Tennis Machine Learning Report

Ying Liu

2/1/2020

1. Introduction

This report is to discuss the creation of a machine learning algorithm that predicts tennis matches using the dataset made available kindly by Jeff Sackmann available online here: https://github.com/JeffSackmann/tennis_atp.

The goal of the project is to produce a machine learning model which predicts the winner of a tennis match. The accuracy metric used will be the number of correct predictions divided by total prediction; this is used to evaulate the performance of the algorithm. This assignment will attempt to predict as accurately as possible the outcome of a group of tennis matches and compare the predictions to the actual result.

In Jeff Sackmann's ATP dataset, he lists every match played by year in csv files. He also lists details of players as well as players rankings in other files. For this project we will only use match results from the last 10 years (2010 to 2019).

Another dataset was used during the data analysis phase. The Game Insight Group produced player ratings for a group of players in early 2019. The dataset is available online here: https://www.perfect-tennis.com/player-dna/

The key steps to this project were:

1. Prepare the problem via loading libraries & dataset and splitting out the validation set
2. Summarise data and key statistics via tables or graphs
3. Clean and wrangle data whilst considering feature selection
4. Evaulate a series of machine learning algorithmns on the training set and improve results; step 3 will be revisited during this step to support the process
5. Report on results; create a standalone model for predictions on the validation set and conclusion

2. Method / Analysis

2.1 Load Libraries

The following libraries are required to run the code in the rest of the report:

```
# Libraries required for the rest of the code is located here
if(!require(pbapply)) install.packages("pbapply", repos = "http://cran.us.r-project.org")
if(!require(tidyverse)) install.packages("tidyverse", repo = "http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("lubridate", repo = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repo = "http://cran.us.r-project.org")
if(!require(magrittr)) install.packages("magrittr", repo = "http://cran.us.r-project.org")
if(!require(ggrepel)) install.packages("ggrepel", repo = "http://cran.us.r-project.org")
if(!require(sjstats)) install.packages("sjstats", repos = "http://cran.us.r-project.org")
if(!require(mice)) install.packages("mice", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(Rborist)) install.packages("Rborist", repos = "http://cran.us.r-project.org")
if(!require(e1071)) install.packages("e1071", repos = "http://cran.us.r-project.org")
if(!require(RSNNS)) install.packages("RSNNS", repos = "http://cran.us.r-project.org")
if(!require(neuralnet)) install.packages("neuralnet", repos = "http://cran.us.r-project.org")
if(!require(kableExtra)) install.packages("kableExtra", repos = "http://cran.us.r-project.org")
```

2.2 Introduction to Tennis

Tennis is a sport played with racquets that is either against a single opponent or two teams of two players. Single opponent tennis is more popular and easier to model and we will focus on the men's singles competition.

In the game, one player is either the server or the receiver and the opponents stand on opposite sides of a rectangular court. The court is cut off by a standard net strectched across the width of the court. The court can have different surfaces which changes the dynamics of the match.

The scoring system is complex. A match between 2 players is either best of 3 or 5 sets depending on the tournament. To win a set a player needs to be the first to win 6 games. The player also needs to reach 6 games before the other player wins 5 games. To win a set you need to win by two games, so therefore a set can be won 7-5. This means in theory a set could last forever if each player keeps winning alternative games; this had happened in the past where matchs have taken over 5 hours to complete. To combat this, if a set is 6 games all, a tie break is played and the winner of the tie break wins the set 7-6. Players take turns to serve in alternate games. For more details on how tennis is scored or played please see this link: https://en.wikipedia.org/wiki/Tennis

Professional tournaments take place 11 months of the year for both men and women in separate competitions. We will only look at the men's tour.

2.3 Load Data

Before running the code please ensure that the data folder is downloaded and in your working directory project folder. The following code is used to load our dataset:

```
# Load dataframes
atp <- do.call(rbind, lapply(list.files(pattern = "atp_matches_201\\d.csv"), fread))
dna <- read.csv("playerDNA.csv")
```

In our ATP dataset from 2010 - 2019 there is a collection of 29337 matches with a range of match stats and player stats for each of these matches. The match stats in dataset need to be refined for us to predict on. It does not make sense to feed stats from completed matches into our machine learning algorithmns, because we'd be feeding the result of the match into the model and then trying to predict from it. What needs to be done is to collate stats on each player's prior matches using the stats from those matches to try to predict who will win. This will be done in the data wrangling section.

The DNA dataset, shows stats for 56 players. Each player is given a rating for their technical, tactical, phyical and mental abilities, these ratings range from 0 to 100. However we cannot use this dataset for our modelling, only data analysis. This is due to the fact all these ratings are for the players at one specific point in time and would change as players progress. They do however provide insights into results. The details of how these numbers were derived are available on the link in the introduction to the report. To use this data set we need to be able to link it to the main atp dataset. This is done below:

```
# Convert player name to character string and rename
dna <- dna %>%
  mutate(Player, player = as.character(dna$Player)) %>%
  select(-Player)

# Check player names between atp and dna datasets, pull players who don't match then split out the name
noNames <- atp %>%
  select(winner_name, loser_name) %>%
  gather(key = result, value = player) %>%
  group_by(player) %>%
  summarise(wins = sum(result == "winner_name"),
            games = n()) %>%
  right_join(dna, by = "player") %>%
  filter(is.na(games)) %>%
```

```
  separate_rows(player, sep = "\\ ") %>%
  pull(player)

# Check names in atp dataset
atp %>%
  select(winner_name, loser_name) %>%
  gather(key = result, value = player) %>%
  group_by(player) %>%
  filter(grepl(paste(noNames, collapse = "|"), player)) %>%
  summarise(count = n())
```

```
## # A tibble: 20 x 2
##    player                             count
##    <chr>                              <int>
##  1 Albert Montanes                      230
##  2 Albert Ramos                         439
##  3 Alberto Emmanuel Alvarado Larin        9
##  4 Alberto Lim                            7
##  5 Alberto Martin                         9
##  6 Guillermo Alcaide                      1
##  7 Guillermo Garcia Lopez               426
##  8 Guillermo Hormazabal                   1
##  9 Guillermo Olaso                        6
## 10 Guillermo Rivera Aranguiz              1
## 11 Jo Wilfried Tsonga                   541
## 12 Juan Pablo Brzezicki                   2
## 13 Juan Pablo Varillas Patino Samudio    11
## 14 Pablo Andujar                        307
## 15 Pablo Carreno Busta                  328
## 16 Pablo Cuevas                         338
## 17 Pablo Galdon                           1
## 18 Pablo Nunez                            2
## 19 Stanislas Wawrinka                   562
## 20 Stanislav Vovk                         1
```

```
# Replace player names which don't match
dna$player <- dna$player %>%
  str_replace("-", " ") %>%
  str_replace("Stan Wawrinka", "Stanislas Wawrinka") %>%
  str_replace("Albert Ramos Vinolas", "Albert Ramos")

# Remove unused values
rm(noNames)
```

After converting the variable name of the player in the DNA dataset, we check which players don't appear in both datasets. The atp dataset holds every match, so every player that is given a DNA values should appear in the ATP dataset. The check shows that the use of "-" and shortening of names causes any mismatches. These are replaced in the DNA dataset, so that the DNA dataset matches the names in ATP.

2.4 Summarise Data

We will use the 56 players in the DNA dataset for our summary statistics. These players will have played quite a lot of games and therefore will generate more signal than noise.

2.4.1 Descriptive Stats

```
# Games, wins, win% by court type
atp %>%
  select(winner_name, loser_name, surface) %>%
  filter(winner_name %in% dna$player) %>%
  filter(loser_name %in% dna$player) %>%
  gather(key = result, value = name, -surface) %>%
  group_by(name) %>%
  summarise(wins = sum(result == "winner_name"),
            games = n(),
            winPerc = wins / games,
            grassWins = sum(result == "winner_name" & surface == "Grass"),
            grassGames = sum(surface == "Grass"),
            grassWinPerc = sum(result == "winner_name" & surface == "Grass") / sum(surface == "Grass"),
            clayWins = sum(result == "winner_name" & surface == "Clay"),
            clayGames = sum(surface == "Clay"),
            clayWinPerc = sum(result == "winner_name" & surface == "Clay") / sum(surface == "Clay"),
            hardWins = sum(result == "winner_name" & surface == "Hard"),
            hardGames = sum(surface == "Hard"),
            hardWinPerc = sum(result == "winner_name" & surface == "Hard") / sum(surface == "Hard")) %>%
  filter(games > 15) %>%
  arrange(desc(winPerc)) %>%
  slice(1:25) %>%
  knitr::kable() %>%
  kable_styling(latex_options = c("striped", "scale_down"))
```

name	wins	games	winPerc	grassWins	grassGames	grassWinPerc	clayWins	clayGames	clayWinPerc	hardWins	hardGames	hardWinPerc
Novak Djokovic	371	449	0.826	31	39	0.795	82	104	0.788	258	306	0.843
Rafael Nadal	311	392	0.793	18	25	0.720	144	164	0.878	149	203	0.734
Roger Federer	284	372	0.763	49	58	0.845	39	59	0.661	196	255	0.769
Andy Murray	232	327	0.709	39	48	0.812	42	67	0.627	150	211	0.711
Juan Martin Del Potro	142	216	0.657	17	24	0.708	25	41	0.610	100	151	0.662
Dominic Thiem	122	206	0.592	5	12	0.417	58	86	0.674	59	108	0.546
Stanislas Wawrinka	172	291	0.591	12	22	0.545	53	90	0.589	107	179	0.598
Alexander Zverev	94	160	0.588	8	16	0.500	32	50	0.640	54	94	0.574
Nick Kyrgios	79	138	0.572	9	18	0.500	12	24	0.500	58	94	0.617
David Ferrer	165	289	0.571	7	16	0.438	66	104	0.635	92	169	0.544
Kei Nishikori	152	269	0.565	9	23	0.391	43	69	0.623	100	177	0.565
Tomas Berdych	175	312	0.561	15	26	0.577	37	66	0.561	123	220	0.559
Gael Monfils	126	229	0.550	10	17	0.588	26	54	0.481	89	157	0.567
Jo Wilfried Tsonga	143	260	0.550	19	34	0.559	30	58	0.517	94	168	0.560
Milos Raonic	130	242	0.537	18	32	0.562	19	44	0.432	93	166	0.560
John Isner	120	237	0.506	7	15	0.467	25	51	0.490	88	171	0.515
Marin Cilic	147	293	0.502	20	34	0.588	27	60	0.450	100	199	0.503
Grigor Dimitrov	128	256	0.500	12	22	0.545	23	55	0.418	93	179	0.520
Roberto Bautista Agut	107	218	0.491	7	17	0.412	22	53	0.415	78	147	0.531
Andrey Rublev	35	72	0.486	1	3	0.333	6	13	0.462	28	56	0.500
David Goffin	97	202	0.480	8	19	0.421	23	47	0.489	66	136	0.485
Richard Gasquet	118	255	0.463	11	24	0.458	34	66	0.515	72	164	0.439
Alex De Minaur	24	53	0.453	1	4	0.250	0	4	0.000	23	45	0.511
Jack Sock	55	122	0.451	1	5	0.200	14	25	0.560	40	92	0.435
Pablo Carreno Busta	66	149	0.443	1	4	0.250	29	61	0.475	36	84	0.429

This table shows the 25 players in the DNA dataset with the highest win percentage in the last 10 years; also listed is the number of wins they had in the period and number of games they played. Each player's record by court type is then further broken down. A quick glance shows variablity in performance by court type.

```
# Correlation of all players in ATP data set
atp %>%
  select(winner_name, loser_name, surface) %>%
  gather(key = result, value = player, -surface) %>%
  group_by(player) %>%
  mutate(winPerc = sum(result == "winner_name") / n(),
         grassWinPerc = sum(result == "winner_name" & surface == "Grass") / sum(surface == "Grass"),
```

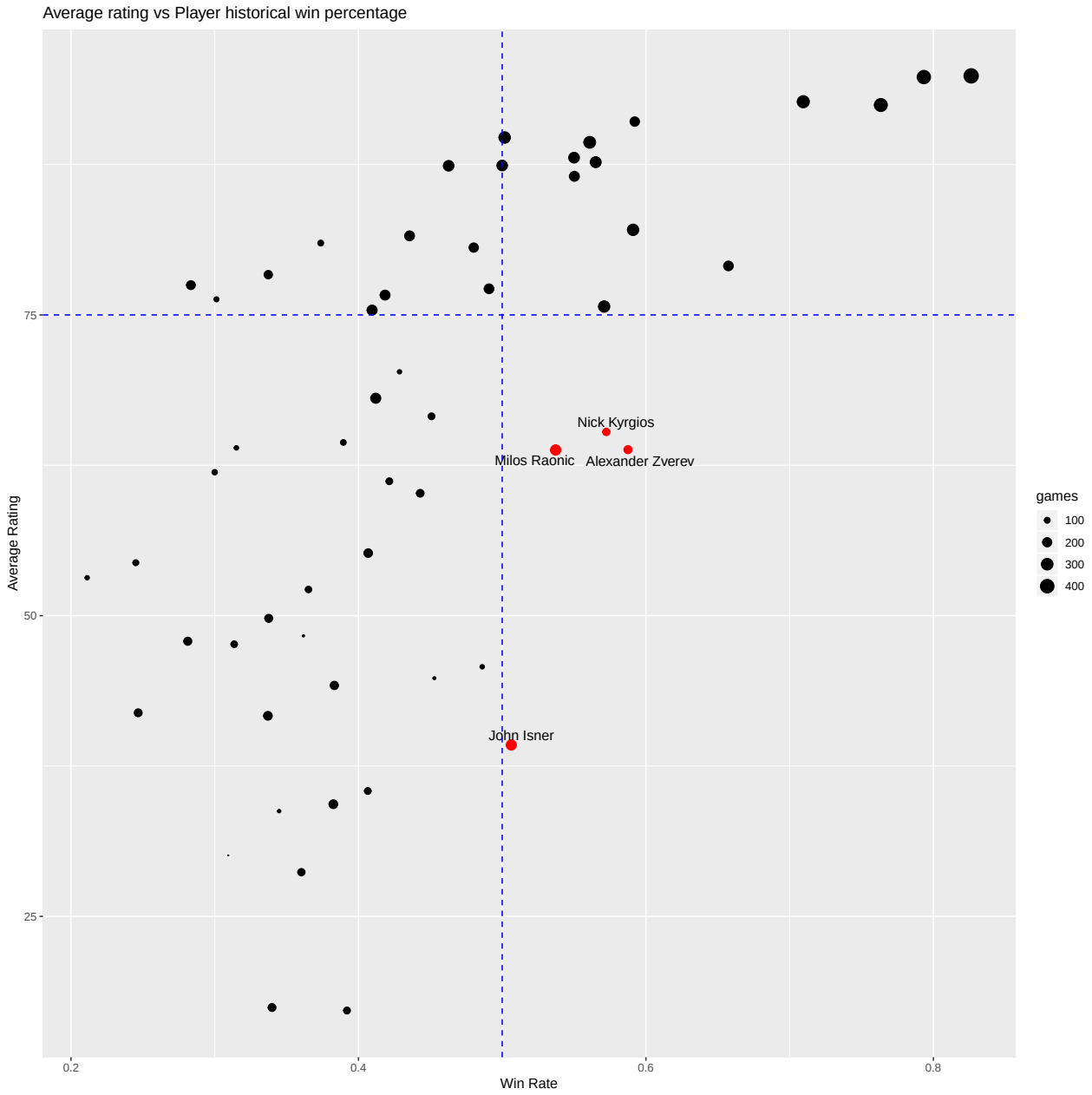
```
        clayWinPerc = sum(result == "winner_name" & surface == "Clay") / sum(surface == "Clay"),
        hardWinPerc = sum(result == "winner_name" & surface == "Hard") / sum(surface == "Hard")) %>%
  ungroup() %>%
  select(-c(surface, result, player)) %>%
  distinct() %>%
  cor(use = "complete.obs")
```

```
##              winPerc grassWinPerc clayWinPerc hardWinPerc
## winPerc        1.000        0.612       0.655       0.788
## grassWinPerc   0.612        1.000       0.237       0.449
## clayWinPerc    0.655        0.237       1.000       0.321
## hardWinPerc    0.788        0.449       0.321       1.000
```

The table above shows the correlation of winning by court type and player. This table shows that different surfaces have a major effect on player's performance.

2.4.2 Data Visualisation

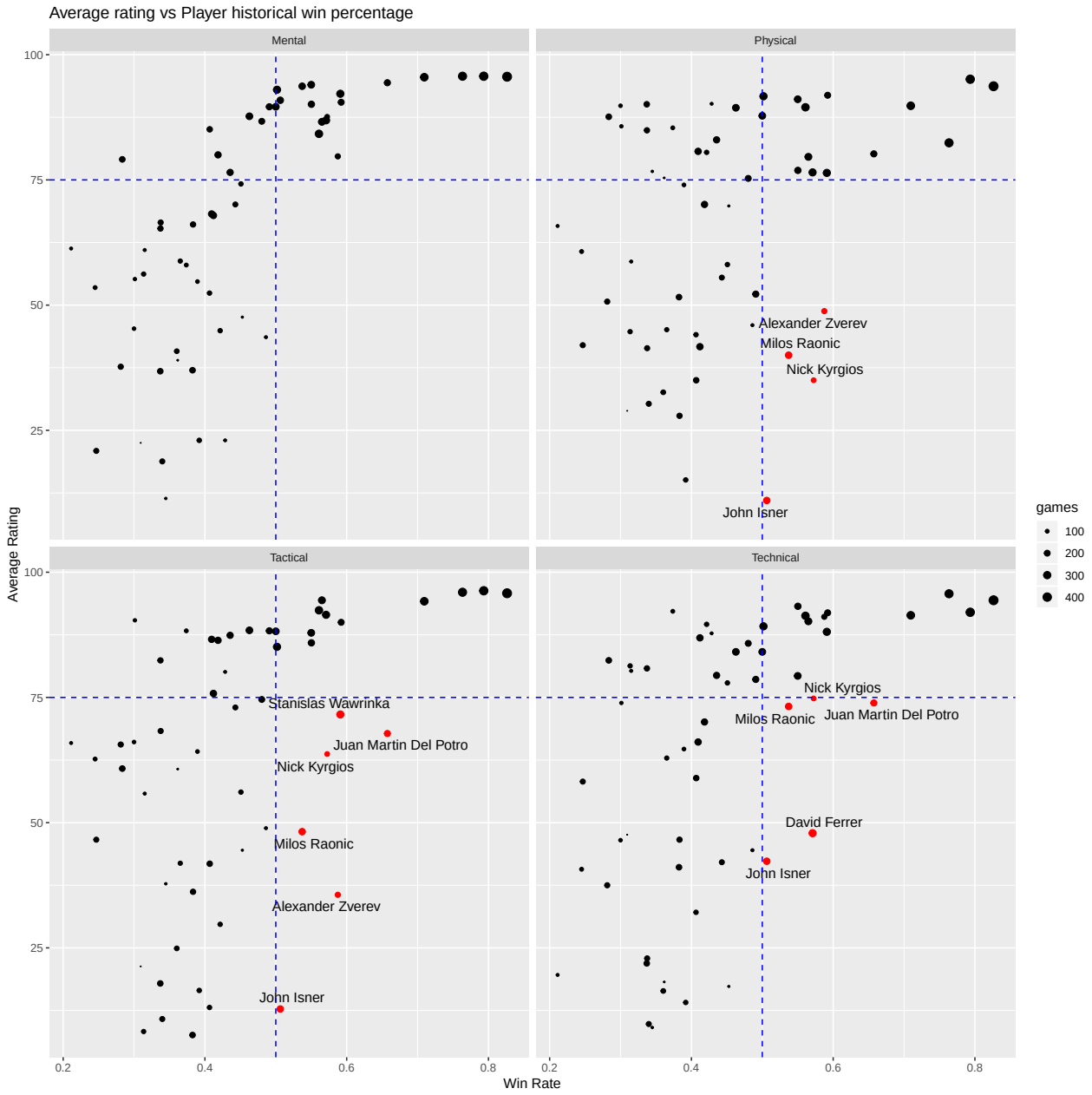
```
# Graph of win percentage vs ratings by average of traits
atp %>% select(winner_name, loser_name) %>%
  filter(winner_name %in% dna$player) %>%
  filter(loser_name %in% dna$player) %>%
  gather(key = result, value = player) %>%
  group_by(player) %>%
  summarise(wins = sum(result == "winner_name"),
            games = n(),
            winPerc = wins / games) %>%
  right_join(dna, by = "player") %>%
  mutate(rating = (Mental + Physical + Tactical + Technical)/4) %>%
  ggplot(aes(winPerc, rating, label = player)) +
  geom_point(aes(size = games,
                 color = ifelse(winPerc > 0.5 & rating < 75, "red", "black"))) +
  scale_color_identity() +
  scale_size(range = c(0,5)) +
  geom_text_repel(aes(label = ifelse(winPerc > 0.5 & rating < 75, as.character(player), ""))) +
  geom_hline(yintercept = 75,
             linetype = "dashed",
             color = "blue") +
  geom_vline(xintercept = 0.5,
             linetype = "dashed",
             color = "blue") +
  labs(title = "Average rating vs Player historical win percentage",
       y = "Average Rating",
       x = "Win Rate")
```



This graph plots the average rating of all 56 of our players against their historical win rate. A high DNA rating does not also equal a high win rate (Top left quadrant). Players with a high rating and a high win percentage have done so over a lot games; which indicates they deserve their rating. Players with both low ratings and win rates have usually played fewer games. This makes sense as they would usually be knocked out of a tournament earlier meaning they would play less games a year. Only a few players (In red) have a win rate of above 0.5 and an average rating of below 75. This is investigated further in the following graphs.

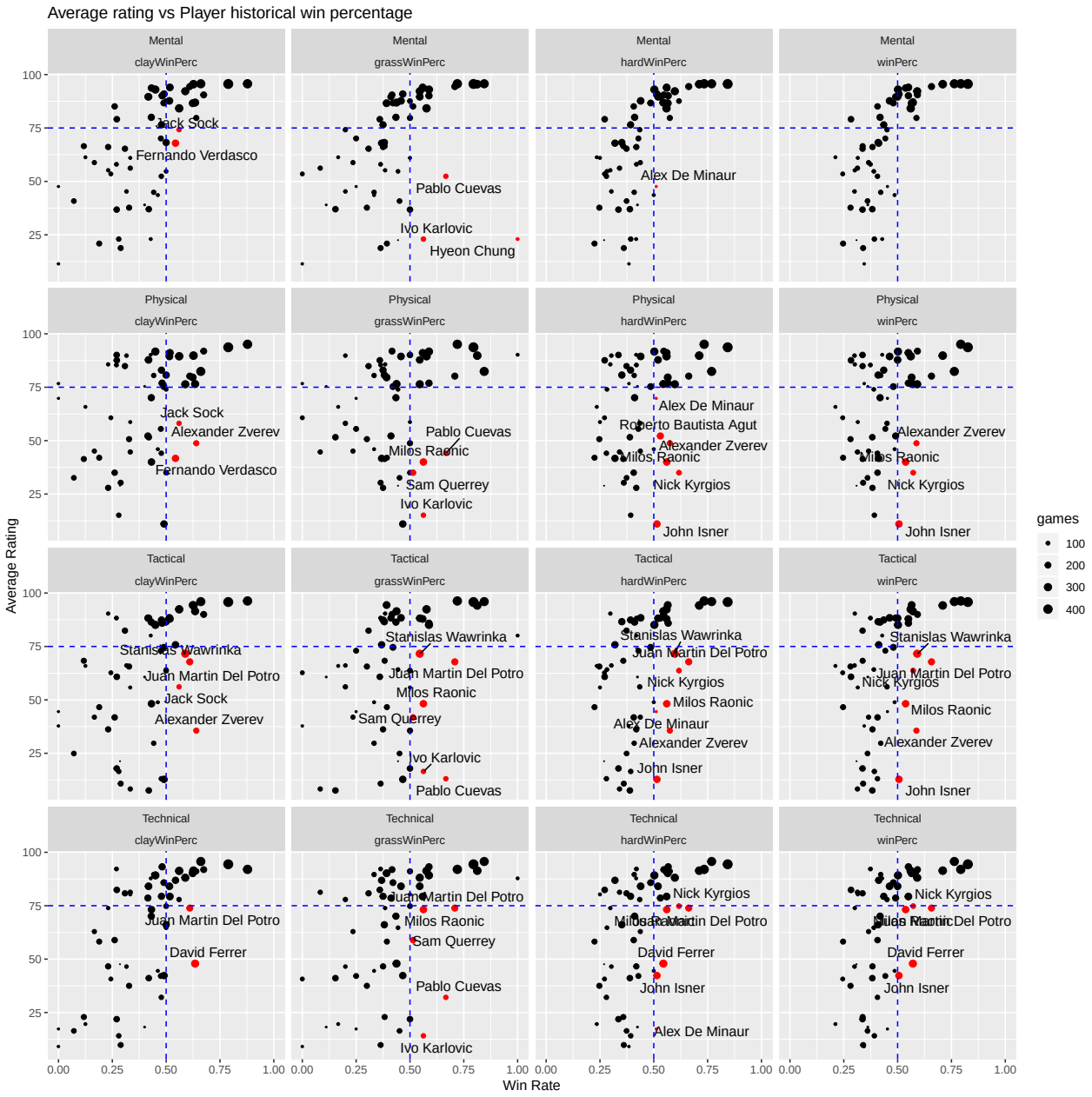
```
# Graph of win percentage vs ratings by trait
atp %>% select(winner_name, loser_name) %>%
  filter(winner_name %in% dna$player) %>%
  filter(loser_name %in% dna$player) %>%
  gather(key = result, value = player) %>%
  group_by(player) %>%
  summarise(wins = sum(result == "winner_name"),
```

```
          games = n(),
          winPerc = wins / games) %>%
right_join(dna, by = "player") %>%
gather(key = trait, value = rating, -c(player,winPerc,wins,games)) %>%
ggplot(aes(winPerc, rating)) +
geom_point(aes(size = games,
               color = ifelse(winPerc > 0.5 & rating < 75, "red", "black"))) +
scale_color_identity() +
scale_size(range = c(0,3)) +
facet_wrap(trait ~ .) +
geom_text_repel(aes(label = ifelse(winPerc > 0.5 & rating < 75, as.character(player), "")),
                point.padding = 0.1) +
geom_hline(yintercept = 75,
           linetype = "dashed",
           color = "blue") +
geom_vline(xintercept = 0.5,
           linetype = "dashed",
           color = "blue") +
labs(title = "Average rating vs Player historical win percentage",
     y = "Average Rating",
     x = "Win Rate")
```




```
# Win Perc by court type by trait
atp %>%
  select(winner_name, loser_name, surface) %>%
  filter(winner_name %in% dna$player) %>%
  filter(loser_name %in% dna$player) %>%
  gather(key = result, value = player, -surface) %>%
  group_by(player) %>%
  summarise(wins = sum(result == "winner_name"),
            games = n(),
            winPerc = wins / games,
            grassWins = sum(result == "winner_name" & surface == "Grass"),
            grassGames = sum(surface == "Grass"),
            grassWinPerc = sum(result == "winner_name" & surface == "Grass") / sum(surface == "Grass"),
            clayWins = sum(result == "winner_name" & surface == "Clay"),
```

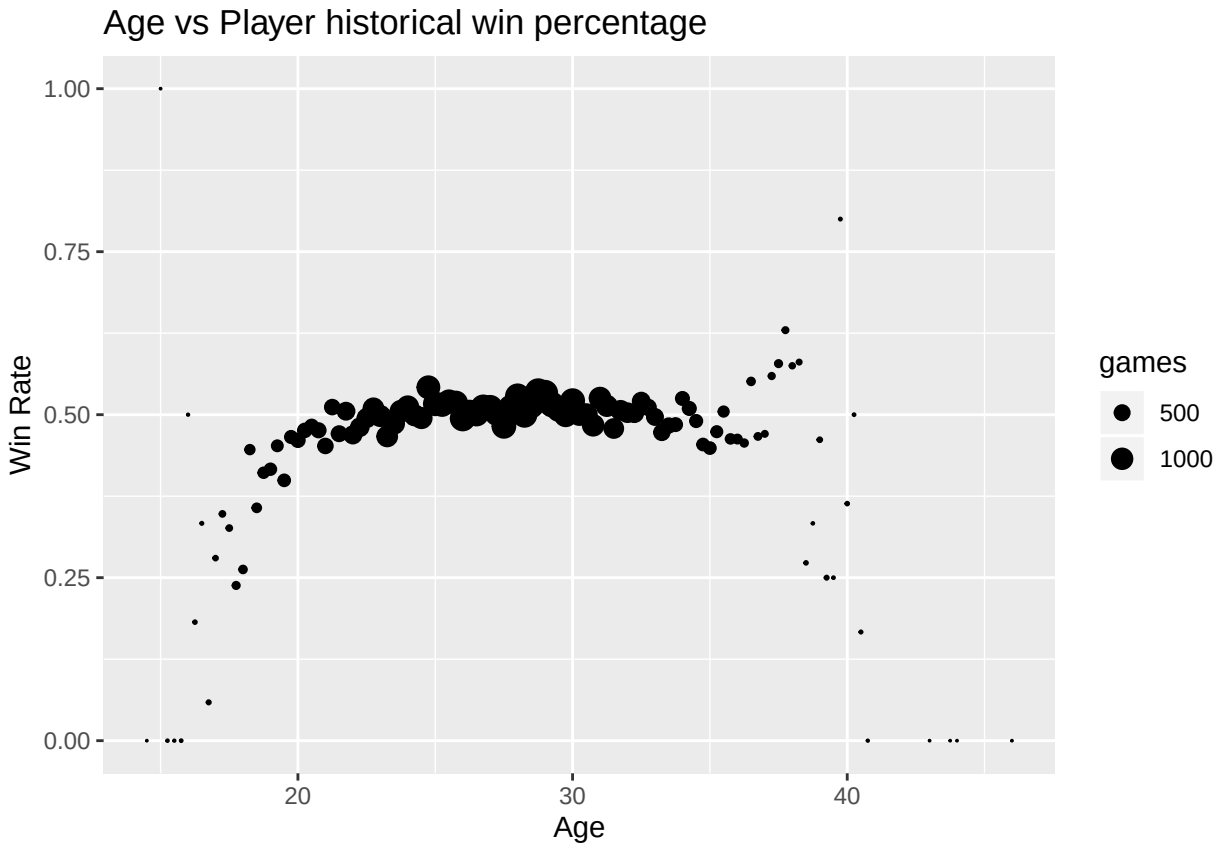
```
            clayGames = sum(surface == "Clay"),
            clayWinPerc = sum(result == "winner_name" & surface == "Clay") / sum(surface == "Clay"),
            hardWins = sum(result == "winner_name" & surface == "Hard"),
            hardGames = sum(surface == "Hard"),
            hardWinPerc = sum(result == "winner_name" & surface == "Hard") / sum(surface == "Hard")) %>%
  right_join(dna, by = "player") %>%
  gather(key = trait, value = rating, c(Technical, Tactical, Physical, Mental)) %>%
  gather(key = courtType, value = winP, c(winPerc, grassWinPerc, clayWinPerc, hardWinPerc)) %>%
  ggplot(aes(winP, rating)) +
  geom_point(aes(size = games,
                 color = ifelse(winP > 0.5 & rating < 75, "red", "black"))) +
  scale_color_identity() +
  scale_size(range = c(0,3)) +
  facet_wrap(trait ~ courtType) +
  geom_text_repel(aes(label = ifelse(winP > 0.5 & rating < 75, as.character(player), "")),
                  point.padding = 0.1) +
  geom_hline(yintercept = 75,
             linetype = "dashed",
             color = "blue") +
  geom_vline(xintercept = 0.5,
             linetype = "dashed",
             color = "blue") +
  labs(title = "Average rating vs Player historical win percentage",
       y = "Average Rating",
       x = "Win Rate")
```



What can be seen from these charts is that a player's mental rating is the most important. In the chart of average rating vs win percentage by trait we can see that there are no players with a mental rating of below 75 who also have a win rate of above 0.5. However a deficency in physical, tactical or technical prowess can be overcome and players with low ratings in those areas can still have win rates of above 0.5. The same trend is picked up when we separate the charts by court type; there are less players in the bottom right quadrant.

```
# Player performance by age
atp %>%
  select(winner_age, loser_age) %>%
  gather(key = result, value = age) %>%
  mutate(age = round(age*4)/4) %>%
  group_by(age) %>%
  mutate(winPerc = sum(result == "winner_age") / n(),
         games = n()) %>%
```

```
  select(-result) %>%
  distinct() %>%
  ggplot(aes(age, winPerc, size = games)) +
  geom_point() +
  scale_size(range = c(0,4)) +
  labs(title = "Age vs Player historical win percentage",
       y = "Win Rate",
       x = "Age")
```



A players age has some bearing on their performance, the graph above shows win rates by age. There are not many games played by players under 20 or over 35, win rates are also dramatically lower for these ages groups. Player performance peaks in their late twenties but are generally steady for a 10 year period. Most matches are played by players in their 20s.

2.5 Prepare data

There are currently 49 variables in our dataset, not all these are necessary or lend themselves to modelling. Country, hand etc are removed. Any factors with unknown values are assigned as unknown.

```
# Remove unnecessary variables & Convert factors
atp <- atp %>%
  select(-c(draw_size, match_num, winner_id, winner_seed, winner_entry, winner_ioc, loser_id, loser_seed
            w_SvGms, l_SvGms, winner_ht, loser_ht, score)) %>%
  mutate_at(vars(surface, tourney_level, winner_hand, loser_hand, round, best_of), as.factor)

# Adjust factors with no values to unknown values
atp$surface[which(atp$surface == "")] <- "None"
atp$surface <- factor(atp$surface)
```

```r
atp$winner_hand[which(atp$winner_hand == "")] <- "U"
atp$winner_hand <- factor(atp$winner_hand)
atp$loser_hand[which(atp$loser_hand == "")] <- "U"
atp$loser_hand <- factor(atp$loser_hand)
```

2.5.1 Data Cleansing - NAs

The remaining variables are analysed for their Na values.

```r
# Remove NA's
summary(atp)
```

```
##   tourney_id        tourney_name          surface      tourney_level
##  Length:29337       Length:29337       Carpet:   51   A:15669
##  Class :character   Class :character   Clay  : 9150   D: 2790
##  Mode  :character   Mode  :character   Grass : 3226   F:  136
##                                        Hard  :16712   G: 5080
##                                        None  :  198   M: 5662
##
##
##   tourney_date      winner_name        winner_hand   winner_age
##  Min.   :20100103   Length:29337       L: 3833     Min.   :14.9
##  1st Qu.:20120430   Class :character   R:25172     1st Qu.:24.6
##  Median :20141005   Mode  :character   U:  332     Median :27.4
##  Mean   :20144850                                  Mean   :27.4
##  3rd Qu.:20170501                                  3rd Qu.:30.1
##  Max.   :20191124                                  Max.   :40.6
##                                                    NA's   :4
##   loser_name        loser_hand   loser_age    best_of       round
##  Length:29337       L: 4050    Min.   :14.5   3:22478   R32    :9103
##  Class :character   R:24501    1st Qu.:24.2   5: 6859   R16    :5184
##  Mode  :character   U:  786    Median :27.2             R64    :4280
##                                Mean   :27.2             R128   :3200
##                                3rd Qu.:30.1             RR     :2992
##                                Max.   :46.0             QF     :2592
##                                NA's   :15               (Other):1986
##     minutes          w_ace           w_df          w_svpt        w_1stIn
##  Min.   :   0   Min.   :  0    Min.   : 0     Min.   :  0    Min.   :  0
##  1st Qu.:  78   1st Qu.:  3    1st Qu.: 1     1st Qu.: 57    1st Qu.: 35
##  Median : 100   Median :  6    Median : 2     Median : 73    Median : 45
##  Mean   : 108   Mean   :  7    Mean   : 3     Mean   : 78    Mean   : 48
##  3rd Qu.: 131   3rd Qu.: 10    3rd Qu.: 4     3rd Qu.: 95    3rd Qu.: 59
##  Max.   :1146   Max.   :113    Max.   :26     Max.   :491    Max.   :361
##  NA's   :3468   NA's   :2162   NA's   :2162   NA's   :2162   NA's   :2162
##     w_1stWon       w_2ndWon      w_bpSaved      w_bpFaced        l_ace
##  Min.   :  0    Min.   : 0     Min.   : 0     Min.   : 0     Min.   :  0
##  1st Qu.: 27    1st Qu.:12     1st Qu.: 1     1st Qu.: 2     1st Qu.:  2
##  Median : 34    Median :16     Median : 3     Median : 4     Median :  4
##  Mean   : 37    Mean   :17     Mean   : 3     Mean   : 5     Mean   :  5
##  3rd Qu.: 44    3rd Qu.:20     3rd Qu.: 5     3rd Qu.: 7     3rd Qu.:  7
##  Max.   :292    Max.   :82     Max.   :24     Max.   :30     Max.   :103
##  NA's   :2162   NA's   :2162   NA's   :2162   NA's   :2162   NA's   :2162
##       l_df          l_svpt         l_1stIn        l_1stWon       l_2ndWon
##  Min.   : 0     Min.   :  0    Min.   :  0    Min.   :  0    Min.   :  0
```

```
##  1st Qu.: 2     1st Qu.: 60    1st Qu.: 35    1st Qu.: 23    1st Qu.: 10
##  Median : 3     Median : 77    Median : 46    Median : 31    Median : 14
##  Mean   : 3     Mean   : 81    Mean   : 49    Mean   : 33    Mean   : 15
##  3rd Qu.: 5     3rd Qu.: 97    3rd Qu.: 59    3rd Qu.: 41    3rd Qu.: 19
##  Max.   :21     Max.   :489    Max.   :328    Max.   :284    Max.   :101
##  NA's   :2162   NA's   :2162   NA's   :2162   NA's   :2162   NA's   :2162
##    l_bpSaved      l_bpFaced     winner_rank   winner_rank_points   loser_rank
##  Min.   : 0     Min.   : 0     Min.   :   1   Min.   :    1      Min.   :   1
##  1st Qu.: 2     1st Qu.: 6     1st Qu.:  17   1st Qu.:  645      1st Qu.:  35
##  Median : 4     Median : 8     Median :  43   Median : 1009      Median :  66
##  Mean   : 5     Mean   : 9     Mean   :  76   Mean   : 1890      Mean   : 115
##  3rd Qu.: 7     3rd Qu.:11     3rd Qu.:  81   3rd Qu.: 1950      3rd Qu.: 110
##  Max.   :25     Max.   :31     Max.   :2101   Max.   :16950      Max.   :2159
##  NA's   :2162   NA's   :2162   NA's   :198    NA's   :198        NA's   :521
##  loser_rank_points
##  Min.   :    1
##  1st Qu.:  510
##  Median :  752
##  Mean   : 1074
##  3rd Qu.: 1160
##  Max.   :16950
##  NA's   :521
```

```
atp <- na.omit(atp)
```

Due to the complexity of missing records we will not impute values. Any records with NAs are omitted and this will be the dataset we use going forwards.

2.5.2 Data Transformation

In our ATP dataset the winner is always in the same column. We want to feed 2 players into our model and ask the model to predict the winner. The winner cannot come from the same column. A variable 'winner' is created to record who actually won the match so we can check our predictions. Other variables are converted into rates to further reduce the number of variables we have. Please note some of the code in this section will take a while to run because the operation is rowwise.Some of the variables are also converted into rates.

```
# Set seed
set.seed(17)

# Using dataset, replace winner & losers with Player 0 or 1 by random. Record who won in another column
# Ensure all stats assigned to players are recorded by the correct column rows
# Remove Height, which has lots of na's
atp <- atp %>%
  mutate(winner = as.factor(sample(0:1, n(), replace = T))) %>%
  mutate(name.p0 = if_else(winner == 0, winner_name, loser_name),
         name.p1 = if_else(winner == 0, loser_name, winner_name),
         age.p0 = if_else(winner == 0, winner_age, loser_age),
         age.p1 = if_else(winner == 0, loser_age, winner_age),
         hand.p0 = if_else(winner == 0, winner_hand, loser_hand),
         hand.p1 = if_else(winner == 0, loser_hand, winner_hand),
         aceRate.p0 = if_else(winner == 0, w_ace/w_svpt, l_ace/l_svpt),
         aceRate.p1 = if_else(winner == 0, l_ace/l_svpt, w_ace/w_svpt),
         dfRate.p0 = if_else(winner == 0, w_df/w_svpt, l_df/l_svpt),
         dfRate.p1 = if_else(winner == 0, l_df/l_svpt, w_df/w_svpt),
         fSrvInRate.p0 = if_else(winner == 0, w_1stIn/w_svpt, l_1stIn/l_svpt),
```

```
        fSrvInRate.p1 = if_else(winner == 0, l_1stIn/l_svpt, w_1stIn/w_svpt),
        fSrvWonRate.p0 = if_else(winner == 0, w_1stWon/w_1stIn, l_1stWon/l_1stIn),
        fSrvWonRate.p1 = if_else(winner == 0, l_1stWon/l_1stIn, w_1stWon/w_1stIn),
        SSrvWonRate.p0 = if_else(winner == 0, w_2ndWon/(w_svpt-w_1stIn), l_2ndWon/(l_svpt-l_1stIn)),
        SSrvWonRate.p1 = if_else(winner == 0, l_2ndWon/(l_svpt-l_1stIn), w_2ndWon/(w_svpt-w_1stIn)),
        bpSavedRate.p0 = if_else(winner == 0, w_bpSaved/w_bpFaced, l_bpSaved/l_bpFaced),
        bpSavedRate.p1 = if_else(winner == 0, l_bpSaved/l_bpSaved, w_bpSaved/w_bpSaved),
        rankPts.p0 = if_else(winner == 0, winner_rank_points, loser_rank_points),
        rankPts.p1 = if_else(winner == 0, loser_rank_points, winner_rank_points),
        rank.p0 = if_else(winner == 0, winner_rank, loser_rank),
        rank.p1 = if_else(winner == 0, loser_rank, winner_rank)) %>%
  select(-c(winner_name, winner_hand, winner_age,
            loser_name, loser_hand, loser_age,
            w_ace, w_df, w_svpt, w_1stIn, w_1stWon, w_2ndWon, w_bpSaved, w_bpFaced, winner_rank_points,
            l_ace, l_df, l_svpt, l_1stIn, l_1stWon, l_2ndWon, l_bpSaved, l_bpFaced, loser_rank_points,
```

The next step is to adjust the stats. Currently each record holds the specific stats for a match. We do not want to feed into our model match stats because they occur after the fact. Therefore all variables apart from rank are to be adjusted to reflect a player's prior record rather than his performance for that match.

The first values to add to our dataset is a players win rate by court type and form. All matches are gathered into a dataframe - courtWinRate - by surface. Then for each match in the atp dataset, each players win rate for that court type is calculated for the last 3 years prior to the match we are predicting. Each players form is also calculated for the last 3 months. This is done by manipulating the tourney date variable.

```
# Court type win rate
# Games, wins, win% by court type
courtWinRate <- atp %>%
  select(name.p0, name.p1, surface, tourney_date, winner) %>%
  gather(key = label, value = name, -c(surface, tourney_date, winner)) %>%
  mutate(win = str_extract(label, '\\d') == winner)

# Create new variables giving players winning % for the last 3 months and winning percentage on court s
atp <- atp %>%
  rowwise() %>%
  mutate(surfaceWinRate.p0 = sum((courtWinRate$name == name.p0) &
                                   (surface == courtWinRate$surface) &
                                   ((courtWinRate$tourney_date < tourney_date) & (courtWinRate$tourney_dat
                                   (courtWinRate$win == TRUE)) /
                                 sum((courtWinRate$name == name.p0) &
                                   (surface == courtWinRate$surface) &
                                   ((courtWinRate$tourney_date < tourney_date) & (courtWinRate$tourney_dat
         surfaceWinRate.p1 = sum((courtWinRate$name == name.p1) &
                                   (surface == courtWinRate$surface) &
                                   ((courtWinRate$tourney_date < tourney_date) & (courtWinRate$tourney_dat
                                   (courtWinRate$win == TRUE)) /
                                 sum((courtWinRate$name == name.p1) &
                                   (surface == courtWinRate$surface) &
                                   ((courtWinRate$tourney_date < tourney_date) & (courtWinRate$tourney_dat
         form.p0 = sum((courtWinRate$name == name.p0) &
                         (courtWinRate$win == TRUE) &
                         ((courtWinRate$tourney_date < tourney_date) & (courtWinRate$tourney_date + 300
                       sum((courtWinRate$name == name.p0) &
                         ((courtWinRate$tourney_date < tourney_date) & (courtWinRate$tourney_date + 300
         form.p1 = sum((courtWinRate$name == name.p1) &
```

```
                          (courtWinRate$win == TRUE) &
                          ((courtWinRate$tourney_date < tourney_date) & (courtWinRate$tourney_date + 300
                    sum((courtWinRate$name == name.p1) &
                          ((courtWinRate$tourney_date < tourney_date) & (courtWinRate$tourney_date + 300

# Check to see if values make sense
atp %>% select(tourney_date, name.p0, name.p1, surfaceWinRate.p0, surface, surfaceWinRate.p1, form.p0,
  filter(tourney_date > 20141001) %>%
  filter(name.p0 == "Roger Federer") %>%
  slice(1:20) %>%
  knitr::kable() %>%
  kable_styling(latex_options = c("striped", "scale_down"))
```

tourney_date	name.p0	name.p1	surfaceWinRate.p0	surface	surfaceWinRate.p1	form.p0	form.p1
20141005	Roger Federer	Julien Benneteau	0.832	Hard	0.550	0.842	0.525
20141005	Roger Federer	Novak Djokovic	0.832	Hard	0.881	0.842	0.875
20141005	Roger Federer	Gilles Simon	0.832	Hard	0.582	0.842	0.512
20141020	Roger Federer	Gilles Muller	0.838	Hard	0.442	0.855	0.444
20141020	Roger Federer	Denis Istomin	0.838	Hard	0.515	0.855	0.565
20141020	Roger Federer	Grigor Dimitrov	0.838	Hard	0.606	0.855	0.750
20141027	Roger Federer	Jeremy Chardy	0.843	Hard	0.494	0.866	0.537
20141109	Roger Federer	Kei Nishikori	0.829	Hard	0.721	0.857	0.814
20141109	Roger Federer	Milos Raonic	0.829	Hard	0.730	0.857	0.723
20141109	Roger Federer	Stanislas Wawrinka	0.829	Hard	0.652	0.857	0.674
20150104	Roger Federer	Grigor Dimitrov	0.824	Hard	0.606	NaN	NaN
20150104	Roger Federer	Milos Raonic	0.824	Hard	0.710	NaN	NaN
20150119	Roger Federer	Yen Hsun Lu	0.830	Hard	0.494	1.000	0.600
20150312	Roger Federer	Andreas Seppi	0.813	Hard	0.525	0.917	0.647
20150312	Roger Federer	Jack Sock	0.813	Hard	0.509	0.917	NaN
20150312	Roger Federer	Milos Raonic	0.813	Hard	0.702	0.917	0.714
20150412	Roger Federer	Jeremy Chardy	0.762	Clay	0.511	0.889	0.500
20150427	Roger Federer	Jarkko Nieminen	0.750	Clay	0.432	0.850	0.467
20150427	Roger Federer	Daniel Gimeno Traver	0.750	Clay	0.484	0.850	0.643
20150503	Roger Federer	Nick Kyrgios	0.771	Clay	0.556	0.875	0.643

The table shows Roger Federer's winrate from October 2014 to May 2015. The surface winrate adjusts as expected once Roger moves to play on clay. Roger's form also moves as expected with date.

Tennis is played in tournaments, players keep on playing if they win until the tournament is won. All players accumulate fatigue. This has been simulated by working out the total minutes played by the player in the last month.

```
# Create Fatigue Dataframe
fatigue <- atp %>%
  select(name.p0, name.p1, tourney_date, minutes) %>%
  gather(key = label, value = name, -c(tourney_date, minutes))

# Get fatigue for player from last month
atp <- atp %>%
  rowwise() %>%
  mutate(fatigue.p0 = sum(fatigue$minutes[((fatigue$name == name.p0) &
                                          ((fatigue$tourney_date < tourney_date) & (fatigue$tourney_
         fatigue.p1 = sum(fatigue$minutes[((fatigue$name == name.p1) &
                                          ((fatigue$tourney_date < tourney_date) & (fatigue$tourney_

# Check data makes sense
atp %>% select(tourney_date, name.p0, name.p1, fatigue.p0, fatigue.p1, minutes) %>%
```

```
  filter(name.p0 == "Andy Murray" | name.p1 == "Andy Murray") %>%
  filter(tourney_date > 20141001) %>%
  slice(1:20) %>%
  knitr::kable()
```

tourney_date	name.p0	name.p1	fatigue.p0	fatigue.p1	minutes
20141005	Andy Murray	Teymuraz Gabashvili	837	209	85
20141005	Jerzy Janowicz	Andy Murray	404	837	89
20141005	David Ferrer	Andy Murray	194	837	115
20141013	Vasek Pospisil	Andy Murray	610	1126	99
20141013	Andy Murray	Jan Lennard Struff	1126	351	86
20141013	Andy Murray	Viktor Troicki	1126	429	83
20141013	David Ferrer	Andy Murray	552	1126	161
20141020	Andy Murray	Jurgen Melzer	1555	468	80
20141020	Andy Murray	Fabio Fognini	1555	240	72
20141020	Andy Murray	Kevin Anderson	1555	447	164
20141020	Andy Murray	David Ferrer	1555	1034	119
20141020	Andy Murray	Tommy Robredo	1555	1109	200
20141027	Julien Benneteau	Andy Murray	421	1776	72
20141027	Grigor Dimitrov	Andy Murray	1127	1776	70
20141027	Novak Djokovic	Andy Murray	809	1776	101
20141109	Andy Murray	Roger Federer	1307	718	56
20141109	Kei Nishikori	Andy Murray	472	1307	95
20141109	Milos Raonic	Andy Murray	1008	1307	92
20150119	Yuki Bhambri	Andy Murray	0	0	133
20150119	Marinko Matosevic	Andy Murray	191	0	102

Looking at Andy Murray's fatigue we can see that the value adjusts as we progress in time. One interesting fact about the dataset is discovered here; because the dataset does not contain the date of the match just the date of the tournament, player values are only updated for the next tournament a player plays and not the next match.

A player's head to head record with another player is another variable to investigate. This is done below by finding all prior matches where both players have played each other and constructing a ledger. The value of the ledger at that point in time in then added as a variable for the match.

```
# Head to Head Balance
h2h <- atp %>%
  select(name.p0, name.p1, winner, tourney_date) %>%
  mutate(win = str_extract("name.p0", '\\d') == winner)

# For every match find culmulative head to head to that date
# Negative values give p1 the advantage
atp <- atp %>% rowwise() %>%
  mutate(h2h = 2*(sum((((h2h$name.p0 == name.p0) & (h2h$name.p1 == name.p1)) &
                        (h2h$tourney_date < tourney_date) &
                        (h2h$winner == 0)) |
                        (((h2h$name.p0 == name.p1) & (h2h$name.p1 == name.p0)) &
                        (h2h$tourney_date < tourney_date) &
                        (h2h$winner == 1)))) -
                sum((((h2h$name.p0 == name.p0) & (h2h$name.p1 == name.p1)) |
                     ((h2h$name.p0 == name.p1) & (h2h$name.p1 == name.p0))) &
                      (h2h$tourney_date < tourney_date)))

# Check a h2h matchup to ensure accuracy of function
```

```
atp %>% select(tourney_date, name.p0, name.p1, winner, h2h) %>%
  filter(((name.p0 == "Roger Federer") & (name.p1 == "Rafael Nadal")) |
         ((name.p1 == "Roger Federer") & (name.p0 == "Rafael Nadal"))) %>%
  slice(1:20) %>%
  knitr::kable() %>%
  kable_styling(latex_options = c("striped", "scale_down"))
```

tourney_date	name.p0	name.p1	winner	h2h
20100509	Roger Federer	Rafael Nadal	1	0
20101121	Rafael Nadal	Roger Federer	1	1
20110323	Rafael Nadal	Roger Federer	0	0
20110501	Roger Federer	Rafael Nadal	1	-1
20110522	Rafael Nadal	Roger Federer	0	2
20111120	Rafael Nadal	Roger Federer	1	3
20120116	Rafael Nadal	Roger Federer	0	2
20120308	Rafael Nadal	Roger Federer	1	3
20130307	Rafael Nadal	Roger Federer	0	2
20130512	Rafael Nadal	Roger Federer	0	3
20130811	Roger Federer	Rafael Nadal	1	-4
20131104	Rafael Nadal	Roger Federer	0	5
20140113	Roger Federer	Rafael Nadal	1	-6
20170116	Roger Federer	Rafael Nadal	0	-7
20170306	Rafael Nadal	Roger Federer	1	6
20170320	Roger Federer	Rafael Nadal	0	-5
20171009	Rafael Nadal	Roger Federer	1	4
20190527	Roger Federer	Rafael Nadal	1	-3
20190701	Rafael Nadal	Roger Federer	1	4

The table shows the history of how the head to head record of Roger Federer and Rafael Nadal has progressed from 2010 to 2019. A positive h2h value is given if player 0 has the edge, while a negative value is given if player 1 is ahead. The table also allows us to check that the operation to get the head to head value is correct.

Match stats also need to be adjusted to reflect prior performance to the match for each player, not the player's stats for that particular match. A player's performance over the last 2 years for the different statistics is assessed below. Any Nan values were dividing by 0 and are from when there is no record rather than an Na value, these are replaced with an actual 0.

```
# Stats Record
stats <- atp %>%
  select(name.p0, name.p1, tourney_date, aceRate.p0, aceRate.p1, dfRate.p0, dfRate.p1,
```

```
        fSrvInRate.p0, fSrvInRate.p1, fSrvWonRate.p0, fSrvWonRate.p1,
        SSrvWonRate.p0, SSrvWonRate.p1, bpSavedRate.p0, bpSavedRate.p1) %>%
  gather(key = label, value = name, c(name.p0, name.p1)) %>%
  mutate(aceRate = if_else(str_extract(label, '\\d') == 0, aceRate.p0, aceRate.p1),
         dfRate = if_else(str_extract(label, '\\d') == 0, dfRate.p0, dfRate.p1),
         fSrvInRate = if_else(str_extract(label, '\\d') == 0, fSrvInRate.p0, fSrvInRate.p1),
         fSrvWonRate = if_else(str_extract(label, '\\d') == 0, fSrvWonRate.p0, fSrvWonRate.p1),
         SSrvWonRate = if_else(str_extract(label, '\\d') == 0, SSrvWonRate.p0, SSrvWonRate.p1),
         bpSavedRate = if_else(str_extract(label, '\\d') == 0, bpSavedRate.p0, bpSavedRate.p1)) %>%
  select(-c(aceRate.p0, aceRate.p1, dfRate.p0, dfRate.p1, fSrvInRate.p0, fSrvInRate.p1,
            fSrvWonRate.p0, fSrvWonRate.p1, SSrvWonRate.p0, SSrvWonRate.p1, bpSavedRate.p0, bpSavedRate
            label))

# Na's are from dividing by 0, replace with 0
stats[is.na(stats)] <- 0

# Add stats for the players from the last two year
atp <- atp %>% rowwise() %>%
  mutate(aceRate.p0 = sum(stats$aceRate[((stats$name == name.p0) &
                                          ((stats$tourney_date < tourney_date) & (stats$tourney_date +
                         sum((stats$name == name.p0) &
                               ((stats$tourney_date < tourney_date) & (stats$tourney_date + 20000 > tourn
         aceRate.p1 = sum(stats$aceRate[((stats$name == name.p1) &
                                          ((stats$tourney_date < tourney_date) & (stats$tourney_date +
                         sum((stats$name == name.p1) &
                               ((stats$tourney_date < tourney_date) & (stats$tourney_date + 20000 > tourn
         dfRate.p0 = sum(stats$dfRate[((stats$name == name.p0) &
                                          ((stats$tourney_date < tourney_date) & (stats$tourney_date +
                         sum((stats$name == name.p0) &
                               ((stats$tourney_date < tourney_date) & (stats$tourney_date + 20000 > tourn
         dfRate.p1 = sum(stats$dfRate[((stats$name == name.p1) &
                                          ((stats$tourney_date < tourney_date) & (stats$tourney_date +
                         sum((stats$name == name.p1) &
                               ((stats$tourney_date < tourney_date) & (stats$tourney_date + 20000 > tourn
         fSrvInRate.p0 = sum(stats$fSrvInRate[((stats$name == name.p0) &
                                         ((stats$tourney_date < tourney_date) & (stats$tourney_date + 2
                         sum((stats$name == name.p0) &
                               ((stats$tourney_date < tourney_date) & (stats$tourney_date + 20000 > tourn
         fSrvInRate.p1 = sum(stats$fSrvInRate[((stats$name == name.p1) &
                                         ((stats$tourney_date < tourney_date) & (stats$tourney_date + 2
                         sum((stats$name == name.p1) &
                               ((stats$tourney_date < tourney_date) & (stats$tourney_date + 20000 > tourn
         fSrvWonRate.p0 = sum(stats$fSrvWonRate[((stats$name == name.p0) &
                                           ((stats$tourney_date < tourney_date) & (stats$tourney_
                         sum((stats$name == name.p0) &
                               ((stats$tourney_date < tourney_date) & (stats$tourney_date + 20000 > tourn
         fSrvWonRate.p1 = sum(stats$fSrvWonRate[((stats$name == name.p1) &
                                           ((stats$tourney_date < tourney_date) & (stats$tourney_
                         sum((stats$name == name.p1) &
                               ((stats$tourney_date < tourney_date) & (stats$tourney_date + 20000 > tourn
         SSrvWonRate.p0 = sum(stats$SSrvWonRate[((stats$name == name.p0) &
                                             ((stats$tourney_date < tourney_date) & (stats$tourney
                         sum((stats$name == name.p0) &
```

```
                                ((stats$tourney_date < tourney_date) & (stats$tourney_date + 20000 > tourn
        SSrvWonRate.p1 = sum(stats$SSrvWonRate[((stats$name == name.p1) &
                                                ((stats$tourney_date < tourney_date) & (stats$tourney
                             sum((stats$name == name.p1) &
                                ((stats$tourney_date < tourney_date) & (stats$tourney_date + 20000 > tourn
        bpSavedRate.p0 = sum(stats$bpSavedRate[((stats$name == name.p0) &
                                                ((stats$tourney_date < tourney_date) & (stats$tourney
                             sum((stats$name == name.p0) &
                                ((stats$tourney_date < tourney_date) & (stats$tourney_date + 20000 > tourn
        bpSavedRate.p1 = sum(stats$bpSavedRate[((stats$name == name.p1) &
                                                ((stats$tourney_date < tourney_date) & (stats$tourney
                             sum((stats$name == name.p1) &
                                ((stats$tourney_date < tourney_date) & (stats$tourney_date + 20000 > tourn

# Check stats to ensure accuracy of function
atp %>% select(tourney_date, name.p0, aceRate.p0, dfRate.p0, fSrvInRate.p0, fSrvWonRate.p0, SSrvWonRate
  filter(name.p0 == "Roger Federer") %>%
  filter(tourney_date > 20110000) %>%
  slice(1:20) %>%
  knitr::kable() %>%
  kable_styling(latex_options = c("striped", "scale_down"))
```

tourney_date	name.p0	aceRate.p0	dfRate.p0	fSrvInRate.p0	fSrvWonRate.p0	SSrvWonRate.p0	bpSavedRate.p0
20110103	Roger Federer	0.117	0.019	0.620	0.796	0.571	0.639
20110103	Roger Federer	0.117	0.019	0.620	0.796	0.571	0.639
20110117	Roger Federer	0.115	0.018	0.623	0.796	0.579	0.612
20110221	Roger Federer	0.113	0.018	0.625	0.794	0.579	0.627
20110221	Roger Federer	0.113	0.018	0.625	0.794	0.579	0.627
20110221	Roger Federer	0.113	0.018	0.625	0.794	0.579	0.627
20110221	Roger Federer	0.113	0.018	0.625	0.794	0.579	0.627
20110310	Roger Federer	0.112	0.019	0.625	0.792	0.575	0.642
20110310	Roger Federer	0.112	0.019	0.625	0.792	0.575	0.642
20110323	Roger Federer	0.109	0.019	0.623	0.793	0.574	0.647
20110323	Roger Federer	0.109	0.019	0.623	0.793	0.574	0.647
20110323	Roger Federer	0.109	0.019	0.623	0.793	0.574	0.647
20110323	Roger Federer	0.109	0.019	0.623	0.793	0.574	0.647
20110410	Roger Federer	0.108	0.019	0.622	0.792	0.581	0.640
20110410	Roger Federer	0.108	0.019	0.622	0.792	0.581	0.640
20110501	Roger Federer	0.107	0.019	0.624	0.791	0.581	0.650
20110501	Roger Federer	0.107	0.019	0.624	0.791	0.581	0.650
20110508	Roger Federer	0.106	0.019	0.625	0.789	0.581	0.654
20110508	Roger Federer	0.106	0.019	0.625	0.789	0.581	0.654
20110522	Roger Federer	0.105	0.019	0.625	0.789	0.581	0.655

The numbers in the tables show updates as Roger progresses through the tournaments in 2011, they are varying as intended.

We now have created some features for our model as well as updated all the stats so we are not predicting on match stats by a player prior performance. The dataframes not required are removed below. All NAs are also converted to 0, all NA's are as a result of dividing by 0 when we calculated rates for when a player has no data. Variables such as tournament ID / best of / hand are removed because I have assessed them to be of limited predictive value. Other variables such as surface are taken into account by the surface win rate of the players. The summary below shows the final data set to be used for predicting matches.

```
# Remove unneeded dataframes
rm(courtWinRate, fatigue, h2h, stats)
# Convert Na's to 0
atp[is.na(atp)] <- 0
```

```
# Remove variables not to be used as predictors
atp <- atp %>% select(-c(tourney_id, tourney_name, tourney_level, best_of, minutes,
                         round, name.p0, name.p1, hand.p0, hand.p1, surface))
```

2.5.3 Training / Testing / Validation Set

In keeping with the terminology of the movielens project. We will create a validation set to test our final model on. Train a range of models on a seperate training set while testing them again on a separate testing test. Because our data is in time series, it does not make sense to randomly split out matches. All matches before 2018 will be used as the training set, the matches in 2018 will be the testing set, all matches in 2019 will be the validation set.

```
# Use 2019 for validation set
validation <- atp %>%
  filter(tourney_date > 20190000)

# Use 2018 for test set
testing <- atp %>%
  filter(tourney_date > 20180000 & tourney_date < 20190000)

# Any matches before 2018 is inital training data
training <- atp %>%
  filter(tourney_date < 20180000)
```

2.6 Accuracy Metric

The accuracy metric to be used will be number of correct predictions divided by the total amount of matches predicted on.

2.7 Modelling Methods

A number of models were trialed during the course of this project. The idea for the project spawned from this website: https://www.ultimatetennisstatistics.com/inProgressEventsForecasts# using a neural network to predict the outcome of matches. However I was never able to get a neural network to converge. So neural networks and advanced algorithms will be discussed in the context of future works in the conclusion from the observations garnered from modelling.

2.7.1 Baseline Model

The simplest model would be to have our model predict that the favourite for the match will win. In the baseline model we will predict that the winner of the match is the player with the better ranking at that point in time.

```
# Model 1 - Baseline; if a player is has a lower rank predict win
baselineModel <- testing %>%
  select(winner, rank.p0, rank.p1) %>%
  filter(!is.na(rank.p0) & !is.na(rank.p1)) %>%
  mutate(baselinePred = as.integer(ifelse(rank.p0 < rank.p1, 0, 1))) %>%
  mutate(correct = ifelse(baselinePred == winner, 1, 0))

# Table of correct predictions
table(baselineModel$correct)
```

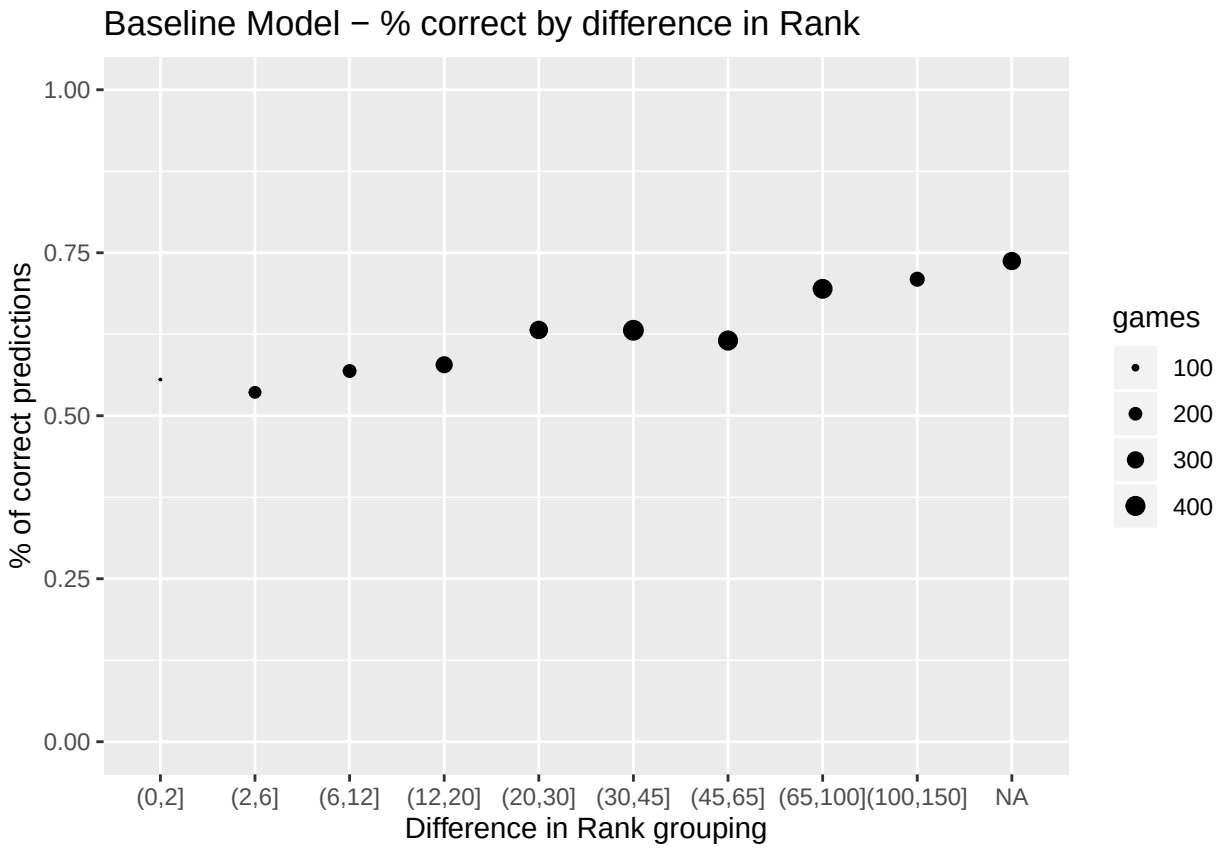
```
##
##    0    1
## 1048 1850
```

```
# Create dataframe to capture all model accuracies
accuracy <- data_frame(Method = "baseline", Accuracy = mean(baselineModel$correct))
# Present table
accuracy %>% knitr::kable()
```

Method	Accuracy
baseline	0.638

Using the approaching of the palyer with the higher ranking wins, our accuracy is 0.638. To explore this further the following graphs were developed:

```
# Graph showing %correct by difference in rank
baselineModel %>%
  mutate(rankdiff = abs(rank.p0 - rank.p1)) %>%
  group_by(grp = cut(rankdiff, breaks = c(0,2,6,12,20,30,45,65,100,150))) %>%
  summarise(perCorrect = sum(correct)/n(),
            games = n()) %>%
  ggplot(aes(grp, perCorrect)) +
  geom_point(aes(size = games)) +
  scale_size(range = c(0,3)) +
  scale_y_continuous(limits = c(0,1)) +
  labs(title = "Baseline Model - % correct by difference in Rank",
       x = "Difference in Rank grouping",
       y = "% of correct predictions")
```

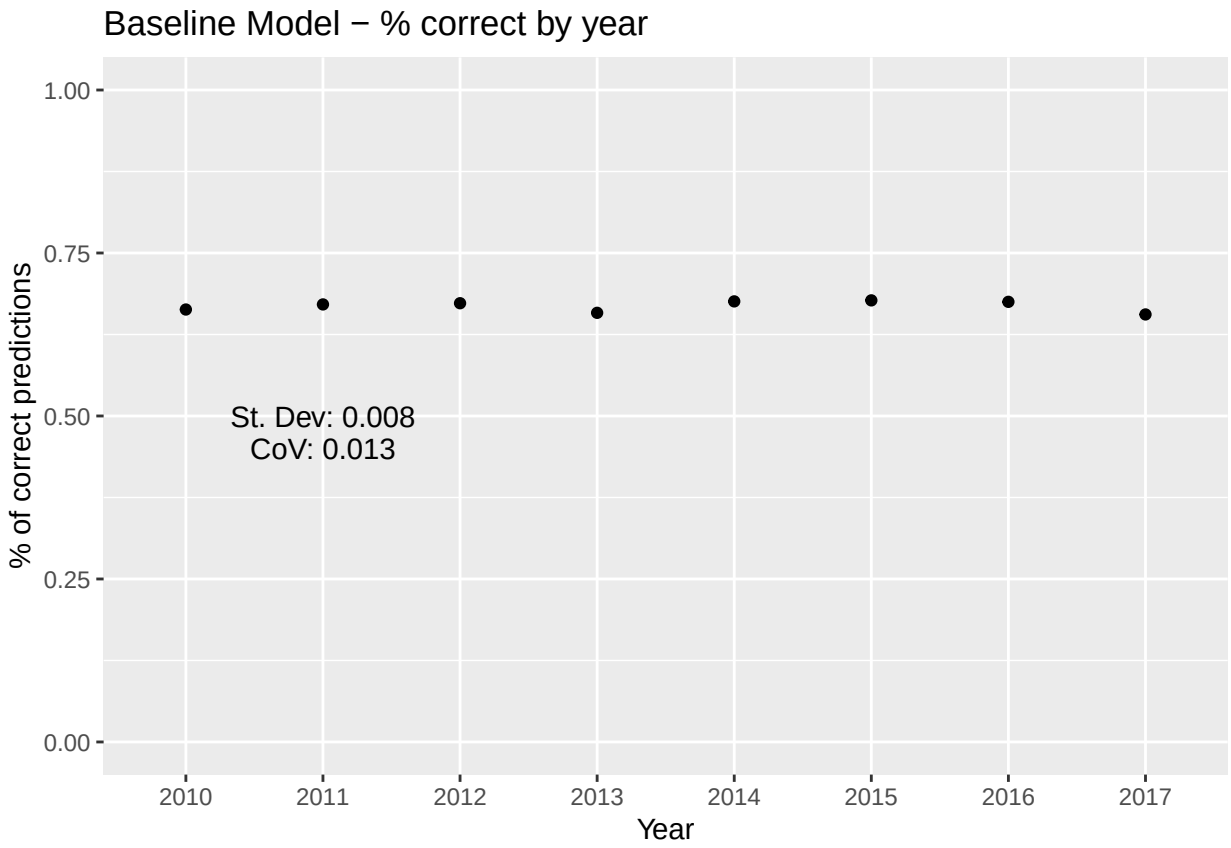



```
# Look at accuracy of training data using baseline model
# Split by year
dates <- seq(20110000, 20180000, 10000)
```

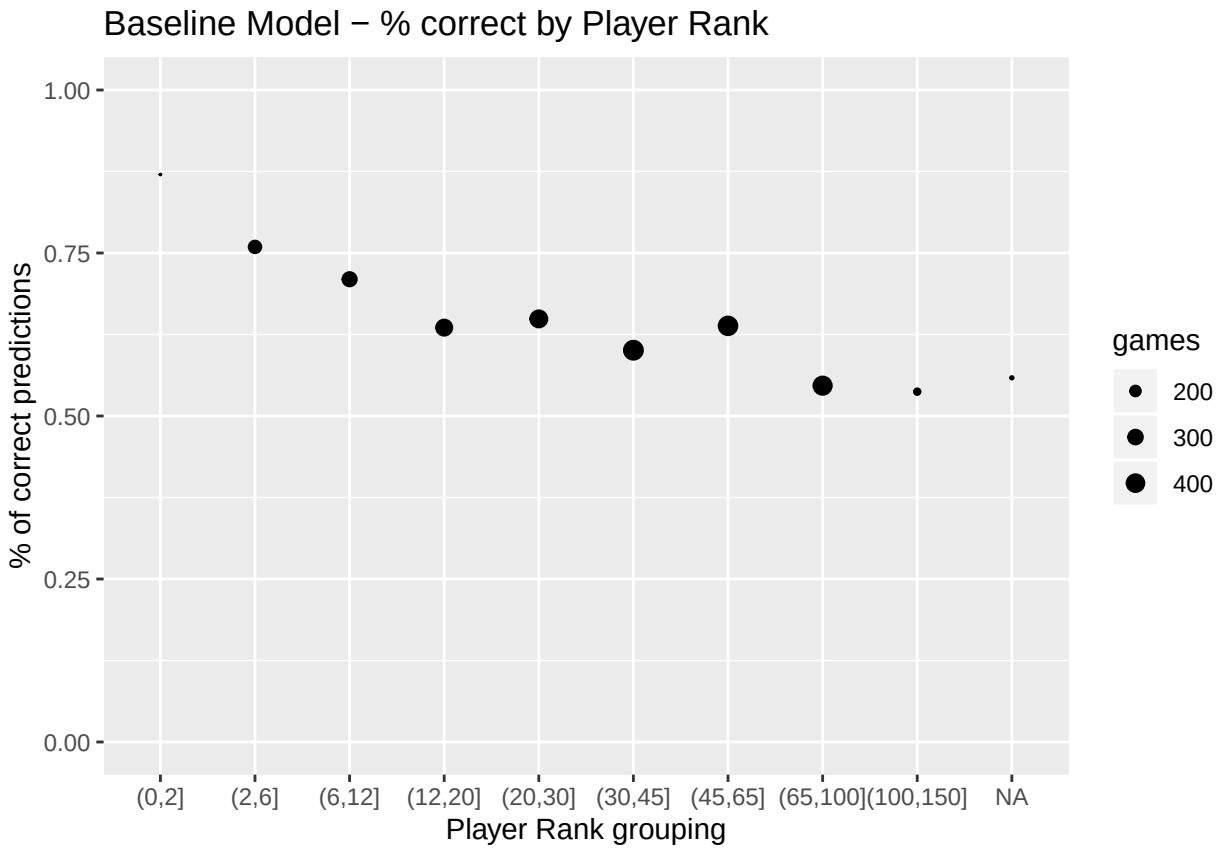
```r
baselineWinsByYear <- pbsapply(dates, function(d){

  yearWins <- training %>%
    filter(tourney_date > (d-10000) & tourney_date < d) %>%
    select(winner, rank.p0, rank.p1) %>%
    filter(!is.na(rank.p0) & !is.na(rank.p1)) %>%
    mutate(baselinePred = as.integer(ifelse(rank.p0 < rank.p1, 0, 1))) %>%
    mutate(correct = ifelse(baselinePred == winner, 1, 0))

  return(mean(yearWins$correct))

})

# Graph showing rate of baseline correct by year
baselineWinsByYear <- as.data.frame(baselineWinsByYear)
rownames(baselineWinsByYear) <- seq(2010,2017,1)
colnames(baselineWinsByYear) <- "correct"
knitr::kable(baselineWinsByYear)
```

	correct
2010	0.663
2011	0.671
2012	0.673
2013	0.658
2014	0.676
2015	0.677
2016	0.675
2017	0.656

```r
baselineWinsByYear %>% ggplot(aes(x = rownames(baselineWinsByYear), y = correct)) +
  geom_point() +
  scale_y_continuous(limits = c(0,1)) +
  annotate("text", x = 2, y = c(0.5, 0.45),
           label = c(paste0("St. Dev: ", round(sd(baselineWinsByYear$correct),3)),
                     paste0("CoV: ", round(cv(baselineWinsByYear$correct), 3)))) +
  labs(title = "Baseline Model - % correct by year",
       x = "Year",
       y = "% of correct predictions")
```




```
# Between 2010 & 2018, difference in rank gives a more accurate prediction of winner
baselineModel %>%
  mutate(minRank = pmin(rank.p0, rank.p1)) %>%
  group_by(grp = cut(minRank, breaks = c(0,2,6,12,20,30,45,65,100,150))) %>%
  summarise(perCorrect = sum(correct)/n(),
            games = n()) %>%
  ggplot(aes(grp, perCorrect)) +
  geom_point(aes(size = games)) +
  scale_size(range = c(0,3)) +
  scale_y_continuous(limits = c(0,1)) +
  labs(title = "Baseline Model - % correct by Player Rank",
       x = "Player Rank grouping",
       y = "% of correct predictions")
```



The first graph shows that as the difference in rank between two players increases, predicting the player with a higher rating will win is more accurate. This makes sense as the ranking difference should provide more certainty over who is more likely to win. The next graph and the table shows the accuracy of using the baseline method over the years in our training dataset. There is not much variability, favourites win about 67% of the time. The final graph shows that the ranking of the favouite player at the point in time the match is played is important in how accurate the baseline method is. The higher the favourite player is ranked, the more likely he is to actually win. Anytime a favourite is ranked greater than 65, their win rate drops to just over 50%; while players ranked 6 or below have a 75% win rate when they are the higher ranked opponent.

2.7.2 Logistic Regression Model

Logistic regression models the probability of a certain event occuring, ie. which player will win. Logistic regession uses the matural logarithm function to find the relationship between variables and uses test data to find co-efficients. These co-efficients are then used to predict future results.

```
# Model 2 - Logistic Regression
# Train model
logitModel <- caret::train(winner ~ . -tourney_date,
                    data = training,
                    method = 'glm',
                    family = "binomial",
                    na.action = NULL)

summary(logitModel)
```

```
##
## Call:
## NULL
```

```
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.859  -1.009  -0.217   1.010   2.907
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)       -1.44e-01   2.09e-01   -0.69  0.49207
## age.p0             2.17e-02   4.32e-03    5.03  5.0e-07 ***
## age.p1            -1.26e-02   4.28e-03   -2.95  0.00319 **
## aceRate.p0        -1.82e+00   5.76e-01   -3.16  0.00155 **
## aceRate.p1         1.34e+00   5.75e-01    2.33  0.01959 *
## dfRate.p0          1.91e+00   1.51e+00    1.26  0.20630
## dfRate.p1         -2.49e+00   1.53e+00   -1.62  0.10441
## fSrvInRate.p0      1.27e-01   3.35e-01    0.38  0.70447
## fSrvInRate.p1     -6.36e-01   3.34e-01   -1.91  0.05666 .
## fSrvWonRate.p0    -1.52e-01   5.00e-01   -0.30  0.76186
## fSrvWonRate.p1     5.38e-01   5.06e-01    1.06  0.28788
## SSrvWonRate.p0    -5.99e-01   5.91e-01   -1.01  0.31089
## SSrvWonRate.p1     7.70e-01   5.95e-01    1.29  0.19598
## bpSavedRate.p0     3.42e-01   2.03e-01    1.69  0.09127 .
## bpSavedRate.p1    -4.35e-01   2.01e-01   -2.16  0.03049 *
## rankPts.p0        -1.99e-04   1.34e-05  -14.91  < 2e-16 ***
## rankPts.p1         2.12e-04   1.39e-05   15.27  < 2e-16 ***
## rank.p0            1.48e-03   2.09e-04    7.07  1.6e-12 ***
## rank.p1           -1.73e-03   2.21e-04   -7.86  3.8e-15 ***
## surfaceWinRate.p0 -1.32e+00   1.08e-01  -12.27  < 2e-16 ***
## surfaceWinRate.p1  1.28e+00   1.07e-01   11.99  < 2e-16 ***
## form.p0           -4.01e-01   9.44e-02   -4.25  2.2e-05 ***
## form.p1            3.20e-01   9.65e-02    3.31  0.00092 ***
## fatigue.p0        -3.41e-04   6.57e-05   -5.20  2.0e-07 ***
## fatigue.p1         3.50e-04   6.58e-05    5.32  1.0e-07 ***
## h2h               -5.40e-02   1.42e-02   -3.80  0.00014 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 27976  on 20180  degrees of freedom
## Residual deviance: 23768  on 20155  degrees of freedom
## AIC: 23820
##
## Number of Fisher Scoring iterations: 4
```

```r
# All significant variables are match stats

# Predict on testing set
logitPred <- predict(logitModel, newdata = testing)

# Look at predicted data
summary(logitPred)
```

```
##    0    1
## 1468 1430
```

```r
# Compare values
logitCorrect <- (logitPred == testing$winner)
table(logitCorrect)
```

```
## logitCorrect
## FALSE  TRUE
##   976  1922
```

```r
# Work out accuracy
accuracy <- bind_rows(accuracy,
                      data_frame(Method = "Logistic Regression",
                                 Accuracy = sum(logitCorrect) / length(logitCorrect)))
accuracy %>% knitr::kable()
```

Method	Accuracy
baseline	0.638
Logistic Regression	0.663

```r
table(logitPred == testing$winner, testing$winner)
```

```
##
##           0   1
##   FALSE 456 520
##   TRUE  948 974
```

Logistics regression improves our accuracy. In the summary of the model, many of the serving variables generated are listed as not statistically significant. We will prune these to see if removing the variables will improve model accuracy and limit overfitting.

2.7.2.1 Pruned Logistic Regression Model

This logistic regression model prunes the variables the orginal logistic regression model does not deem statistically significant.

```r
# Model 2A - Logistic Regression remove more variables
# Train model
logitModel2B <- caret::train(winner ~ . -tourney_date - dfRate.p0 - dfRate.p1
                             - fSrvInRate.p0 - fSrvInRate.p1 - fSrvWonRate.p0
                             - fSrvWonRate.p1 - SSrvWonRate.p0 - SSrvWonRate.p1
                             - bpSavedRate.p0 - bpSavedRate.p1,
                           data = training,
                           method = 'glm',
                           na.action = NULL)

summary(logitModel2B)
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.900  -1.012  -0.218   1.013   2.956
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)       -2.00e-01   1.77e-01   -1.13  0.25829
```

```
## age.p0             2.10e-02   4.29e-03    4.90  9.5e-07 ***
## age.p1            -1.24e-02   4.25e-03   -2.91  0.00359 **
## aceRate.p0        -2.03e+00   4.05e-01   -5.01  5.4e-07 ***
## aceRate.p1         1.89e+00   4.06e-01    4.65  3.3e-06 ***
## rankPts.p0        -2.05e-04   1.30e-05  -15.70  < 2e-16 ***
## rankPts.p1         2.22e-04   1.36e-05   16.30  < 2e-16 ***
## rank.p0            1.47e-03   1.99e-04    7.36  1.8e-13 ***
## rank.p1           -1.71e-03   2.13e-04   -8.03  9.6e-16 ***
## surfaceWinRate.p0 -1.37e+00   1.04e-01  -13.24  < 2e-16 ***
## surfaceWinRate.p1  1.33e+00   1.03e-01   12.93  < 2e-16 ***
## form.p0           -4.34e-01   9.28e-02   -4.68  2.9e-06 ***
## form.p1            3.63e-01   9.47e-02    3.83  0.00013 ***
## fatigue.p0        -3.38e-04   6.55e-05   -5.16  2.5e-07 ***
## fatigue.p1         3.44e-04   6.57e-05    5.24  1.6e-07 ***
## h2h               -5.51e-02   1.42e-02   -3.88  0.00011 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 27976  on 20180  degrees of freedom
## Residual deviance: 23789  on 20165  degrees of freedom
## AIC: 23821
##
## Number of Fisher Scoring iterations: 4
```

```
# Predict on testing set
logitPred2B <- predict(logitModel2B, newdata = testing)

summary(logitPred2B)
```

```
##    0    1
## 1480 1418
```

```
# Compare values
logitCorrect2B <- (logitPred2B == testing$winner)
table(logitCorrect2B)
```

```
## logitCorrect2B
## FALSE  TRUE
##   978  1920
```

```
# Work out accuracy
accuracy <- bind_rows(accuracy,
                      data_frame(Method = "Logistic Regression no serve Stats",
                                 Accuracy = sum(logitCorrect2B) / length(logitCorrect2B)))
accuracy %>% knitr::kable()
```

Method	Accuracy
baseline	0.638
Logistic Regression	0.663
Logistic Regression no serve Stats	0.663

Removing the serving stats did not change the performance of our model, we will proceed with further models using less variables.

Investigating further the following code explores the sensitivity and specificity of our model:

```
# Get number of correct upset predictions
nrow(testing[logitPred2B == testing$winner &
          ((testing$winner == 0 & testing$rank.p0 > testing$rank.p1) | (testing$winner == 1 & testing$ra
```

```
## [1] 239
```

```
# Get number of incorrect upset predictions
nrow(testing[logitPred2B != testing$winner &
                  ((testing$winner == 0 & testing$rank.p0 < testing$rank.p1) | (testing$winner == 1 & te
```

```
## [1] 169
```

```
# Get number of correct favourite predictions
nrow(testing[logitPred2B == testing$winner &
                  ((testing$winner == 0 & testing$rank.p0 < testing$rank.p1) | (testing$winner == 1 & te
```

```
## [1] 1681
```

```
# Get number of incorrect favourite predictions
nrow(testing[logitPred2B != testing$winner &
                        ((testing$winner == 0 & testing$rank.p0 > testing$rank.p1) | (testing$winner ==
```

```
## [1] 809
```

When our model predicts that a favourite will win a match. It gets got it correct 1681 times and wrong 801 times for an accuracy at predicting favourites of 67.7%. The logistic regression model is better at picking favourites than automatically picking the favourite. This model also predicts a player with a lower rank will win on 411 times from our testing set. It is correct 239 times and wrong 169 times for a record of 58.6%. This seems a much better model than our baseline model.

Investigating where the model made errors is done below:

```
testing[logitPred2B != testing$winner &
                  ((testing$winner == 0 & testing$rank.p0 < testing$rank.p1) | (testing$winner == 1 & te
  select(-c(tourney_date, dfRate.p0, dfRate.p1, fSrvInRate.p0, fSrvInRate.p1, rankPts.p0, rankPts.p1,
            fSrvWonRate.p0, fSrvWonRate.p1, SSrvWonRate.p0, SSrvWonRate.p1, bpSavedRate.p0, bpSavedRate
  slice(100:110) %>%
  knitr::kable() %>%
  kable_styling(latex_options = c("striped","scale_down"))
```

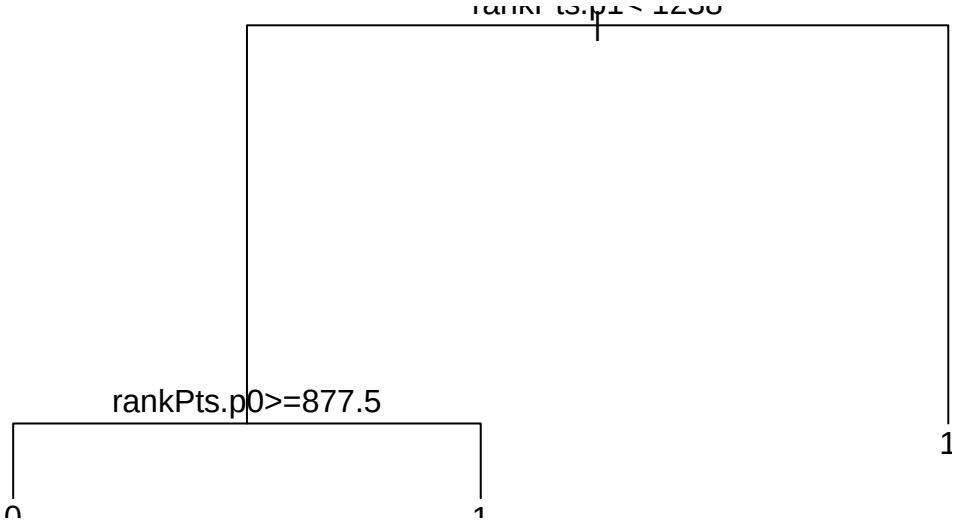
winner	age.p0	age.p1	aceRate.p0	aceRate.p1	rank.p0	rank.p1	surfaceWinRate.p0	surfaceWinRate.p1	form.p0	form.p1	fatigue.p0	fatigue.p1	h2h
1	23.1	36.5	0.044	0.051	90	86	0.000	0.125	0.308	0.267	0	70	3
1	31.4	19.2	0.109	0.095	46	25	0.556	0.286	0.613	0.543	871	421	0
1	26.4	33.5	0.037	0.040	74	53	0.364	0.333	0.414	0.576	472	275	0
1	27.0	25.9	0.080	0.035	78	11	0.333	0.000	0.522	0.611	300	135	0
0	28.2	39.3	0.103	0.234	64	112	0.200	0.600	0.407	0.429	185	113	0
0	34.7	35.1	0.065	0.183	27	60	0.588	0.714	0.500	0.318	320	401	2
1	33.2	32.1	0.236	0.184	10	8	0.688	0.500	0.593	0.697	0	144	5
1	29.0	28.1	0.051	0.102	232	98	0.625	0.286	0.111	0.222	0	129	1
0	30.5	25.0	0.057	0.065	123	133	0.000	0.600	0.222	0.286	65	0	0
0	29.1	36.5	0.051	0.075	131	134	0.000	0.600	0.000	0.375	128	0	0
0	28.2	36.6	0.063	0.051	83	88	0.200	0.525	0.400	0.294	232	423	0

Looking at random sample of incorrect upset predictions, upsets are predicted when a particular player has done better on a surface than the favourite. This stat could be misleading because even if we have taken the last 3 years worth of data, one play may have played significantly less games or won against easier opponents on that surface. Age is also a factor in predicting upsets. All this points to is potential scope to improve our model if we improve the maths of our feature extraction / transformation.

2.7.3 CART Model

A CART model is decision tree learning. Tree models are where the target variable can take a discrete set of values and outputs a classification tree. In the tree structure the leaves represent labels and instuctions for every prediction. The following code is a CART model created using our training data. The variables that the logistic regression model determined to be unimportant have been removed.

```
# Model 3 - CART Model
# Train model
cartModel <- caret::train(winner ~ . -tourney_date - dfRate.p0 - dfRate.p1
                              - fSrvInRate.p0 - fSrvInRate.p1 - fSrvWonRate.p0
                              - fSrvWonRate.p1 - SSrvWonRate.p0 - SSrvWonRate.p1
                              - bpSavedRate.p0 - bpSavedRate.p1,
                    data = training,
                    method = 'rpart',
                    cp = 0.9,
                    minsplit = 5,
                    minbucket = 6)

plot(cartModel$finalModel)
text(cartModel$finalModel)
```




```
# Predict on testing set
cartPred <- predict(cartModel, newdata = testing)

summary(cartPred)
```

```
##    0    1
##  907 1991
```

```
# Compare values
cartCorrect <- (cartPred == testing$winner)
table(cartCorrect)
```

```
## cartCorrect
## FALSE  TRUE
##  1189  1709
```

```
# Work out accuracy
accuracy <- bind_rows(accuracy,
```

```
                      data_frame(Method = "CART Model",
                                 Accuracy = sum(cartCorrect) / length(cartCorrect)))
accuracy %>% knitr::kable()
```

Method	Accuracy
baseline	0.638
Logistic Regression	0.663
Logistic Regression no serve Stats	0.663
CART Model	0.590

The accuracy of the CART model is very low. Below that of the baseline model. Fiddling with the parameters may improve the model, but this approach does not seem to be a good option. The only variables the model uses is ranking points and surface win rate.

2.7.4 Random Forest Model

Random forest is an ensemble learning method. Unlike a CART model where one decision tree is built, a random forest model builds many decision trees, hence the name forest. The idea is a large number of uncorrelated trees will outperform any individual tree. The low correlation is the key. The following is our implementation of a random forest model on our dataset. Again the variables deemed not important by the logistics regression model have been removed. Please note training this model takes a bit of time.

```
# Model 4 - Random Forest Model
# Train model
trainctrl <- trainControl(verboseIter = FALSE)
rfModel <- caret::train(winner ~ . -tourney_date - dfRate.p0 - dfRate.p1
                            - fSrvInRate.p0 - fSrvInRate.p1 - fSrvWonRate.p0
                            - fSrvWonRate.p1 - SSrvWonRate.p0 - SSrvWonRate.p1
                            - bpSavedRate.p0 - bpSavedRate.p1,
                  data = training,
                  method = 'Rborist',
                  trControl = trainctrl)

# Predict on testing set
rfPred <- predict(rfModel, newdata = testing)

summary(rfPred)
```

```
##    0    1
## 1656 1242
```

```
# Compare values
rfCorrect <- (rfPred == testing$winner)
table(rfCorrect)
```

```
## rfCorrect
## FALSE  TRUE
##  1030  1868
```

```
# Work out accuracy
accuracy <- bind_rows(accuracy,
                      data_frame(Method = "Random Forest Model",
                                 Accuracy = sum(rfCorrect) / length(rfCorrect)))
accuracy %>% knitr::kable()
```

Method	Accuracy
baseline	0.638
Logistic Regression	0.663
Logistic Regression no serve Stats	0.663
CART Model	0.590
Random Forest Model	0.645

The accuracy table shows that the performance of the Random Forest model is significantly better than a single decision tree. With better parameter tuning a random forest is a viable option to explore further. The problem with a random forest model is that it is very difficult to explore further. We cannot go through every committee tree. To use random forest we should be sure that our variables are well thought out beforehand.

2.7.5 Ensemble Models

An ensemble model takes the idea of random forest and applies it to different machine learning algorithms rather than different decision trees. 7 machine learning algorithms are listed below and the training data is fed into each. Required packages will be asked to be installed, please choose selection 1.

```
# Model 5 - Ensemble Models
# Models to implement
models <- c("bayesglm", "lda", "naive_bayes", "glmnet", "gamLoess", "pda", "xgbLinear")

# Function to apply each of the models into the caret train function
fits <- pblapply(models, function(model){
  print(model)
  caret::train(winner ~ . -tourney_date - dfRate.p0 - dfRate.p1
                         - fSrvInRate.p0 - fSrvInRate.p1 - fSrvWonRate.p0
                         - fSrvWonRate.p1 - SSrvWonRate.p0 - SSrvWonRate.p1
                         - bpSavedRate.p0 - bpSavedRate.p1,
               method = model,
               data = training)
})
```

```
## [1] "bayesglm"
## [1] "lda"
## [1] "naive_bayes"
## [1] "glmnet"
## [1] "gamLoess"
## [1] "pda"
## [1] "xgbLinear"
```

```
# Apply name of models to each of the model
names(fits) <- models

# Function to create a predicition for each model
pred <- pbsapply(fits, function(object)
  predict(object, newdata = testing))
dim(pred)
```

```
## [1] 2898    7
```

```
# Work out accuracy of each model
accEnsem <- colMeans(pred == testing$winner)
accEnsem
```

```
##    bayesglm         lda naive_bayes      glmnet    gamLoess         pda
##       0.663       0.661       0.646       0.658       0.656       0.661
```

```
##   xgbLinear
##       0.635
```

```
mean(accEnsem)
```

```
## [1] 0.654
```

```
# Implement voting system
votes <- rowMeans(pred == "0")
y_hat <- ifelse(votes > 0.5, "0", "1")

# Add to accuracy metric
accuracy <- bind_rows(accuracy,
                      data_frame(Method = "Ensemble Models",
                                 Accuracy = mean(y_hat == testing$winner)))
accuracy %>% knitr::kable()
```

Method	Accuracy
baseline	0.638
Logistic Regression	0.663
Logistic Regression no serve Stats	0.663
CART Model	0.590
Random Forest Model	0.645
Ensemble Models	0.659

```
# See how many predictions from the ensemble model matches the pruned logistic regression model
mean(y_hat == logitPred2B)
```

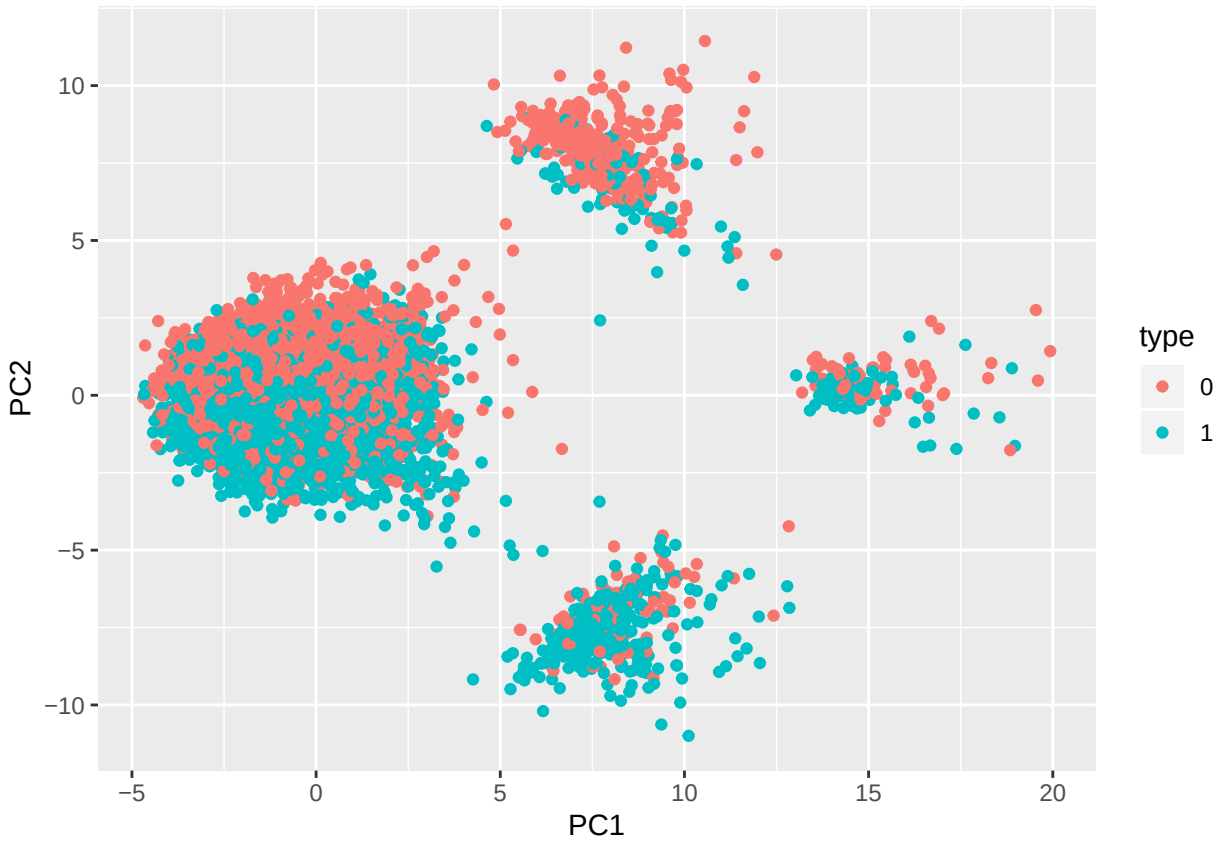
```
## [1] 0.991
```

From the results table, the ensemble model performs quite well. Interestingly many of the models in the ensemble perform at or beat the performance level of the Logistic regression 'glm' model. These models (bayesglm, lda, pda) could be explored further in future through optimisation of parameters. Also the predictions of the ensemble model are almost identical to the predictions of the pruned logistic regression model; 99.6% of predictions are the same.

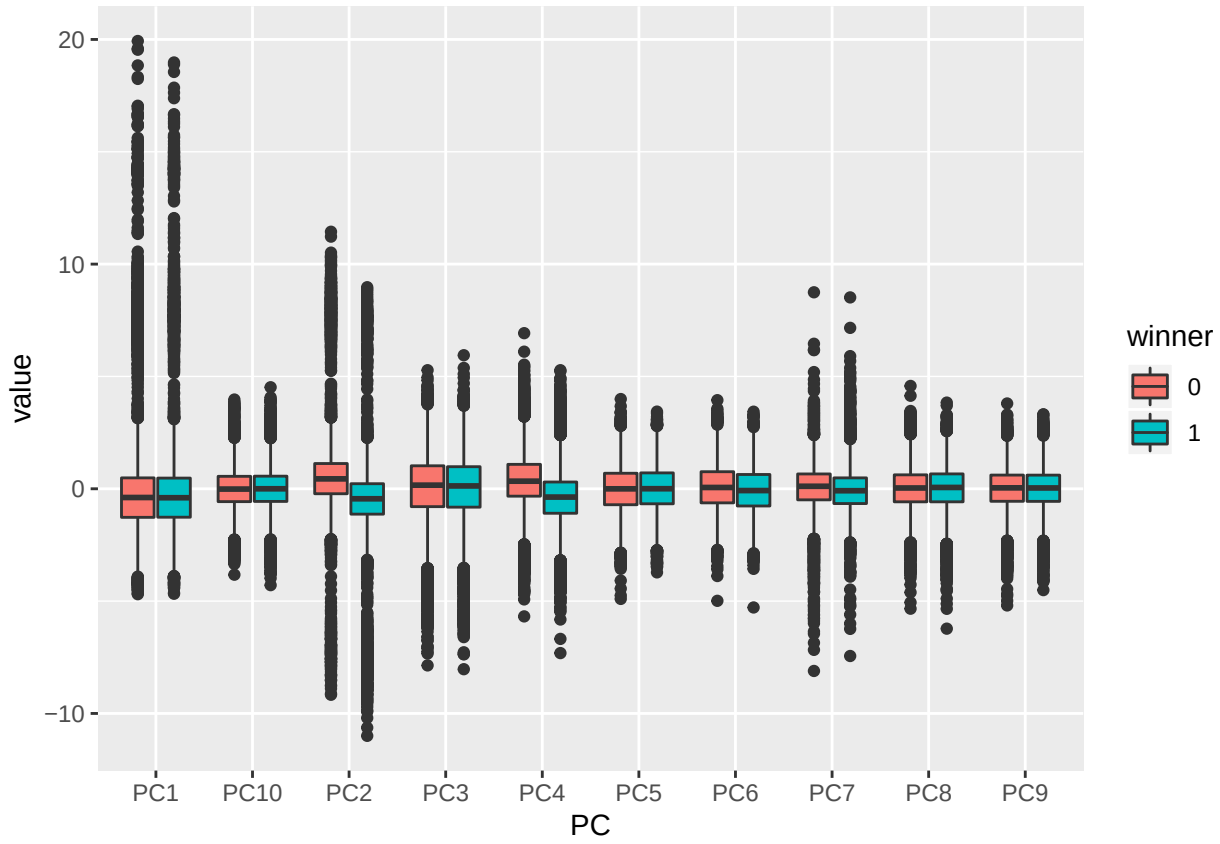
2.7.6 PCA & Logistic Regression

PCA Analysis uses orthogonal transformation to convert variables into a set of linearly uncorrelated variables called principle components. The principle components are generated in order of variance by the algorithm.

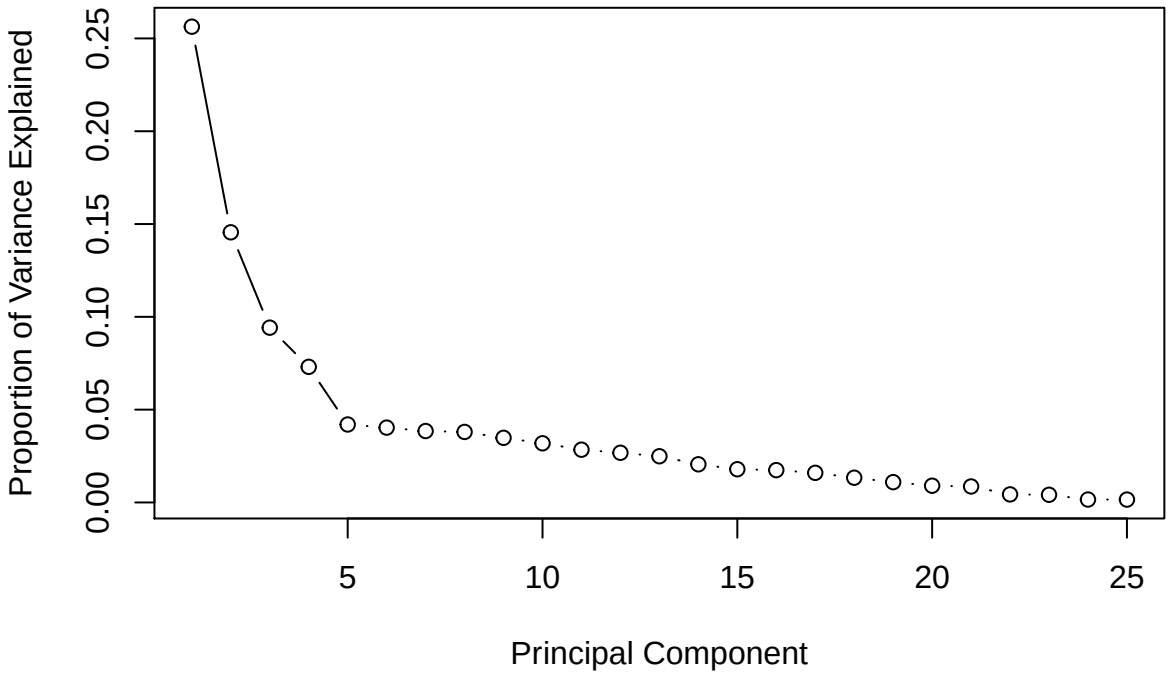
```
# Complete PCA
pcomp <- prcomp(training %>% select(-c(tourney_date, winner)), scale. = T)

# Scatter plot of first 2 principles components colour by winner
data.frame(pcomp$x[,1:2], type = training$winner) %>%
  ggplot(aes(PC1, PC2, color = type)) +
  geom_point()
```



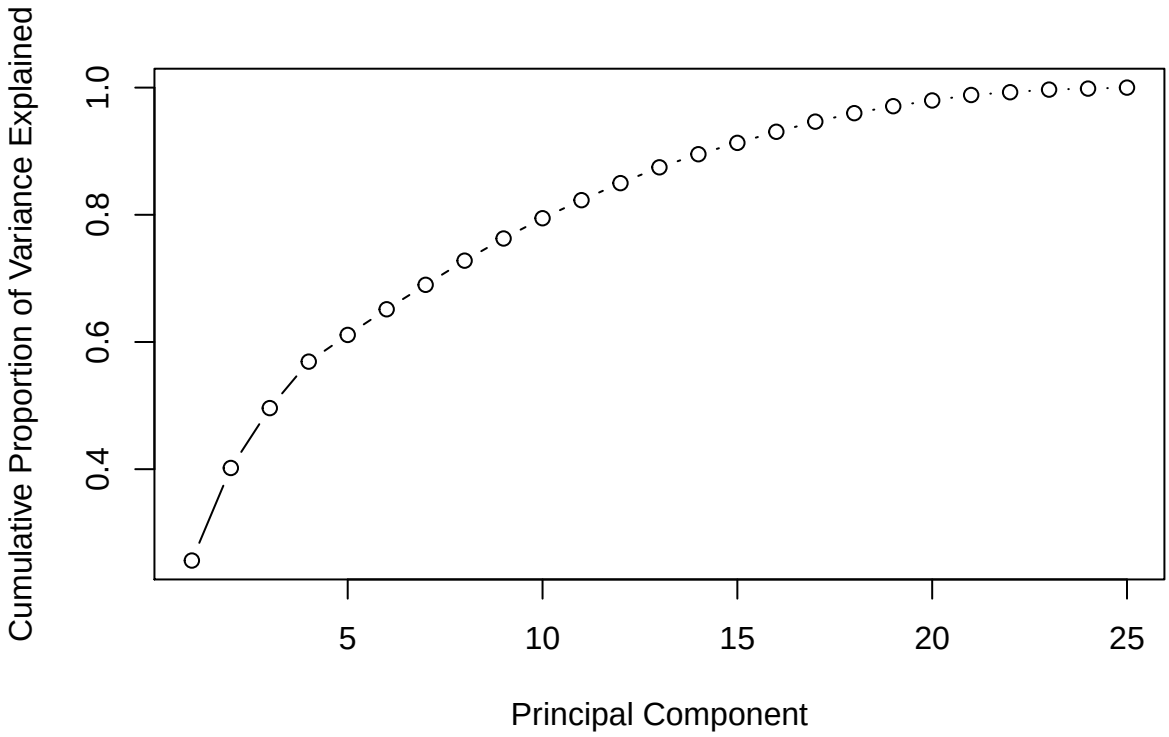

```
# Box plot of the first 10 principle components
data.frame(winner = training$winner, pcomp$x[,1:10]) %>%
  gather(key = "PC", value = "value", -winner) %>%
  ggplot(aes(PC, value, fill = winner)) +
  geom_boxplot()
```




```
# Compute variance of each principle component
pvar <- pcomp$sdev^2

# Compute proportion of variance explained
propvar <- pvar / sum(pvar)

# Plot of Variance explained by each principle component
plot(propvar, xlab = "Principal Component",
     ylab = "Proportion of Variance Explained",
     type = "b")
```




```
# Cumulative variance explained by each principle component
plot(cumsum(propvar), xlab = "Principal Component",
     ylab = "Cumulative Proportion of Variance Explained",
     type = "b")
```




```
# Training Data frame for the first 13 principle components
trainPCA <- as.data.frame(pcomp$x)[,1:13] %>%
  cbind(training$winner) %>% rename(winner = `training$winner`)
```

```r
# Testing Data frame for the first 13 principle components, transformations according to pcomp
testPCA <- as.data.frame(predict(pcomp, testing))[,1:13] %>%
  cbind(testing$winner) %>% rename(winner = `testing$winner`)

# Validation Data frame for the first 13 principle components, transformations according to pcomp
validationPCA <- as.data.frame(predict(pcomp, validation))[,1:13] %>%
  cbind(validation$winner) %>% rename(winner = `validation$winner`)

# Logistic Model using PCA
logitPCA <- caret::train(winner ~ .,
                         data = trainPCA,
                         method = 'glm',
                         na.action = NULL)

summary(logitPCA)
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.529  -1.007  -0.302   1.010   2.771
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.36e-02   1.57e-02   -0.87    0.387
## PC1         -8.39e-05   6.34e-03   -0.01    0.989
## PC2         -3.95e-01   9.65e-03  -40.89  < 2e-16 ***
## PC3         -2.13e-02   1.07e-02   -2.00    0.045 *
## PC4         -5.18e-01   1.37e-02  -37.90  < 2e-16 ***
## PC5          2.37e-02   1.54e-02    1.54    0.124
## PC6         -1.52e-01   1.58e-02   -9.58  < 2e-16 ***
## PC7         -1.22e-01   1.76e-02   -6.91  4.8e-12 ***
## PC8         -7.82e-03   1.66e-02   -0.47    0.637
## PC9         -1.65e-03   1.71e-02   -0.10    0.923
## PC10         1.84e-02   1.77e-02    1.04    0.298
## PC11         5.39e-03   2.01e-02    0.27    0.789
## PC12        -4.27e-02   1.93e-02   -2.21    0.027 *
## PC13        -6.84e-03   2.07e-02   -0.33    0.741
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 27976  on 20180  degrees of freedom
## Residual deviance: 23995  on 20167  degrees of freedom
## AIC: 24023
##
## Number of Fisher Scoring iterations: 4
```

```r
# Predict on testing set
logitPCAPred <- predict(logitPCA, newdata = testPCA)
```

```r
summary(logitPCAPred)
```

```
##    0    1
## 1469 1429
```

```r
# Compare values
logitPCACorrect <- (logitPCAPred == testing$winner)
table(logitPCACorrect)
```

```
## logitPCACorrect
## FALSE  TRUE
##  1003  1895
```

```r
# Work out accuracy
accuracy <- bind_rows(accuracy,
                      data_frame(Method = "Logistic Regression with PCA",
                                 Accuracy = sum(logitPCACorrect) / length(logitPCACorrect)))
accuracy %>% knitr::kable()
```

Method	Accuracy
baseline	0.638
Logistic Regression	0.663
Logistic Regression no serve Stats	0.663
CART Model	0.590
Random Forest Model	0.645
Ensemble Models	0.659
Logistic Regression with PCA	0.654

The analysis above uses PCA to run logistics regression. The first 11 principle components explains over 80% of the variance. Selecting the first 13 principle components and running logistic regression gives an accuracy of 65.3%. Our refined logistics regression model in section 2.7.2.1 is still the most accurate model. Interestingly, not all principle compenents are statistically significant in the logistics regression model. Howver the ones that are have the highest variances between winners being 1 & 0 in the box plot.

3. Results

The models produced have varying results. The best models were the logistic regression with some of the variables pruned away and the ensemble model. Some other models were explored but not included in this report. A neural network was attempted; however convergence on a model was not achieved. A neural network was also attempted using PCA variables, this did not result in convergence either. Investigation of the results has shown that improvements in accuracy could be made from better feature extraction techniques and having a better dataset. In the rest of this section we will build a final logistic regression model as well as a final ensemble model.

3.1 Final Logistic Regression Model

A final model is to now be trained on both the training and testing set to test on the validation set using the same parameters as model 2.7.2.1.

```r
# Baseline; if a player is has a lower rank predict win on validation set
baselineFinalModel <- validation %>%
  select(winner, rank.p0, rank.p1) %>%
  filter(!is.na(rank.p0) & !is.na(rank.p1)) %>%
  mutate(baselinePred = as.integer(ifelse(rank.p0 < rank.p1, 0, 1))) %>%
  mutate(correct = ifelse(baselinePred == winner, 1, 0))
```

```r
# Table of correct predictions
table(baselineFinalModel$correct)
```

```
##
##    0    1
##  993 1583
```

```r
# Work out accuracy
accuracy <- bind_rows(accuracy,
                      data_frame(Method = "Final Baseline Model", Accuracy = mean(baselineFinalModel$co

# Final Model - Logistic Regression remove more variables
# Train model
logitModelFinal <- caret::train(winner ~ . -tourney_date - dfRate.p0 - dfRate.p1
                                - fSrvInRate.p0 - fSrvInRate.p1 - fSrvWonRate.p0
                                - fSrvWonRate.p1 - SSrvWonRate.p0 - SSrvWonRate.p1
                                - bpSavedRate.p0 - bpSavedRate.p1,
                              data = training %>%
                                rbind(testing),
                              method = 'glm',
                              na.action = NULL)

summary(logitModelFinal)
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.899  -1.020  -0.138   1.019   2.976
##
## Coefficients:
##                     Estimate Std. Error z value Pr(>|z|)
## (Intercept)       -3.17e-03   1.59e-01   -0.02  0.98413
## age.p0             1.96e-02   3.86e-03    5.09  3.6e-07 ***
## age.p1            -1.69e-02   3.83e-03   -4.42  1.0e-05 ***
## aceRate.p0        -1.82e+00   3.75e-01   -4.86  1.2e-06 ***
## aceRate.p1         1.79e+00   3.78e-01    4.74  2.2e-06 ***
## rankPts.p0        -2.05e-04   1.24e-05  -16.58  < 2e-16 ***
## rankPts.p1         2.22e-04   1.30e-05   17.15  < 2e-16 ***
## rank.p0            1.35e-03   1.71e-04    7.91  2.5e-15 ***
## rank.p1           -1.75e-03   1.92e-04   -9.12  < 2e-16 ***
## surfaceWinRate.p0 -1.37e+00   9.56e-02  -14.35  < 2e-16 ***
## surfaceWinRate.p1  1.31e+00   9.59e-02   13.63  < 2e-16 ***
## form.p0           -4.26e-01   8.54e-02   -4.99  6.1e-07 ***
## form.p1            3.25e-01   8.74e-02    3.71  0.00021 ***
## fatigue.p0        -3.31e-04   6.10e-05   -5.43  5.6e-08 ***
## fatigue.p1         3.50e-04   6.14e-05    5.70  1.2e-08 ***
## h2h               -6.22e-02   1.30e-02   -4.80  1.6e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##     Null deviance: 31994  on 23078  degrees of freedom
## Residual deviance: 27381  on 23063  degrees of freedom
## AIC: 27413
##
## Number of Fisher Scoring iterations: 4
```

```
# Predict on testing set
logitPredFinal <- predict(logitModelFinal, newdata = validation)

summary(logitPredFinal)
```

```
##    0    1
## 1298 1278
```

```
# Compare values
logitCorrectFinal <- (logitPredFinal == validation$winner)
table(logitCorrectFinal)
```

```
## logitCorrectFinal
## FALSE  TRUE
##   921  1655
```

```
# Work out accuracy
accuracy <- bind_rows(accuracy,
                      data_frame(Method = "Final Logistic Regression Model on Validation",
                                 Accuracy = sum(logitCorrectFinal) / length(logitCorrectFinal)))
accuracy %>% knitr::kable()
```

Method	Accuracy
baseline	0.638
Logistic Regression	0.663
Logistic Regression no serve Stats	0.663
CART Model	0.590
Random Forest Model	0.645
Ensemble Models	0.659
Logistic Regression with PCA	0.654
Final Baseline Model	0.615
Final Logistic Regression Model on Validation	0.642

Our final model follows the same trend as our test algorithm. There is roughly a 3% improvement in accuracy when compared to the baseline. Interestingly our final baseline model only provided a accuracy of 61.5% meaning in 2019 we had a lower win rate for the favourite.

3.2 Final Ensemble Model

A final ensemble of models, the same algorithms as the ones used above are trained on both the testing and training set and tested on the validation set.

```
# Final Ensemble Model
# Models to implement
models <- c("bayesglm", "lda", "naive_bayes", "glmnet", "gamLoess", "pda", "xgbLinear")

# Function to apply each of the models into the caret train function
finalFits <- pblapply(models, function(model){
  print(model)
  caret::train(winner ~ . -tourney_date - dfRate.p0 - dfRate.p1
                          - fSrvInRate.p0 - fSrvInRate.p1 - fSrvWonRate.p0
                          - fSrvWonRate.p1 - SSrvWonRate.p0 - SSrvWonRate.p1
```

```
                              - bpSavedRate.p0 - bpSavedRate.p1,
                method = model,
                data = training %>%
                           rbind(testing))
})
```

```
## [1] "bayesglm"
## [1] "lda"
## [1] "naive_bayes"
## [1] "glmnet"
## [1] "gamLoess"
## [1] "pda"
## [1] "xgbLinear"
```

```
# Apply name of models to each of the model
names(finalFits) <- models

# Function to create a predicition for each model
predFinal <- pbsapply(finalFits, function(object)
  predict(object, newdata = validation))

# Work out accuracy of each model
accFinEnsem <- colMeans(predFinal == validation$winner)
accFinEnsem
```

```
##    bayesglm         lda naive_bayes      glmnet    gamLoess         pda
##       0.642       0.646       0.640       0.642       0.648       0.646
##   xgbLinear
##       0.625
```

```
mean(accFinEnsem)
```

```
## [1] 0.641
```

```
# Implement voting system
votesFinal <- rowMeans(predFinal == "0")
y_hatFinal <- ifelse(votesFinal > 0.5, "0", "1")

# Add to accuracy metric
accuracy <- bind_rows(accuracy,
                      data_frame(Method = "Final Ensemble Models",
                                 Accuracy = mean(y_hatFinal == validation$winner)))
accuracy %>% knitr::kable()
```

Method	Accuracy
baseline	0.638
Logistic Regression	0.663
Logistic Regression no serve Stats	0.663
CART Model	0.590
Random Forest Model	0.645
Ensemble Models	0.659
Logistic Regression with PCA	0.654
Final Baseline Model	0.615
Final Logistic Regression Model on Validation	0.642
Final Ensemble Models	0.643

Our final ensemble model performs well when tested on the validation set. It performs better than the logistic

regression model.

4 Conclusion

This project explored the ATP dataset of men tennis results and used it to build machine learning models and test on different sets of data. A range of features were extracted from the stats and fed into the model to build predictions on the results of matches. A range of machine learning models were explored; the best model found in the project was the ensemble model and logistic regression. Testing on a holdout validation set showed the same level of performance improvement. Some of the models used in the ensemble could be further explored as standalone models for potentially better results.

Other models were experimented with during the project however they did not converge. Further work in the area can be centered on either using more advanced techniques such as Gradient Boosting, Neural Networks or Support Vector Machines to name a few. For these models to work well, the key lesson learnt from this project needs to be implemented; smarter feature extraction. Further work can also be done optimising the parameters of different models used in the project including building some of the models used in the ensemble as stand-alone models and optimising.