroyed + Teleporter_Pads_Destroyed,
##     family = "binomial", data = train_owl)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5616  -0.9708   0.3178   0.9844   2.4208
##
## Coefficients:
##                                         Estimate Std. Error z value Pr(>|z|)
## (Intercept)                           -6.267e-02  2.240e-01  -0.280 0.779658
## Assists                                9.744e-03  8.483e-03   1.149 0.250695
## Average_Time_Alive                    -1.198e-03  8.764e-04  -1.367 0.171736
## Barrier_Damage_Done                   -4.283e-06  1.321e-05  -0.324 0.745719
## Damage_Quick_Melee                    -2.319e-05  8.565e-05  -0.271 0.786565
## Damage_Taken                          -7.403e-07  9.788e-06  -0.076 0.939708
## Deaths                             -2.446e-01  1.433e-02 -17.070  < 2e-16
## Defensive_Assists                   2.187e-02  4.872e-03   4.489 7.17e-06
## Eliminations                        9.415e-02  7.455e-03  12.630  < 2e-16
## Healing_Done                       -1.035e-05  1.208e-05  -0.857 0.391422
## Hero_Damage_Done                   -7.285e-05  1.597e-05  -4.562 5.06e-06
## Knockback_Kills                    -1.819e-02  6.908e-03  -2.634 0.008441
## Objective_Contest_Time_Avg_per_10_Min -9.342e-02  7.942e-01  -0.118 0.906372
## Objective_Time                     -2.377e-03  5.422e-04  -4.385 1.16e-05
## Offensive_Assists                   2.915e-03  4.708e-03   0.619 0.535774
## Shots_Fired                        -7.699e-06  1.370e-05  -0.562 0.574167
## Time_Elapsed_per_Ultimate_Earned    2.479e-04  1.431e-03   0.173 0.862514
## Time_Holding_Ultimate               3.475e-04  3.961e-04   0.877 0.380377
## Ultimates_Used                      3.004e-02  3.047e-02   0.986 0.324296
## Weapon_Accuracy                     1.569e-01  2.271e-01   0.691 0.489655
## Melee_Final_Blows                   2.808e-02  3.591e-02   0.782 0.434264
## Melee_Percentage_of_Final_Blows    -1.158e-01  1.660e-01  -0.698 0.485371
## Solo_Kills                         -3.510e-02  2.944e-02  -1.192 0.233169
## Damage_Blocked                      7.885e-06  7.985e-06   0.987 0.323438
## Environmental_Kills                 1.283e-02  8.815e-02   0.146 0.884271
## Time_Alive                          6.321e-04  3.208e-04   1.970 0.048831
## Environmental_Deaths                6.869e-02  7.804e-02   0.880 0.378813
## Multikills                          8.389e-02  5.356e-02   1.566 0.117293
## Recon_Assists                       1.504e-01  4.077e-02   3.690 0.000225
## Turrets_Destroyed                   9.611e-02  7.451e-02   1.290 0.197068
## Teleporter_Pads_Destroyed          -2.138e-01  2.130e-01  -1.004 0.315533
##
## (Intercept)
## Assists
## Average_Time_Alive
## Barrier_Damage_Done
## Damage_Quick_Melee
## Damage_Taken
## Deaths                             ***
## Defensive_Assists                  ***
## Eliminations                       ***
## Healing_Done
## Hero_Damage_Done                   ***
## Knockback_Kills                    **
## Objective_Contest_Time_Avg_per_10_Min
## Objective_Time                     ***
## Offensive_Assists
## Shots_Fired
## Time_Elapsed_per_Ultimate_Earned
## Time_Holding_Ultimate
## Ultimates_Used
## Weapon_Accuracy
## Melee_Final_Blows
## Melee_Percentage_of_Final_Blows
## Solo_Kills
## Damage_Blocked
## Environmental_Kills
## Time_Alive                         *
## Environmental_Deaths
## Multikills
## Recon_Assists                         ***
## Turrets_Destroyed
## Teleporter_Pads_Destroyed
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 14158  on 10212  degrees of freedom
## Residual deviance: 12185  on 10182  degrees of freedom
## AIC: 12247
##
## Number of Fisher Scoring iterations: 4
```

```r
threshold <- 0.5
preds <- predict(owl_logreg_reduced, newdata = test_owl, type = "response")
pred_labels <- ifelse(preds >= threshold, 1, 0)

# AUC calculation
auc <- roc(test_owl$win_match, preds)$auc
```

```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```

```r
# Classification accuracy
acc <- mean(test_owl$win_match == pred_labels)

# Precision
precision <- sum(preds[test_owl$win_match == 1] >= threshold) / sum(preds >= threshold)

# Recall
recall <- sum(preds[test_owl$win_match == 1] >= threshold) / sum(test_owl$win_match == 1)

cat(paste("AUC:", round(auc, 4), "\n"))
```

```
## AUC: 0.7485
```

```r
cat(paste("Accuracy:", round(acc, 4), "\n"))
```

```
## Accuracy: 0.6996
```

```r
cat(paste("Precision:", round(precision, 4), "\n"))
```

```
## Precision: 0.6967
```

```r
cat(paste("Recall:", round(recall, 4), "\n"))
```

```
## Recall: 0.6996
```

However the evaluation metrics show the model is marginally worse at predicting.

#Model selection Now that the highly correlated varaibles have been removed, I will find the variables that appear to be the most informative.

Performing forwards, backwards and stepwise model selection with both AIC and BIC.

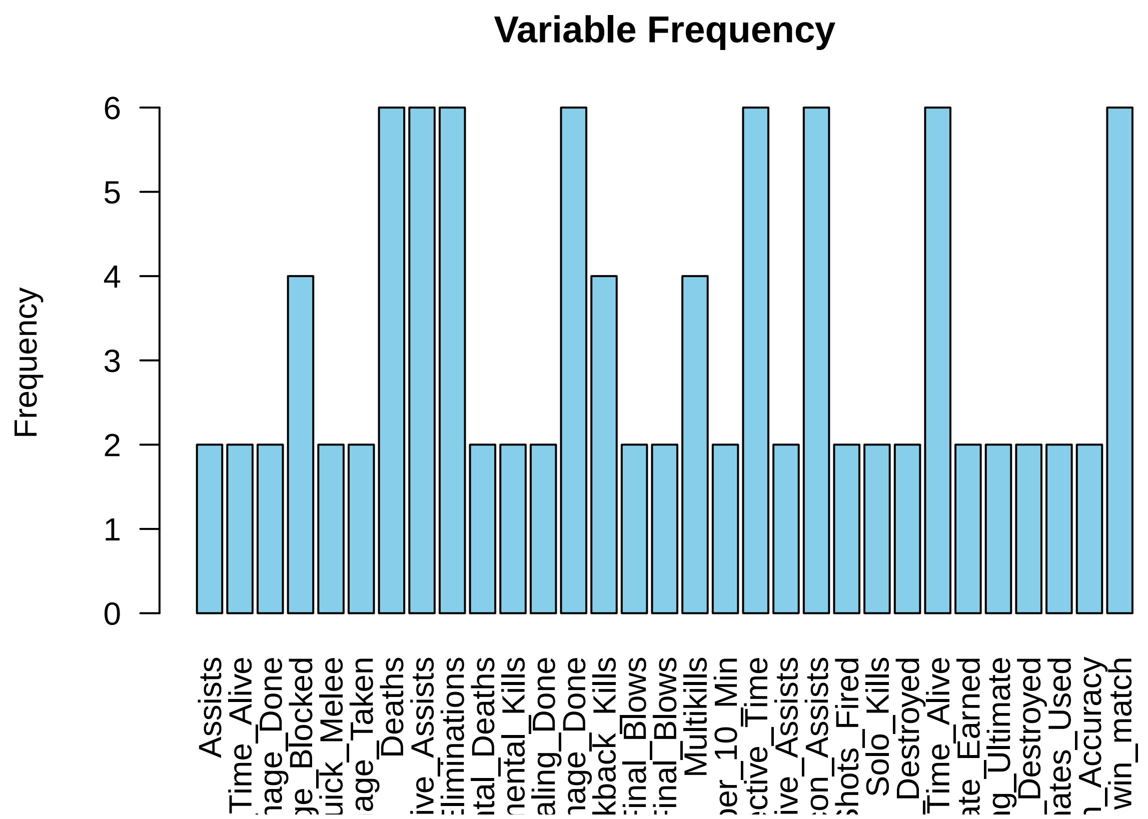
```r
forward_modelaic <- stepAIC(owl_logreg_reduced, direction = "forward", scope = formula(owl_logreg_reduc
backward_modelaic <- stepAIC(owl_logreg_reduced, direction = "backward", scope = formula(owl_logreg_redu
stepwise_modelaic <- stepAIC(owl_logreg_reduced, direction = "both", scope = formula(owl_logreg_reduced

forward_modelbic <- stepAIC(owl_logreg_reduced, direction = "forward", k = log(nrow(train_owl)),
                         scope = formula(owl_logreg_reduced))
backward_modelbic <- stepAIC(owl_logreg_reduced, direction = "backward", k = log(nrow(train_owl)),
                          scope = formula(owl_logreg_reduced))
stepwise_modelbic <- stepAIC(owl_logreg_reduced, direction = "both", k = log(nrow(train_owl)),
                          scope = formula(owl_logreg_reduced))
```

```r
predictors <- c(all.vars(formula(stepwise_modelaic)), all.vars(formula(stepwise_modelbic)), all.vars(fo
```

```r
pred_freq <- table(predictors)


# Plot the bar plot
barplot(pred_freq, las = 2, col = "skyblue", main = "Variable Frequency", ylab = "Frequency")
```



This barplot illustrates the frequency of variables in all of the models selectd. Variables ‘Deaths’, ‘Denfensive_Assits', 'Hero_Damge_Done', 'Objective_Time', 'Recon_Assits', 'Time_Alive' are included in all 6 models.

```
pred_freq[pred_freq==max(pred_freq)]
```

```
## predictors
##            Deaths Defensive_Assists      Eliminations  Hero_Damage_Done
##                 6                 6                 6                 6
##    Objective_Time     Recon_Assists        Time_Alive         win_match
##                 6                 6                 6                 6
```

These variables are therefore the most informative, and will be used for the future model.

Some variables not included here, such as 'Healing_Done' or 'Ultimates_Used' may be significant in the context of the future model predicting team vs team rather than individual players overall.

This study was successful in getting a better understanding of the most informative variables in this context. This will help inform my further model building with this dataset.