ALY6015 - Intermediate Analytics

Team: Predictor
Member: Bhavik, Shreyas and Pravin
Instructor : Dr. Vladimir Shapiro
Project Final Report

Introduction

The use of data analytics has become increasingly popular in sports. Teams and organizations are utilizing data analysis to improve player performance, game strategy, and overall team success. In this exercise, we will be exploring tennis data. Tennis is a sport played among two or four players, where points are awarded to the player when their opponent fails to return the ball. Using tennis data, we will solve two real-time problems for the tennis community. The first problem involves predicting the duration of a match based on the performance of both players. The match duration will be predicted to assist the broadcast company in determining the number of advertisements that can be shown during the game. We will use linear regression to make the match duration prediction, using player performance as our independent variable. The second problem involves predicting the winner or loser of a match to help the betting company set a basic bet for the player. We will use logistic regression to predict a winner or loser, taking into account factors such as aces, double faults, total games won, and total tiebreakers won.

Problem Statement 1

Predicting match duration based on both player's match performance data using linear regression method

Description :

Reason for solving a problem - XYZ sports channel often broadcasts live tennis matches throughout the year. The advertising that is broadcasted throughout the games is how these stations make money. The marketing team of the channel wanted to attract clients who would like to sell their advertisements. For this, the team wants to know the match time duration beforehand so they might come up with several commercials to air during a game. We would like to help the marketing team by providing them with the approximate match time using prediction model techniques.

Method :

Method: Linear Regression

Dependent Variable: Match Duration

Independent Variable: Aces, Double Faults, First Serves In, Break Points Saved, Break Points Converted, Total Points Won, Games Won, Tiebreaks Won, Surface

Reason for consideration of Independent Variable is explained below:

Information on Independent Variables and Reasons for Choosing them

Independent Variables	Defination	Impact on Match duration

Independent Variables	Defination	Impact on Match duration
Aces	Refers to a serve that is not touched by the receiver and results in a point for the server	More aces less will be the match duration as point is scored in first serve shot
Double Faults	A player fails to get their serve into the opposite service box on two consecutive attempts	More DF results in more serves thus higher match duration
First Serves In	First attempt a player makes to serve the ball into the service box	More First serve in results in less over serves thus lower is the match duration
Break Points Saved	A situation where the serving player is one point away from winning the game and Opponent saved the point	More Break Point saved would result in more games thus higher match duration
Break Points Converted	Server score a breakpoint	More Break Point Converted would result in less games thus higher match duration
Total Points Won	Number of points scored in a match	More Points would result in high match duration
Games Won	Number of games won in match	More Points would result in high match duration
Tiebreaks Won	Number of special game won- tiebreak game	More Points would result in high match duration
Surface	The type of court on which a match is played.	Clay surface is easy to play thus more rallies higher match duration

Table 1 - Independent Variable for Problem statement 1

Problem Statement 2

Predicting match winner based on players match performance data using logistic regression.

Description :

XYZ gambling company often wanted to rightly predict the winner to place a bet. To rightly predict, they wanted to use the analytical approach, thus we will be helping them with a predicted model which will help them to understand the winner beforehand.

Method :

Method: Logistic Regression.

Dependent Variable: Winner /Loser

Independent Variables: Aces, Double Faults, First Serves In, Break Points saved, Break Points Converted, Total points won, Game won, Tiebreaks won
Reason for consideration of Independent Variable is explained below:

Variables and their impact on match

Variables	**Impact on match**
Aces	More aces, more probability of winning the match
Double Faults	More DF, less probability of winning the match
First Serves In	More first serve in, more probability of winning the match
Break Points saved	Increase in BP saved increases probability of winning the match
Break Points Converted	Increase in BP converted increases probability of winning the match
Total points won	More Total points won, more probability of winning the match
Game won	More Games won, more probability of winning the match
Tiebreaks won	More tiebreaks won, more probability of winning the match

Table 2 - Independent Variable for Problem statement 2

Exploratory Data Analysis(EDA)

To address our business query we first need to collect our data. We will be using data from three different sources, so the first step is to merge the datasets. After that, we need to select the relevant variables, and then proceed with data preprocessing, which involves checking and removing null and duplicate values.

Loading the Dataset

```
##Reading Match Details
Match_stats <- read.csv("C:\\Users\\bhavi\\Desktop\\MPS Analytics\\Intermediate Analysis\\Project\\Tennis Data\\Tenn
is Data\\match_stats_2017_unindexed_csv.csv")

##Reading Tournament Details
Match_detail<- read.csv("C:\\Users\\bhavi\\Desktop\\MPS Analytics\\Intermediate Analysis\\Project\\Tennis Data\\Tenn
is Data\\match_scores_2017_unindexed_csv.csv")

##Reading Tournament Details
Tournament_detail<-read.csv("C:\\Users\\bhavi\\Desktop\\MPS Analytics\\Intermediate Analysis\\Project\\Tennis Data
\\Tennis Data\\tournaments_1877-2017.csv")
head(Tournament_detail)
```

Data Preparation

```
Match_stats <-merge(Match_stats,Match_detail,by.x = "match_id", by.y = "match_id",
                        all.x = TRUE)
##Keeping only important tournament details
Tournament_detail <- Tournament_detail[, c("tourney_year_id",
                                           "tourney_name",
                                          "tourney_location",
                                          "tourney_month",
                                          "tourney_conditions",
                                          "tourney_surface" )]

##Joining Tournament details to the match data
Match_result <-merge(Match_stats,Tournament_detail,by.x = "tourney_year_id", by.y = "tourney_year_id", all.x = TRUE)

##Dropping irrelevant columns
colnames(Match_result)
Match_result <- Match_result %>% dplyr::select(-c("tourney_order.y",
                                                  "tourney_order.y" ,
                                                  "match_stats_url_suffix.x",
                                                  "match_stats_url_suffix.y" ,
                                                  "match_time",
                                                  "tourney_location",
                                                  "tourney_slug",
                                                  "tourney_url_suffix" ,
                                                  "tourney_round_name",
                                                  "match_order",
                                                  "winner_name" ,
                                                  "winner_player_id",
                                                   "winner_slug",
                                                  "loser_name",
                                                  "loser_player_id" ,
                                                  "loser_slug",
                                                  "winner_seed",
                                                  "loser_seed"))

Match_result <- Match_result %>% dplyr::select(-ends_with("total"))
colnames(Match_result)
```

Understanding the data dimension and structure

Dataset Dimensions

Numer_Obs	# of Observation	3815
Num_Col	# of Numeric Variables	38
Cat_Col	# of Categorical Variables	6

Table 3 - Dataset Dimensions

There are total 3,815 observations in our dataset and a total of 44 variables out of which 38 are numeric variables and 6 are categorical variables

Checking for NA values

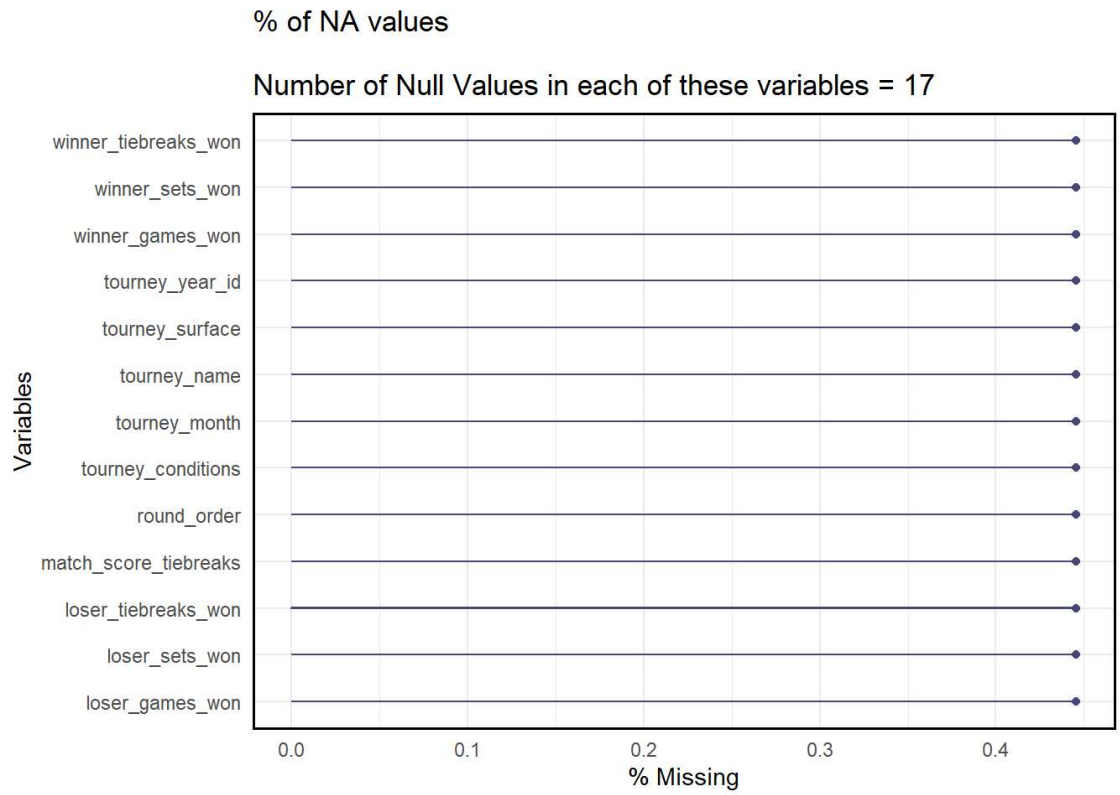

Figure 1 - % of NA values within Variables

From the table and graph we see that only 13 variables has missing values which is less than 0.5% of the total observation, thus removing the NA rows from the dataset

```
#Dropping NA Values
Match_result <- na.omit(Match_result)
```

Checking for Duplicate rows

# of Duplicate rows

Number of duplicates in data	6

Table 4 - # of Duplicate values in Data set

There are 6 duplicates which are less than 0.2% of the the total observation thus, removing duplicates from the data

```
##Removing Duplicates
Match_result <- Match_result[!duplicated(Match_result$match_id),]
```

Now that the data pre-processing has been finished, we can examine how many matches have a recorded match duration, analyze the distribution of match duration, handle any extreme values in match duration as outliers, and then assess the relationship between various variables and match duration.

In addition, we will examine how the winner and loser of each match perform in terms of match performance metrics, such as aces, double faults, total points won, and so on.

```
##Counting Unique tournaments and Matches in Data
kable(cbind(c("# of Tournaments", "# of Matches"),
            rbind(n_distinct(Match_result$tourney_year_id),
      n_distinct(Match_result$match_id))),
      align = "c",
      #col.names = "Count",
      #caption = "<font><b><center> # of NA values within Variables",
      digits = 2) %>%
   kable_styling(bootstrap_options = c("striped", "hover","bordered"))
```

# of Tournaments	66
# of Matches	3792

Table 5 - #Tournament and #Matches

The dataset has details of 67 tournaments and ~3,792 matches

Checking the Match Duration Distribution

```
kable(t(quantile(Match_result$match_duration,probs = seq(0,1,0.05))),
      align = "c",
      #col.names = "Count",
      caption = "<font><b><center> Distribution of Match Time ",
      digits = 2) %>%
   kable_styling(bootstrap_options = c("striped", "hover","bordered"),font_size = 12)
```

Distribution of Match Time

0%	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%	100%
8	55	62	66	70	74	79	83	87	91	96	101	107	113	119	126	133	141	152	170.45	315

Table 6 - Distribution for Match Duration

```
ggplot(data = Match_result, aes(x = match_duration)) +
  geom_histogram(fill = "tan1", color = "black") +
  labs(x = "Duration(min)", y = "Frequency", title = "Match Duration Distribution") +
  theme(plot.title = element_text(color = "black", size = 14, face = "bold"),
        panel.border = element_rect(color = "black", fill = NA, size = 1))
```

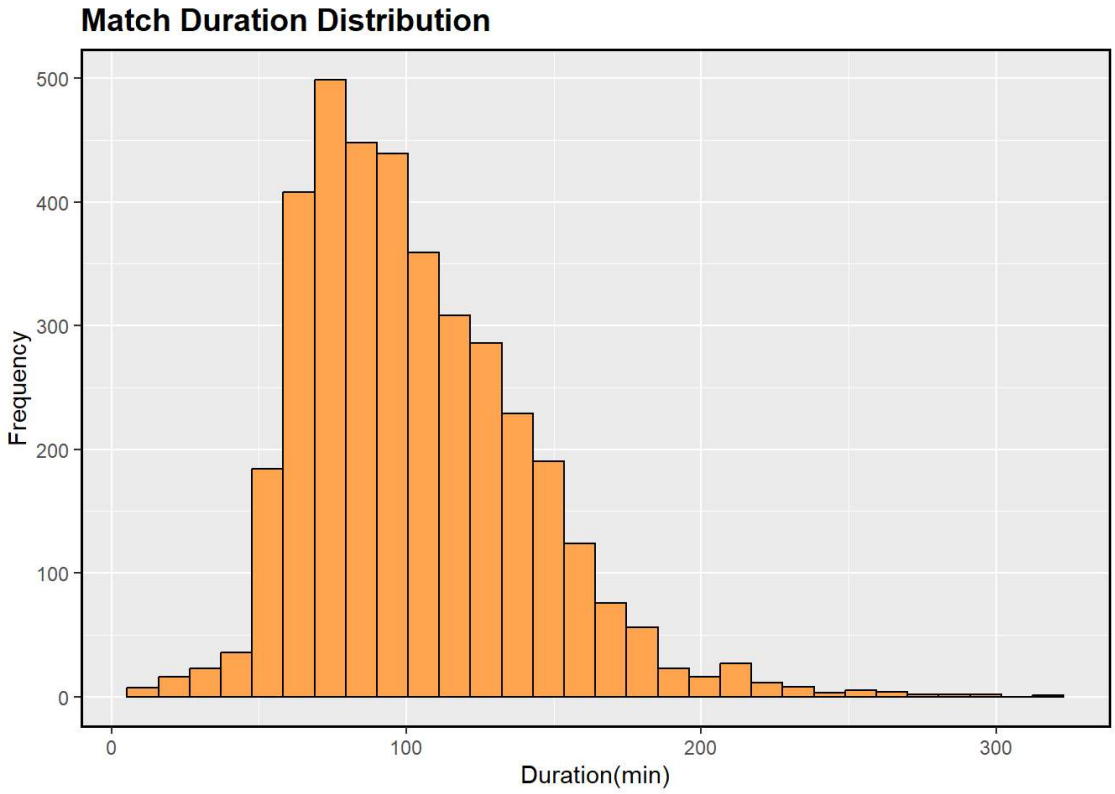

Figure 2 - Match Duration Distribution

From the histogram and quantile distribution, it is observed that most of the matches in our dataset(~55%) has a duration between 60min to 120mins.Only 10% of the matches are played beyond 150mins. There a huge gap between 95% and 100% , thus need to dig deep to understand the outlier in match duration

```
##Quantile distribution of match duration
quantile(Match_result$match_duration,probs = seq(0,1,0.05))
quantile(Match_result$match_duration,probs = seq(0.95,1,0.01))

##Calculating upper and lower limit of match duration
Lower_Limit <-  quantile(Match_result$match_duration, 0.25) - 1.5*IQR(Match_result$match_duration)
Upper_Limit <- quantile(Match_result$match_duration, 0.75) + 1.5*IQR(Match_result$match_duration)
```

Distribution of Match Time

95%	96%	97%	98%	99%	100%
170.45	175.36	183.27	200.18	217.09	315

Table 7 - Distribution of Match Time for outlier treatment

Looking into quantile(95% to 100%), upper and lower distribution, we see there is huge gap between 98% and 100 thus removing this 2% of the data from the dataset

```
##Removing Outliers from Match Duration
Match_result <- Match_result[Match_result$match_duration <= 200,]
```

Given our understanding of the game, we had anticipated the impact of each match performance variable on match duration as stated in the problem statement's method. With the availability of data, we can now examine how match duration is influenced by the independent variable and how match performance metric

differ for winners and losers

1- Impact of both players Double Fault on match duration

```
##Scatter between the Winner Double Faults & Loser Double Faults
plot4<-ggplot(Match_result,aes(winner_double_faults,match_duration,col=Match_result$match_duration))+
  xlab("Double Faults: Winner")+
  ylab("Match Duration")+
  geom_point() +
  labs(color = "Match Duration") +
  theme(plot.title = element_text(color = "black", size = 14, face = "bold"),
        panel.border = element_rect(color = "black", fill = NA, size = 1))

plot5<-ggplot(Match_result,aes(loser_double_faults,match_duration,col=Match_result$match_duration))+
  xlab("Double Faults: Loser")+
  ylab("Match Duration")+
    geom_point() +
  labs(color = "Match Duration") +
  theme(plot.title = element_text(color = "black", size = 14, face = "bold"),
        panel.border = element_rect(color = "black", fill = NA, size = 1))


grid.arrange(plot4,plot5,ncol= 1,
     top = textGrob("Double Fault vs Match Duration",gp=gpar(fontsize=14,font=1)))
```

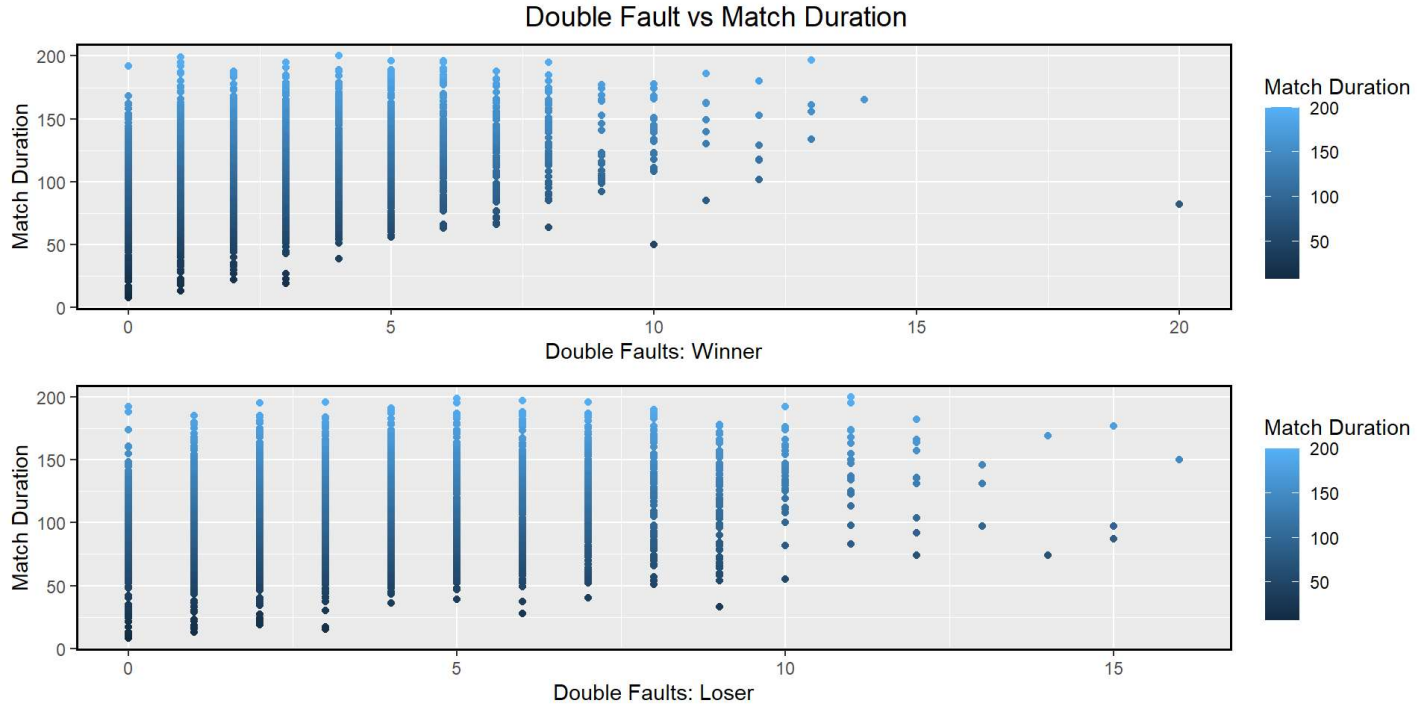

Figure 3 - Double Faults vs Match Duration

From the graph, we can see that there isn't a significant contrast between the number of double faults and the duration of the match for the winning and losing players. However, it's evident that the number of double faults made by both players affects the length of the match. When players commit a low number of double faults, there is a broad range of match durations. On the other hand, when players commit a higher number of double faults, the match duration tends to be longer

2. Impact of Number of Aces on Match duration

```
##Scatter Plot grid between the Winner Aces & Loser Aces
plot10<- ggplot(Match_result,aes(Match_result$winner_aces,Match_result$match_duration))+
  xlab("Number of Aces of Winner")+
  ylab("Match Duration")+
  #ggtitle(" Aces of Winner vs Duration of Match") +
  geom_point(col = "tan1") +
  theme(plot.title = element_text(color = "black", size = 14, face = "bold"),
        panel.border = element_rect(color = "black", fill = NA, size = 1))



plot11<- ggplot(Match_result,aes(Match_result$loser_aces,Match_result$match_duration))+
  xlab("Number of Aces of Loser")+
  ylab("Match Duration")+
  #ggtitle("Aces of Loser vs Duration of Match")+
  geom_point(col= "lightblue") +
  theme(plot.title = element_text(color = "black", size = 14, face = "bold"),
        panel.border = element_rect(color = "black", fill = NA, size = 1))


grid.arrange(plot10,plot11,ncol=2 ,
     top = textGrob("# Aces vs Match Duration",gp=gpar(fontsize=14,font=1)))
```

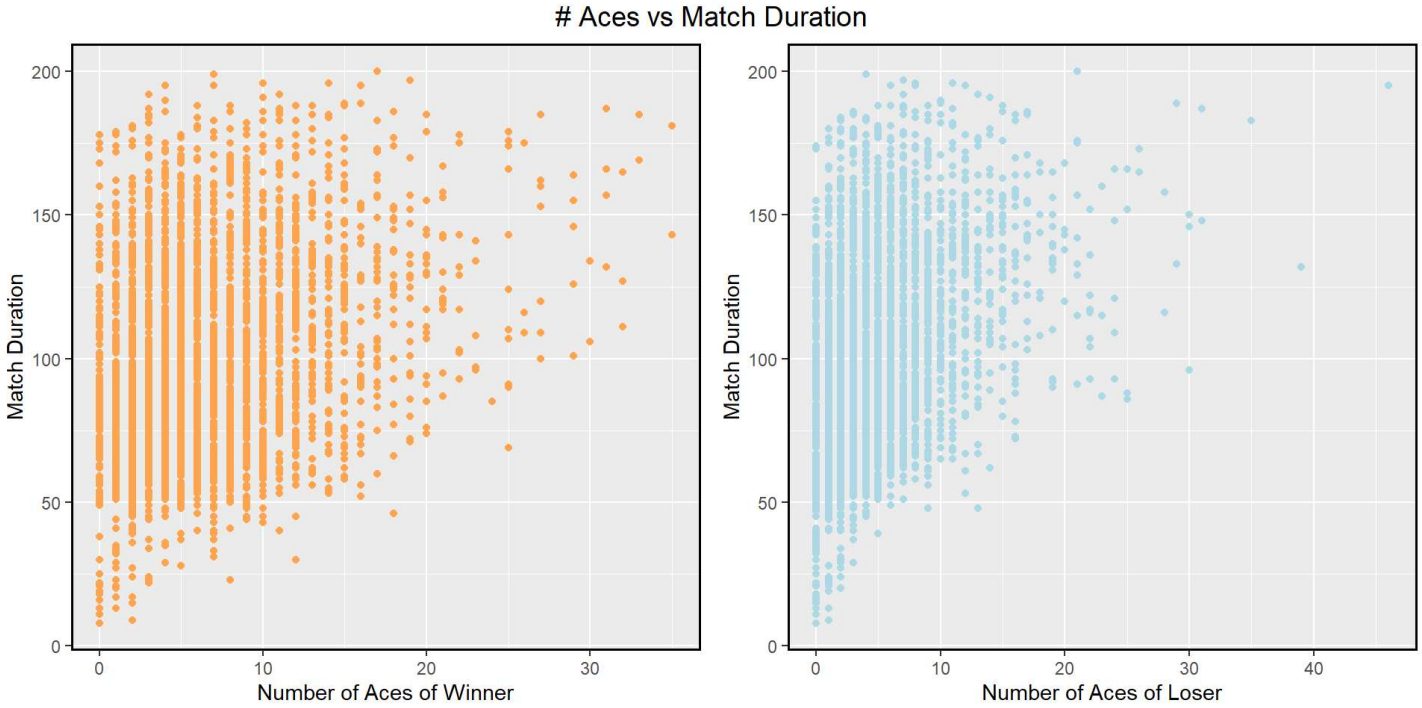

Figure 4 - Aces vs Match Duration

From the graph we can see that there no pattern followed by Aces and Match Duration. But it is observed that aces are well distributed for winner from low to high but for loser the number of aces are more towards lower number

3. Impact of Total Points Won on Match Duration

```
##Scatter plot grid between the Winner First Serve Points & Loser First Serve Points
plot6<- ggplot(Match_result,aes(Match_result$winner_total_points_won,Match_result$match_duration))+
  xlab("Total Points Won: Winner")+
  ylab("Match Duration")+
  geom_point(col = "tan1") +
  theme(plot.title = element_text(color = "black", size = 14, face = "bold"),
        panel.border = element_rect(color = "black", fill = NA, size = 1))



plot7<- ggplot(Match_result,aes(Match_result$loser_total_points_won,Match_result$match_duration))+
  xlab("Total Points Won:: Loser")+
  ylab("Match Duration")+
   geom_point(col= "lightblue") +
  theme(plot.title = element_text(color = "black", size = 14, face = "bold"),
        panel.border = element_rect(color = "black", fill = NA, size = 1))


grid.arrange(plot6,plot7,ncol=2,
     top = textGrob("Total Points Won vs Match Duration",gp=gpar(fontsize=14,font=1)))
```

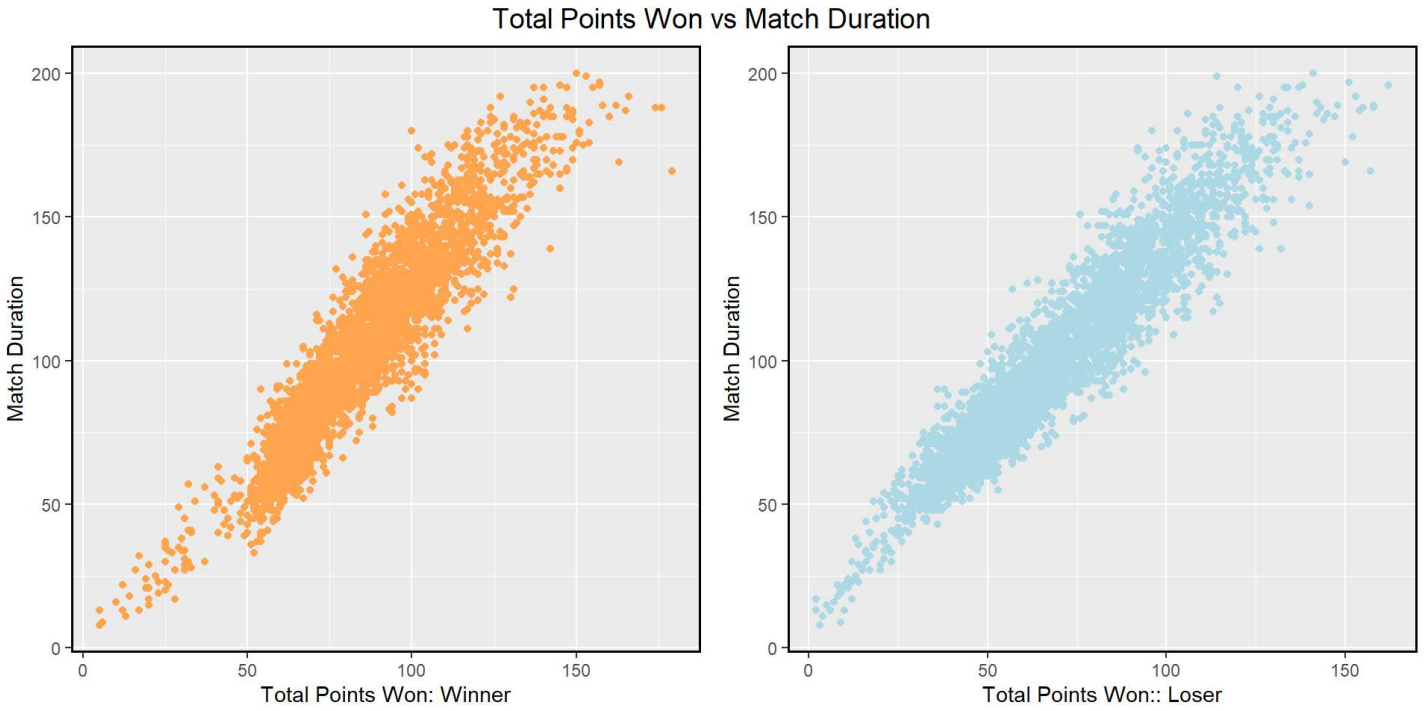

Figure 5 - Total Point Won vs Match Duration

From the graph it is clear that Total Point Won has an impact on match duration. As the Total Point Won increases we see the duration increase. Similar trend is observed for both winner and loser. There is a linear relation between Total Point Won and match duration

4. Impact of Surface on Match duration

```
##Creating dataset to compare Tournament Surface & Average Time
Surface_Distribution <- Match_result %>% group_by(tourney_surface) %>% summarise( Avg_time = mean(match_duration))
Surface_Distribution$tourney_surface<-as.factor(Surface_Distribution$tourney_surface)

##Plotting barplot
ggplot(Surface_Distribution,aes(tourney_surface,Avg_time,fill=Surface_Distribution$tourney_surface))+
  xlab("Surface of the Tennis Court")+
  ylab("Average Time")+
  geom_bar(stat="identity")+
  ggtitle("Average Time by Surface of Court", ) +
  labs(fill = "Surface") +
  theme(plot.title = element_text(color = "black",hjust = 0.5, size = 14, face = "bold"),
        panel.border = element_rect(color = "black", fill = NA, size = 1))
```

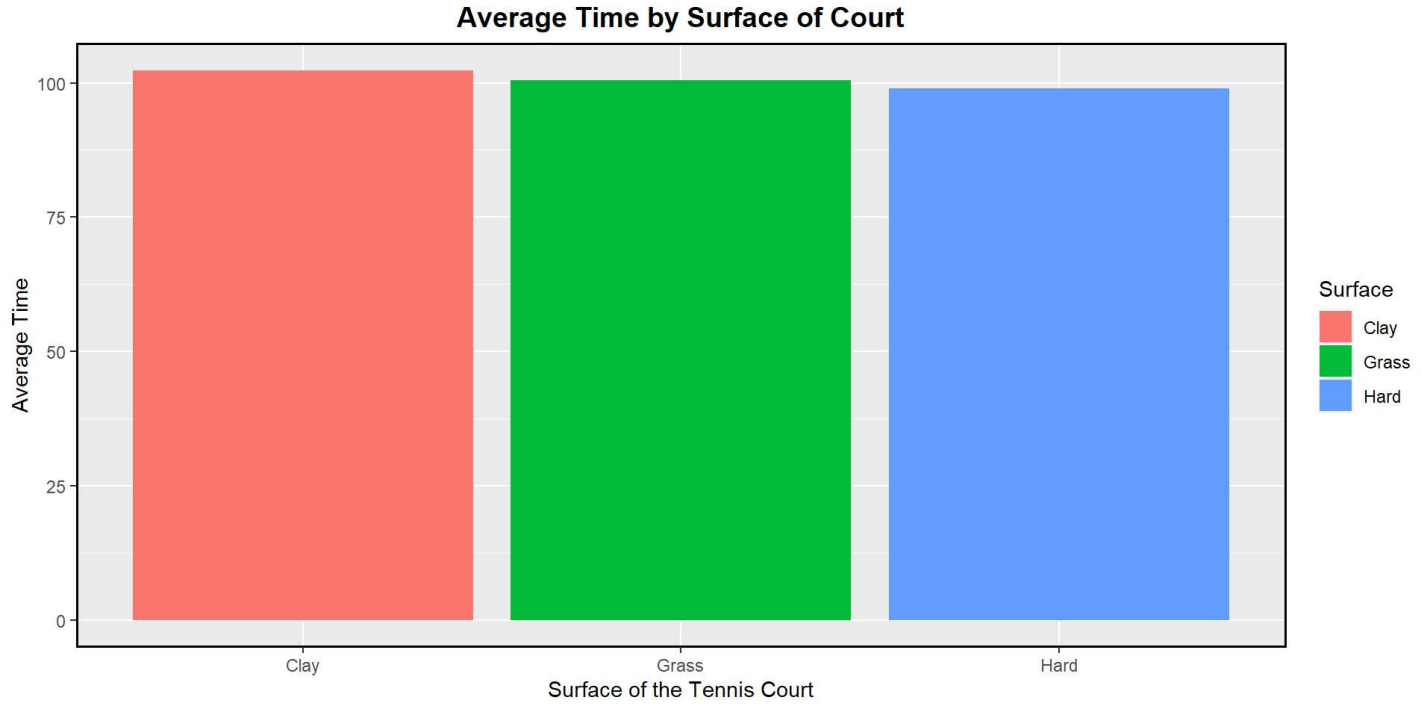

Figure 6 - Surface Impact on Match Duration

From the graph it is observed that clay surface have slightly higher match duration than other surface

5. Differences in Losers and Winners Match performance metric

```
plot_13 <- data.frame(cbind(c("Winner", "Loser"), rbind(round(mean(Match_result$winner_double_faults),2),  round(mea
n(Match_result$loser_double_faults),2)))) %>% ggplot(aes(X1,as.numeric(X2))) +
  xlab("")+
  ylab("Average of Double Faults")+
  ggtitle("Average Double Fault Distribution between Winner & Loser")+
  geom_bar(stat="identity",
       fill="tan1")+
  theme(plot.title = element_text(color = "black",hjust = 0.5, size = 11, face = "bold"),
        panel.border = element_rect(color = "black", fill = NA, size = 1))


plot_14 <- data.frame(cbind(c("Winner", "Loser"), rbind(round(mean(Match_result$winner_aces),2),  round(mean(Match_r
esult$loser_aces),2)))) %>% ggplot(aes(X1,as.numeric(X2))) +
  xlab("")+
  ylab("Average of Aces")+
  ggtitle("Average Aces Distribution between Winner & Loser")+
  geom_bar(stat="identity",
       fill="lightblue")+
  theme(plot.title = element_text(color = "black",hjust = 0.5, size = 11, face = "bold"),
        panel.border = element_rect(color = "black", fill = NA, size = 1))

plot_15 <- data.frame(cbind(c("Winner", "Loser"), rbind(round(mean(Match_result$winner_total_points_won),2),  round
(mean(Match_result$loser_total_points_won),2)))) %>% ggplot(aes(X1,as.numeric(X2))) +
  xlab("")+
  ylab("Average of Total Point Won")+
  ggtitle("Average Total Point Won  Distribution between Winner & Loser")+
  geom_bar(stat="identity",
       fill="tan1")+
  theme(plot.title = element_text(color = "black",hjust = 0.5, size = 11, face = "bold"),
        panel.border = element_rect(color = "black", fill = NA, size = 1))

plot_16 <- data.frame(cbind(c("Winner", "Loser"), rbind(round(mean(Match_result$winner_tiebreaks_won),2),  round(mea
n(Match_result$loser_tiebreaks_won),2)))) %>% ggplot(aes(X1,as.numeric(X2))) +
  xlab("")+
  ylab("Average of Tiebreaks Won")+
  ggtitle("Average Tiebreaks Won Distribution between Winner & Loser")+
  geom_bar(stat="identity",
       fill="lightblue")+
  theme(plot.title = element_text(color = "black",hjust = 0.5, size = 11, face = "bold"),
        panel.border = element_rect(color = "black", fill = NA, size = 1))

grid.arrange(plot_13,plot_14, plot_15,plot_16, ncol=2)
```

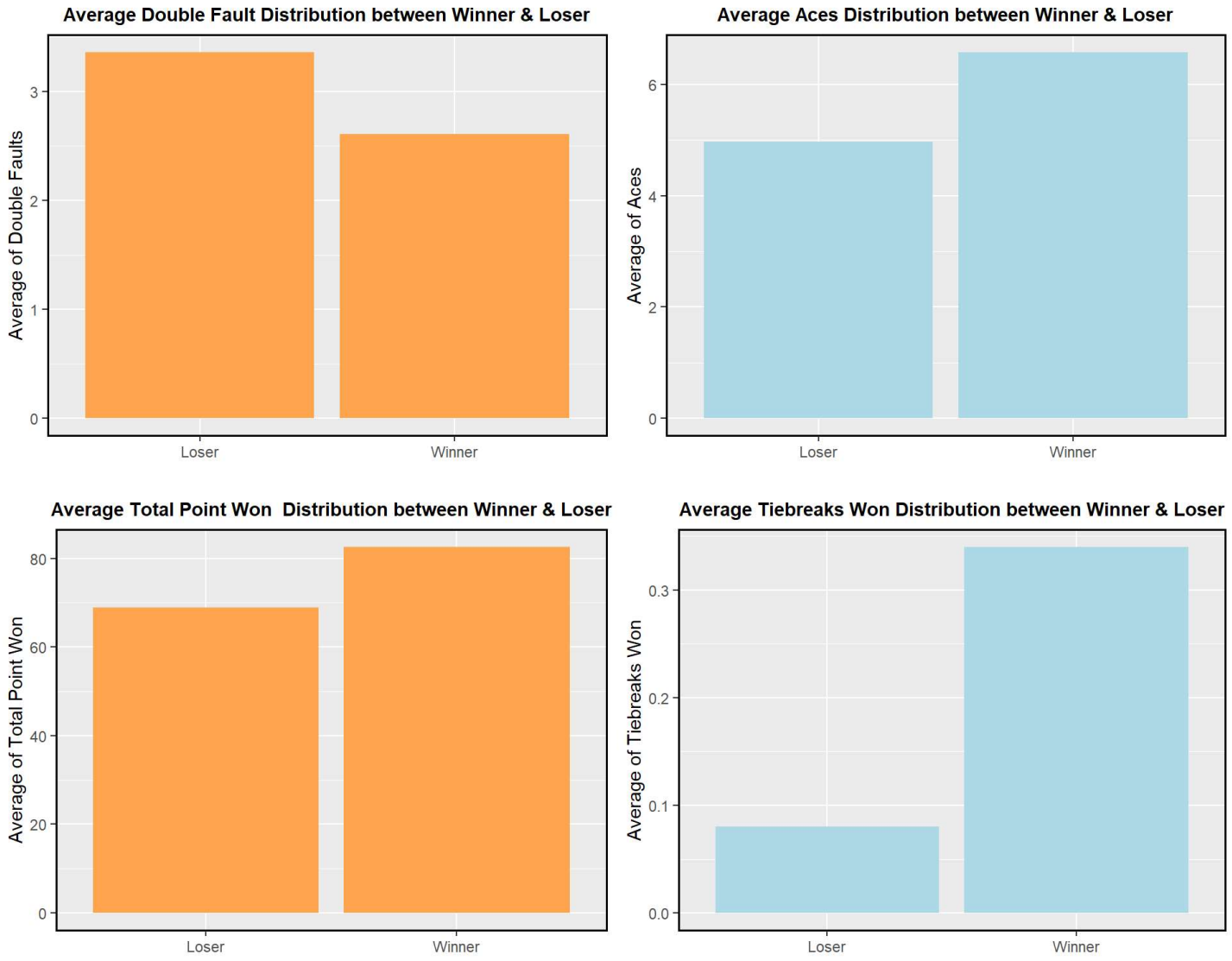

Figure 7 - Difference in Match Performance metric between winner & loser

The graph indicates that winners have a higher number of aces, total points won, and tie break points compared to losers. On the other hand, losers tend to have more double faults. Therefore, it can be inferred that a player's chances of winning are reduced if they have a higher number of double faults, but increased if they score more aces and points.

In order to make an accurate prediction about the duration of a match, it is crucial to determine the variables that are highly correlated with the match duration. These variables can then be used to develop a model that estimates the expected length of the match. To accomplish this, one possible method is to construct a correlation matrix and examine the correlations between various variables and the match duration. By creating a correlation plot, we can gain insights into how different variables are correlated with the duration of the match.

When predicting the duration of a future match, it may not be known in advance which player will emerge as the winner and which will be the loser. As a result, it is common practice to designate one player as Player 1 and the other as Player 2, regardless of their eventual outcome. This allows for the development of predictive models that do not depend on knowledge of the winner or loser in advance, and can therefore provide accurate estimates of match duration regardless of the eventual outcome.

```
##Changing the column names
colnames(Match_result) <- gsub("^winner", "P1", colnames(Match_result))
colnames(Match_result) <- gsub("^loser", "P2", colnames(Match_result))

##Filtering Numerical Variables
Numerical_Vraiable <- select_if(Match_result, is.numeric)
```

```
##Plotting Correlation Matrix
corr <- cor(Numerical_Vraiable)
##Correlation plot
corrplot(corr,  order="original", tl.cex = 1.5)
```

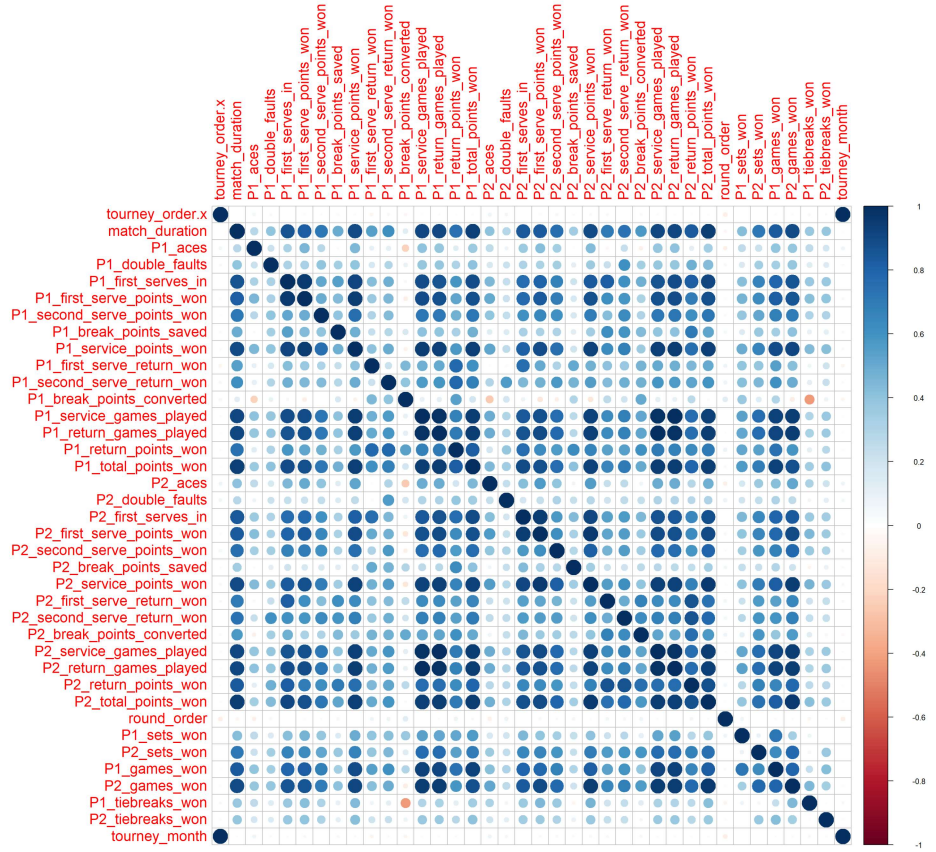

Figure 8 - Correlation Matrix

From the correlation matrix and plot it was observed that around 21 variables were highly correlated with match duration. The variable with the highest correlation coefficient was the Total Point Won, followed by return game played and service game played. Winner breakpoint converted and round order(match round-final, semi1final, 1st round) share the least correlated coefficient with match duration. Below is the table with top 5 variables in terms of correlation coefficient with match duration

Correlation of Variables with Match Duration

rowname	variable	correlation
match_duration	P2_total_points_won	0.95
match_duration	P1_total_points_won	0.94
match_duration	P1_return_games_played	0.91

rowname	variable	correlation
match_duration	P2_service_games_played	0.91
match_duration	P1_service_games_played	0.91
match_duration	P2_return_games_played	0.91
match_duration	P1_service_points_won	0.90
match_duration	P2_service_points_won	0.89
match_duration	P2_games_won	0.89
match_duration	P1_first_serves_in	0.87

Table 8 - Top 5 variables with high correlation coefficient with match duration.

Building Model for Problem Statement 1 - Linear Regression

To initiate the model building process, we will begin by referring to a correlation matrix. Typically, variables that display a strong correlation with the dependent variable are included in the model. In this case, we have identified 21 variables that have a high correlation with match duration. However, we have observed that many of these variables also have a strong correlation with each other. For instance, “return played” and “serve played” are both strongly correlated with “total points” and “first serve in”. Moreover, “set won” is correlated with “game point won”. Given our knowledge of the game and the correlation matrices, we have shortlisted a set of independent variables that will be employed in constructing a linear regression model for predicting match duration.

P1_aces	P2_aces
P1_double_faults	P2_double_faults
P1_first_serves_in	P2_first_serves_in
P1_break_points_saved	P2_break_points_saved
P1_break_points_converted	P2_break_points_converted
P1_total_points_won	P2_total_points_won
P1_games_won	P2_games_won
P1_tiebreaks_won	P2_tiebreaks_won
tourney_conditions	tourney_surface

Table 8 - Independent Variables.

The reason for choosing these variables is stated in the Method of Problem Statement 1

Preparing the dataset based on shortlisted variables

It was observed that the playing surface may played a role in determining the duration of a match, as indicated by the plot of Surface vs. Match duration. In order to incorporate this variable into our analysis, we first need to convert the categorical data into numerical form. This can be done using a technique called one-hot encoding, which transforms the categorical variable into a numerical representation that can be used in our analysis.

```
##One hot encoding for Surface-Hard
Model_data$Hard_Surface <- case_when(Model_data$tourney_surface == "Hard" ~ 1,TRUE ~ 0)
##One hot encoding for Surface-Hard
Model_data$Clay_Surface<- case_when(Model_data$tourney_surface == "Clay" ~ 1, TRUE ~ 0)

##Predicting Match duration only for Hard and Clay thus dropping matches played on glass
Model_data <- Model_data %>% filter(tourney_surface != "Grass")

##Removing the Categorical variable
Model_data <- Model_data %>% dplyr::select(-c(tourney_surface))
```

We have removed grass as we don't have enough data points to grass thus proceed the with checking class balance for Surface Conditions with Hard and clay

Class imbalance check for surface(in %)

Hard	63.41
Clay	36.59

Table 9 - Class imbalance check for hard & clay surface.

Based on the data in the table, it can be seen that there is an overrepresentation of matches played on hard surfaces, accounting for 65% of the total matches. This class imbalance could skew the results and cause the model to place too much emphasis on hard surfaces, leading to biased predictions. To counteract this issue, it is necessary to address the class imbalance by applying techniques that balance the distribution of classes in the dataset. This will ensure that the model is not overly influenced by any particular class and can make accurate predictions based on the relationships between the predictors and the response variable.

To address the issue we will be using undersampling method where we will reduce the size of the majority class(Hard Surface) to match the size of the minority class(Clay Surface).

```
##Checking the number of clay surface observations
Clay <- Model_data %>% filter(Clay_Surface == 1)
Hard <- Model_data %>% filter(Hard_Surface == 1)

##Performing Undersampling on Hard surface
Hard_sample  <- sample_n(Hard,nrow(Clay))

##Class balanced data
Model_data <- rbind(Clay, Hard_sample)

kable(cbind(c("Hard","Clay"),
        rbind(sum(Model_data$Hard_Surface), sum(Model_data$Clay_Surface))),
      align="c",
      caption = "<font><center><b>Class balanced data for surface </b></center>",
      digits = 2) %>%
  kable_styling(bootstrap_options = c("basic", "striped", "bordered"), font_size = 14)
```

Class balanced data for surface

Hard	1176
Clay	1176

```
Model_data<- Model_data %>% dplyr::select(-c("Hard_Surface"))
```

Table 10 - Class imbalance data for hard & clay surface.

Splitting the data to build Linear Regression Model

```
set.seed(123)
trainIndex <- createDataPartition(y = Model_data$match_duration, p=.70, list = FALSE)

# Loading training data (70% from the original dataset):
train <- Model_data[trainIndex, ]

# Loading test data (30% from the original dataset):
test <- Model_data[-trainIndex, ]
```

The dataset is splitted into 70:30 ration for train and test. The model will be built using train dataset whereas it will be tested for its performance on test data

Training the model with train dataset

```
set.seed(123)
##Building Linear Model
Fit <- lm(match_duration ~. , data = train )
summary(Fit)
```

```
##
## Call:
## lm(formula = match_duration ~ ., data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -37.324  -5.625  -0.226   5.080  42.767
##
## Coefficients:
##                            Estimate Std. Error t value             Pr(>|t|)
## (Intercept)                -1.36329    1.19772  -1.138              0.25519
## P1_aces                    -0.44775    0.05810  -7.706   0.0000000000000224 ***
## P1_double_faults           -0.23053    0.12364  -1.865              0.06243 .
## P1_first_serves_in         -0.06581    0.03793  -1.735              0.08293 .
## P1_break_points_saved      -0.03433    0.10995  -0.312              0.75488
## P1_break_points_converted -1.03921    0.70611  -1.472              0.14128
## P1_total_points_won         0.74916    0.06930  10.810 < 0.0000000000000002 ***
## P2_aces                    -0.28510    0.06637  -4.296   0.0000184519303631 ***
## P2_double_faults           -0.47030    0.10937  -4.300   0.0000180734867058 ***
## P2_first_serves_in         -0.08867    0.03665  -2.419              0.01566 *
## P2_break_points_saved      -0.13222    0.10356  -1.277              0.20187
## P2_break_points_converted   1.96955    0.74214   2.654              0.00803 **
## P2_total_points_won         0.62098    0.06375   9.741 < 0.0000000000000002 ***
## P1_games_won                0.70716    0.43276   1.634              0.10244
## P2_games_won                0.13530    0.41891   0.323              0.74675
## P1_tiebreaks_won            0.01066    0.60935   0.017              0.98605
## P2_tiebreaks_won            3.91937    0.99205   3.951   0.0000812177879473 ***
## Clay_Surface                2.90456    0.47648   6.096   0.0000000013555252 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.993 on 1631 degrees of freedom
## Multiple R-squared:  0.9274, Adjusted R-squared:  0.9267
## F-statistic:  1226 on 17 and 1631 DF,  p-value: < 0.00000000000000022
```

The result of a linear model suggest that certain variables had significant impact on the model. These variables include P1_aces, P1_double_faults, P1_total_points_won, P2_aces, P2_double_faults, P2_first_serves_in, P2_break_points_converted, P2_total_points_won, P2_tiebreaks_won, and clay_surface

The model has a high R-squared value of ~93%, indicating a good fit, but it is possible that this high value could be due to multicollinearity, Thus it is important the model for multicollinearity

Source - https://www.statisticssolutions.com/multicollinearity/ (https://www.statisticssolutions.com/multicollinearity/)

Checking multicollinearity

```
##Check VIF in the model for multicollinearity
kable((vif(Fit)),
      align="c",
      caption = "<font><b>Multicollinearity Check</b>",
      digits=2,
      #col.names = "VIF score")
      )%>%
    kable_styling(bootstrap_options = c("basic", "striped", "bordered"), font_size = 12, full_width = FALSE) %>%
   row_spec(c(5,6,11,12,13,14), bold = TRUE, italic = TRUE, background =  "#BBBBBB" )
```

Multicollinearity Check

	x
P1_aces	1.62
P1_double_faults	1.50
P1_first_serves_in	7.40
P1_break_points_saved	2.14
P1_break_points_converted	***24.57***
P1_total_points_won	***48.11***
P2_aces	1.63
P2_double_faults	1.33
P2_first_serves_in	6.96
P2_break_points_saved	1.99
P2_break_points_converted	***21.45***
P2_total_points_won	***52.32***
P1_games_won	***39.23***
P2_games_won	***65.30***
P1_tiebreaks_won	2.20
P2_tiebreaks_won	1.50
Clay_Surface	1.16

Table 11 - Multicollinearity Check.

We have used VIF (Variance Inflation Factor) to a measure of multicollinearity between independent variables in a our model. A rule of thumb for interpreting the variance inflation factor:

- 1 = not correlated.
- Between 1 and 5 = moderately correlated.
- Greater than 10 = highly correlated.

source - https://www.statisticshowto.com/variance-inflation-factor/ (https://www.statisticshowto.com/variance-inflation-factor/)

The results of the model indicate that some variables are highly correlated, namely P1_breaks_points_converted, P1_total_points_won, L_break_points_converted, L_total_points_won, P1_games_won, and L_games_won. This high correlation among them suggests that they are causing multicollinearity in the model. Therefore, it is necessary to remove these variables from the model in order to avoid the issue of multicollinearity.

Removing the highly correlated variables

```
train <- train %>% dplyr::select(c("match_duration",
                                   "P1_aces",
                                   "P1_double_faults",
                                   "P1_break_points_saved",
                                   "P1_total_points_won",
                                    "P2_aces",
                                   "P2_double_faults",
                                   "P2_break_points_saved",
                                   "P2_total_points_won",
                                    "P1_tiebreaks_won",
                                   "P2_tiebreaks_won",
                                   "Clay_Surface"))

test <- test %>% dplyr::select(c("match_duration",
                                   "P1_aces",
                                   "P1_double_faults",
                                   "P1_break_points_saved",
                                   "P1_total_points_won",
                                   "P2_aces",
                                   "P2_double_faults",
                                   "P2_break_points_saved",
                                   "P2_total_points_won",
                                   "P1_tiebreaks_won",
                                   "P2_tiebreaks_won",
                                    "Clay_Surface"))
```

In order to get rid of multicollinearity in our model, we have dropped these variables from the model - P1_breaks_points_converted, P2_break_points_converted, P1_games_won,P2_games_won

```
set.seed(123)
##Fitting the model after removing the multicollinearity
Fit <- lm(match_duration ~. , data = train  )
```

```
##
## Call:
## lm(formula = match_duration ~ ., data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -37.956  -5.605  -0.288   5.252  41.649
##
## Coefficients:
##                       Estimate Std. Error t value             Pr(>|t|)
## (Intercept)           -1.52487    1.10824  -1.376             0.169030
## P1_aces               -0.47743    0.05365  -8.900 < 0.0000000000000002 ***
## P1_double_faults      -0.15188    0.11937  -1.272             0.203435
## P1_break_points_saved -0.11465    0.09356  -1.225             0.220599
## P1_total_points_won    0.69227    0.03492  19.822 < 0.0000000000000002 ***
## P2_aces               -0.32292    0.06332  -5.100        0.00000037937 ***
## P2_double_faults      -0.35261    0.10429  -3.381             0.000739 ***
## P2_break_points_saved -0.28252    0.08621  -3.277             0.001071 **
## P2_total_points_won    0.75570    0.03172  23.827 < 0.0000000000000002 ***
## P1_tiebreaks_won      -0.08531    0.49271  -0.173             0.862558
## P2_tiebreaks_won       2.30858    0.94093   2.453             0.014251 *
## Clay_Surface           2.91218    0.48036   6.063        0.00000000166 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.095 on 1637 degrees of freedom
## Multiple R-squared:  0.9255, Adjusted R-squared:  0.925
## F-statistic:  1849 on 11 and 1637 DF,  p-value: < 0.00000000000000022
```

Multicollinearity Check

	VIF score
P1_aces	1.35
P1_double_faults	1.37
P1_break_points_saved	1.51
P1_total_points_won	11.94
P2_aces	1.45
P2_double_faults	1.18
P2_break_points_saved	1.35
P2_total_points_won	12.66
P1_tiebreaks_won	1.41
P2_tiebreaks_won	1.32
Clay_Surface	1.15

Table 12 - Multicollinearity Check.

From the summary and VIF table it was observed that after dropping the highly correlated columns, all variables except P1_double_faults, P1_break_points_saved, and P1_tiebreaks_won played a significant role in building the model. Despite the high R-square value of the model, which is 92%, the presence of a

strong correlation between P1_points_won and P2_points_won could not be ignored. However, dropping either of these variables would not be an option as it would lead to a loss of information related to the match duration based on the two players involved. However if multicollinearity caused the model to overfit we will address the issue by running a ridge model after completing the process of feature selection.

Checking RMSE value for Train and Test FIT model

```
set.seed(123)
#Loading independent variable for train and test
x_train <- model.matrix(match_duration ~., train)[,-1]
x_test <- model.matrix(match_duration ~., test)[,-1]

#Loading dependent variable for train and test
y_train <- train$match_duration
y_test <- test$match_duration
```

```
set.seed(123)
##Predicting match duration for train data
pred_train <- predict(Fit,newx = x_train)

##Predicting match duration for train data
pred_test <- predict(Fit,newx = x_test)

##Comparing RMSE value
kable(t(cbind(c("Train","Test"),rbind(rmse(y_train, pred_train),
                                      rmse(y_test, pred_test)))),
      align="c",
      caption="<font><b><center>Train & Test Values</b></center>",
      digits=2) %>%
  kable_styling(bootstrap_options = c("basic","striped","bordered"),font_size = 15)
```

Train & Test Values

Train	Test
9.06225449298058	46.8346648564

Table 13 - Match Duration based on Train & Test Data.

The comparison of root mean squared error (RMSE) values between the train and test data revealed a large discrepancy. The RMSE value for the training data was 9 while it was 46 for the test data, indicating that the model was overfitting. To overcome this issue, a method of feature selection is planned to be used in order to remove some columns and prevent overfitting.

```
##Feature Selection through stepwise method
set.seed(123)
Both <- stepAIC(Fit,direction = "both")
Backward <- stepAIC(Fit,direction = "backward")

Both$call
Backward$call
```

The result of the stepwise feature selection method, with a backward direction and both direction was found to have similar results. both techniques suggested dropping P1_tiebreaks_won and P1_breakpoints_saved for a better AIC value

```
##Building best subset model
set.seed(123)
leaps <- regsubsets( match_duration ~ . , data = train)
plot(leaps, scale= "adjr2")
```

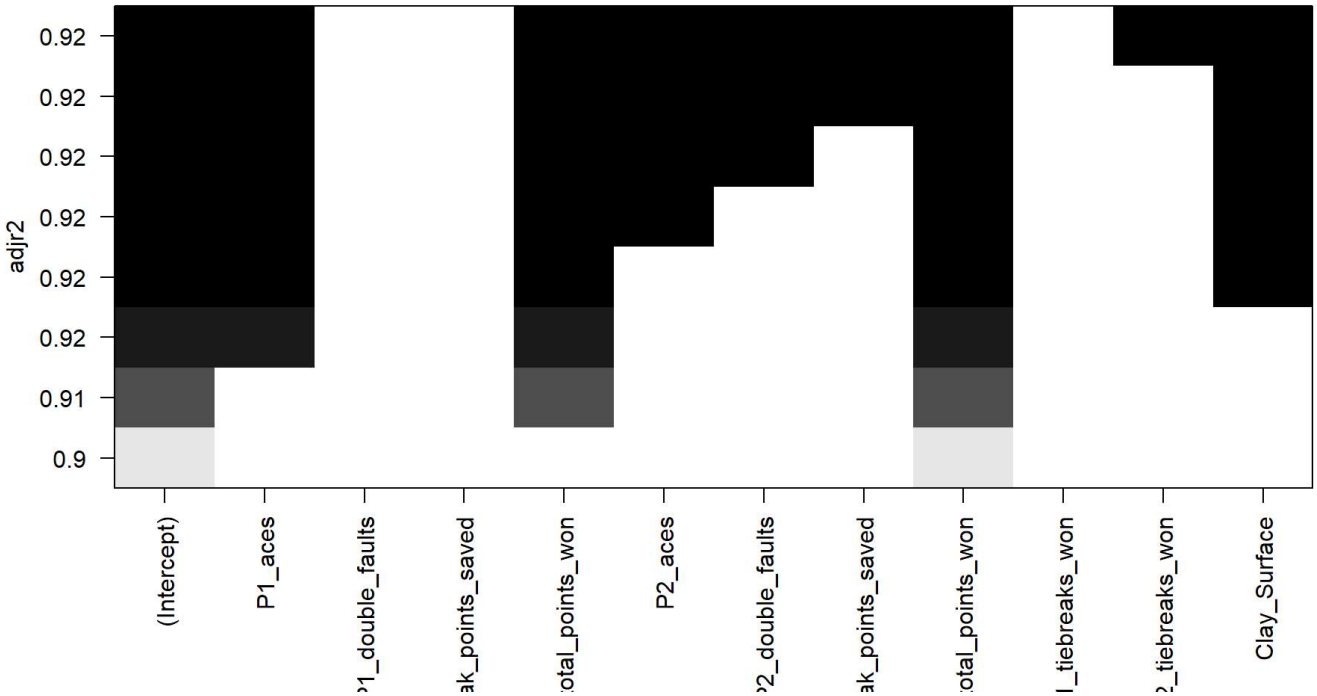

Figure 9 - Best Subset Model

The Best subset selection method suggested to drop P1_tiebreaks_won, P1_breakpoints_saved and P2_breakpoints_saved for better result

Comparison of AIC for Model

Linear Model	11974.8402136178
Feature Selction Backward	11972.3525887664
Feature Selection -Both	11972.3525887664
Best_Subset	11972.4266355047

Table 14 - Comparison of AIC Values

The results from the table suggested that the AIC values were almost the same, so it was expected that the RMSE values would also be similar to the fit model(the initial model). To verify this the backward feature selection model is used to check RMSE for test and train for overfitting evaluation

Train & Test Values

Train	Test
9.0664111550367	46.8316186884522

Table 15 - Comparison of RMSE Values

The feature selection method produced skewed results as the RMSE values for the train data were around 8.7, whereas for the test data, the RMSE was 46. The large difference between these values indicates overfitting, and therefore, it is necessary to run a ridge model to address this issue.

The result of the analysis showed that there is a strong correlation between the player1 total points won and the player2 total points won in the dataset. This has led to multicollinearity, which has caused the model to overfit. To tackle the issue, ridge regression will be applied. This method adds a penalty term to the regression equation, which reduces the coefficients of correlated variables towards zero and helps in reducing overfitting. In this way, we aim to address the overfitting issue in our case.

```
set.seed(123)
##Reading cv.glmnet function to get optimal lamba value through ridge technique
ridge_cv <- cv.glmnet(x_train,y_train,alpha = 0)
```

```
##Lambda.min value
ridge_lambdamin <- ridge_cv$lambda.min

##Lambda.1se
ridge_lambda1se <- ridge_cv$lambda.1se
```

```
##Lamdbamin and Lmabdamax
plot(ridge_cv)
abline(v = log(ridge_lambdamin), col = "turquoise3", lty = 2, lwd = 2)
text(x = log(ridge_lambdamin-0.4), y = 750, label = "lambd.min = 3.17", col = "turquoise3",srt = 90)
abline(v = log(ridge_lambda1se), col = "mediumpurple1", lty = 2, lwd = 2)
text(x = log(ridge_lambda1se+0.5), y = 750, label = "lambd.1se = 4.20", col = "mediumpurple1",srt = 90)
legend("bottomright", legend = c("lambda.min", "lambda.1se"),
       col = c("turquoise3", "mediumpurple1"), lty = c(2, 2), lwd = c(2, 2))
```

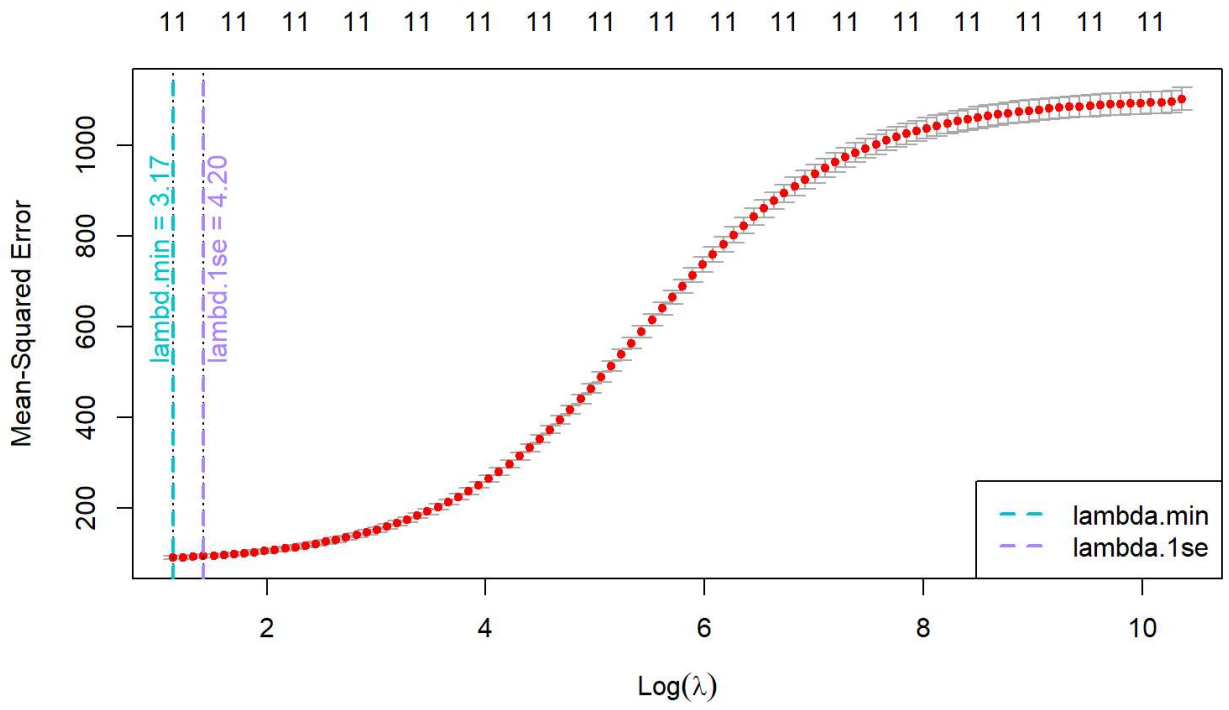

Figure 10 - Ridge Regression

lambdamin of 3.17, it suggests that the regularization parameter value of 3.17 provides gives the minimum cross-validated error among all the lambda values tested. This lambda value represents the model with the best balance between fit and complexity

```
set.seed(123)
##Building Ridge Model with lambda,min
model_ridge_min <- glmnet(x_train, y_train, alpha=0, lambda=ridge_cv$lambda.min)
```

```
Ridge_coef <- data.matrix(coef(model_ridge_min))
colnames(Ridge_coef) <- "Coefficient"

Ridge_coef <- t(apply(Ridge_coef, 1, function(x) rev(sort(x))))


kable(t(Ridge_coef),
      align="c",
      caption="<font><b><center>Train & Test Values</b></center>",
      digits=2) %>%
  kable_styling(bootstrap_options = c("basic","striped","bordered"),font_size = 14) %>%
   column_spec(1, width = "5%") %>% column_spec(2, width = "5%")
```

Train & Test Values

(Intercept)	3.84
P1_aces	-0.33
P1_double_faults	0.17
P1_break_points_saved	0.36
P1_total_points_won	0.65
P2_aces	-0.08
P2_double_faults	-0.18
P2_break_points_saved	0.06
P2_total_points_won	0.61
P1_tiebreaks_won	1.74
P2_tiebreaks_won	5.45
Clay_Surface	3.32

Table 16 - Ridge Coefficients

From the Ridge model it is observed that coefficient for total point won for both the players has reduced down to 0 due to multicollinearity

Checking if the overfitting issue is resolved

```
##Predicting train y through backward selection model and train data
pred_rasso_train <- predict(model_ridge_min,newx = x_train)

##Predicting train y through backward selection model and train data
pred_rasso_test <- predict(model_ridge_min,newx = x_test)
```

```
##Comparing RMSE value
kable(t(cbind(c("Train","Test"),rbind(rmse(y_train, pred_rasso_train),
                                      rmse(y_test, pred_rasso_test)))),
      align="c",
      caption="<font><b><center>Train & Test Values</b></center>",
      digits=2) %>%
  kable_styling(bootstrap_options = c("basic","striped","bordered"),font_size = 15)
```

Train & Test Values

Train	Test
9.3960491702715	9.47662383594537

Table 17 - Comparison of RMSE Values

The RMSE values for train and test are ~9 which is low thus suggesting a good fit. As both the RMSE values are close our overfitting issue is resolved and we can use ridge model for our prediction

Creating Ridge model equation

```
##
##Creating the dataframe to print equation
Ridge_coef <- t(apply(Ridge_coef, 1, function(x) rev(sort(x))))
var_names <- colnames(Ridge_coef)
equation <- paste(round(Ridge_coef,2), "*", var_names, collapse = " + ")
equation <- paste("Match Duration = ", equation)

print(equation)
```

```
## [1] "Match Duration =  5.45 * P2_tiebreaks_won + 3.84 * (Intercept) + 3.32 * Clay_Surface + 1.74 * P1_tiebreaks_w
on + 0.65 * P1_total_points_won + 0.61 * P2_total_points_won + 0.36 * P1_break_points_saved + 0.17 * P1_double_fault
s + 0.06 * P2_break_points_saved + -0.08 * P2_aces + -0.18 * P2_double_faults + -0.33 * P1_aces"
```

Following equation can be used to predict the match duration. We can use this equation by feeding in players pre-match performance data to predict the future match.The coefficients in the equation shows how match duration changes with one unit change of each independent variable

Our intention is to provide these equations to our stakeholders and ask them to enter the till-date average match performance data for players into the equations, which will allow them to make predictions about match duration.

Building Model for Problem Statement 2 - Logistic Regression

In order to predict the winner and loser, the winner columns(P1) from 50% of the matches are extracted and stored as winner data. These matches were randomly selected through sampling. The rest of the matches are used to determine the loser and the columns corresponding to the losers(P2)are picked and

stored as loser data. The losers are assigned a flag of 0, and the winners are assigned a flag of 1. Both the winner and loser data are then combined by appending them together. Finally, this combined dataset is used to understand the factors which will lead to win or lose of the player

```
set.seed(123)
##Selecting winner data
Winner_data <- sample_n(Match_result, 0.5*nrow(Match_result))

##Loser Data
Loser_data <- subset(Match_result, (Match_result$match_id %in% Winner_data$match_id))


##Filtering only winner columns
Winner_data <- Winner_data %>% dplyr::select(c("match_duration",
                                    "P1_aces",
                                    "P1_double_faults",
                                    "P1_first_serves_in",
                                    "P1_break_points_saved",
                                    "P1_break_points_converted",
                                    "P1_total_points_won",
                                    "P1_games_won",
                                    "P1_tiebreaks_won"))

##Removing winner from column name
colnames(Winner_data) <- gsub("^P1_", "", colnames(Winner_data))

##Filtering only loser columns
Loser_data <- Loser_data %>% dplyr::select(c("match_duration",
                                             "P2_aces",
                                    "P2_double_faults",
                                    "P2_first_serves_in",
                                    "P2_break_points_saved",
                                    "P2_break_points_converted",
                                    "P2_total_points_won",
                                    "P2_games_won",
                                    "P2_tiebreaks_won"))
##Removing loser from column name
colnames(Loser_data) <- gsub("^P2_", "", colnames(Loser_data))

##Assigning winner and loser Winner_flag
Winner_data$Winner_flag <- 1
Loser_data$Winner_flag <- 0

##Appending winner and Loser data
Logistic_data <- rbind(Winner_data, Loser_data)
Logistic_data <- Logistic_data %>% dplyr::select(-match_duration)
```

To predict winner or loser of the match below mentioned variables will be used as a independent variable

Variable for Predicting Winner or Loser Check

Independent Variable
aces
double_faults
first_serves_in
break_points_saved

Independent Variable
break_points_converted
total_points_won
games_won
tiebreaks_won

Table 18 - Independent variable for logistic regression

To avoid the bias in the model lets check the class imbalance in dependent variable

Class imbalance check for win and loss

Winner_flag	no_obs
0	1858
1	1858

Table 19 - Class imbalance check for logistic

From the table it is observed that we have balance data as we have equal observations

```
#Loading independent variable for train and test
set.seed(111)
trainIndex <- createDataPartition(y = Logistic_data$Winner_flag, p=.70, list = FALSE)

train_log <- Logistic_data[trainIndex, ]

# Loading test data (30% from the original dataset):
test_log <- Logistic_data[-trainIndex, ]

#Loading independent variable for train and test
x_train_log <- model.matrix(Winner_flag ~., train_log)[,-1]
x_test_log  <- model.matrix(Winner_flag ~., test_log)[,-1]

#Loading dependent variable for train and test
y_train_log <- train_log$Winner_flag
y_test_log <- test_log$Winner_flag
```

To build a logistic regression the data is splitted into 70% train and 30% test

Building logistic regression using glm function

```
model_1og <- glm(Winner_flag~., data = train_log, family=binomial(link="logit"))
summary(model_1og)
```

```
##
## Call:
## glm(formula = Winner_flag ~ ., family = binomial(link = "logit"),
##     data = train_log)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.2730  -0.1786   0.0008   0.2099   5.4978
##
## Coefficients:
##                         Estimate Std. Error z value             Pr(>|z|)
## (Intercept)            -2.630148   0.351241  -7.488   0.0000000000000698 ***
## aces                   -0.002928   0.018621  -0.157                0.875
## double_faults          -0.291734   0.038585  -7.561   0.0000000000000401 ***
## first_serves_in        -0.119640   0.011264 -10.621 < 0.0000000000000002 ***
## break_points_saved      0.208310   0.034136   6.102   0.0000000010450765 ***
## break_points_converted  0.863209   0.075880  11.376 < 0.0000000000000002 ***
## total_points_won       -0.186611   0.014040 -13.291 < 0.0000000000000002 ***
## games_won               1.595660   0.088370  18.057 < 0.0000000000000002 ***
## tiebreaks_won           3.337484   0.231338  14.427 < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3607.14  on 2601  degrees of freedom
## Residual deviance:  985.84  on 2593  degrees of freedom
## AIC: 1003.8
##
## Number of Fisher Scoring iterations: 7
```

In summary, the logistic regression model suggests that all independent variables, with the exception of “aces,” are significant in building the model. The model coefficient shows that a higher number of tie_breaks_won, games_won, break_saved, and converted would increase the chances of a player winning, while a higher number of double faults would result in a low probability of a win. However, the strange finding is that a higher number of first serves in and total points won may lead to low chance of a win.

Using feature selection method on our model to identify the most important features thereby increase the fit of the model and reduce the number of features needed to achieve the desired level of accuracy.

```
Both<- step(model_1og ,direction = "both")
```

The feature selection method suggested to drop ace from the model as it provides better AIC, Thus comparing the AIC below

Comparison of AIC for Model

Logistic Model	1003.8443617714
Feature Selection -Both	1001.86908228407

Table 20 - AIC check for logistic

Based on the information provided in the table, we can conclude that the Feature selection method model has a slightly lower AIC, which suggests a better fit in comparison to the normal model.

Thus we will be removing ace from our model

```
model_1og <- glm(Winner_flag~ double_faults + first_serves_in +
    break_points_saved + break_points_converted + total_points_won +
    games_won + tiebreaks_won, data = train_log, family=binomial(link="logit"))
summary(model_1og)
```

Checking the prediction of our model through confusion matrix

```
##Converting to factor
train_log$Winner_flag <- as.factor(train_log$Winner_flag)
test_log$Winner_flag <- as.factor(test_log$Winner_flag)

##Predicting the win or loss through logistic model and train data
P_train <- predict(model_1og, newdata = train_log, type = "response")
Ptrain_classes <- as.factor(ifelse(P_train >= 0.5, 1,0))

Confusion_Matrix_train <- confusionMatrix(Ptrain_classes, train_log$Winner_flag)
```

```
##           Reference
## Prediction    0    1
##          0 1193   96
##          1  108 1205
```

The results suggest that the model is performing well in terms of accuracy The high number of true positive and true negative values indicates that the model is accurately predicting the outcome for many instances for train data

```
##Predicting the win or loss through logistic model and train data
P_test <- predict(model_1og, newdata = test_log,type = "response")
Ptest_classes <- as.factor(ifelse(P_test >= 0.5, 1,0))

Confusion_Matrix_test <- confusionMatrix(Ptest_classes, test_log$Winner_flag)
```

```
##           Reference
## Prediction   0   1
##          0 528  39
##          1  29 518
```

The test results too imply that the model is exhibiting good accuracy, as evidenced by the high number of true positive and true negative values. This suggests that the model is able to accurately predict the outcome for a significant number of instances in the text data.

```
Accuracy <- Confusion_Matrix_train$overall[1]
Precision <- Confusion_Matrix_train$byClass[5]
Recall <- Confusion_Matrix_train$byClass[6]
Specificity <- Confusion_Matrix_train$byClass[2]
metrics <- data.frame(
  #Metric = c("Accuracy", "Precision", "Recall", "Specificity"),
  Value = c(Accuracy, Precision, Recall, Specificity)
)

##Creating report for test data
Accuracy_test <- Confusion_Matrix_test$overall[1]
Precision_test <- Confusion_Matrix_test$byClass[5]
Recall_test <- Confusion_Matrix_test$byClass[6]
Specificity_test <- Confusion_Matrix_test$byClass[2]
metrics_test <- data.frame(
  #Metric = c("Accuracy", "Precision", "Recall", "Specificity"),
  Value = c(Accuracy_test, Precision_test, Recall_test, Specificity_test)
)

kable(cbind(c("Train", "Test"),rbind(t(metrics),t(metrics_test))),
      caption = "Confusion Matrix Result Comparison",
      digits = 2) %>%
   kable_styling(bootstrap_options = c("striped", "hover","bordered"))
```

Confusion Matrix Result Comparison

		Accuracy	Precision	Recall	Specificity
Value	Train	0.921598770176787	0.925523661753297	0.916986933128363	0.926210607225211
Value	Test	0.938958707360862	0.931216931216931	0.947935368043088	0.929982046678636

Table 21 - Performance Metrics of model

The table displays the performance of a binary classification model on both train and test data, with accuracy, precision, recall, and specificity values reported for both datasets.

The model achieved high accuracy values for both train and test datasets, with the test data accuracy slightly higher than the train data accuracy, suggesting the model is generalizing well. The precision values were also high for both datasets, indicating that the model was good at correctly predicting positive classes. The recall values showed that the model correctly identified a large proportion of positive instances in both datasets.

The specificity values were also high for both datasets, indicating the model was able to correctly identify negative instances.

Overall, the model performed well on both train and test datasets, demonstrating good classification performance.

```
#plotting ROC curve
ROC1 <- roc(test_log$Winner_flag, P_test)
plot(ROC1, print.auc = TRUE, auc.polygon = TRUE, auc.polygon.col = "lightblue", grid = c(0.1, 0.2), xlim = c(1,0), m
ain = "ROC with AUC shaded")
```

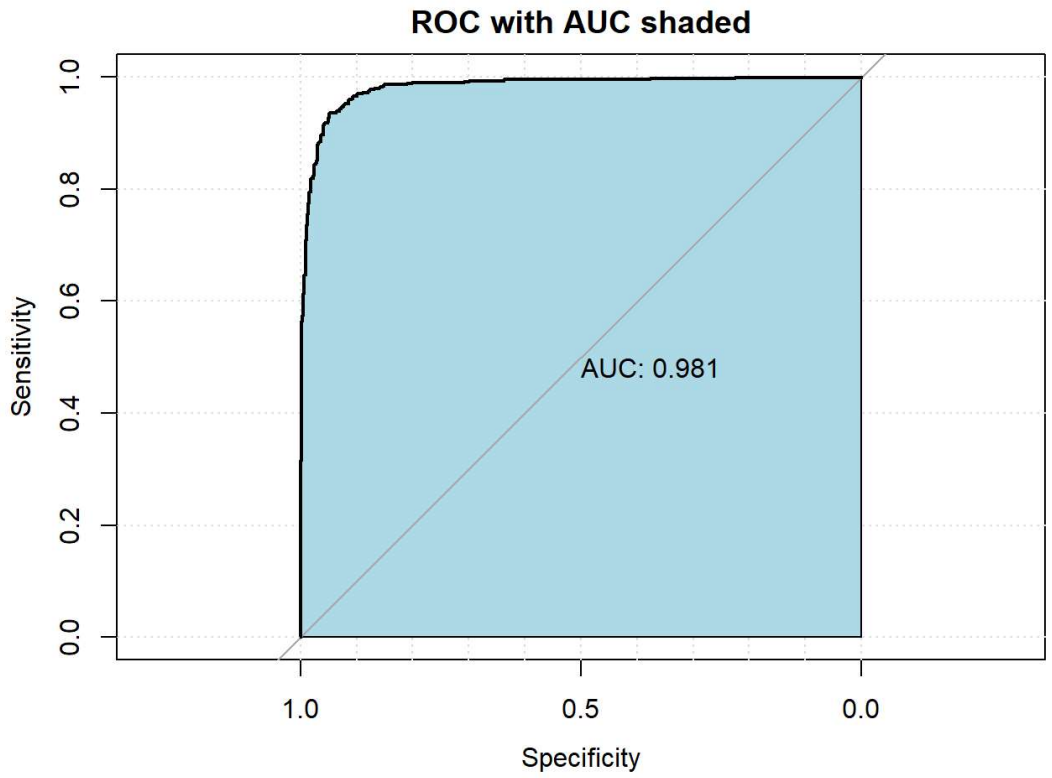

Figure 11 - ROC Curve

The ROC curve is a plot of the true positive rate (TPR) against the false positive rate (FPR). In general, if the curve is close to the top left corner the model has a better fit as it has high specificity associated with high sensitivity. In our model the graph is close to the top left corner thus we can claim our model is better for prediction

AUC is a metric that ranges from 0 to 1, with higher values indicating better performance. An AUC value of 0.988 is close to 1,This suggests that the model is able to differentiate between positive and negative classes with a high degree of accuracy.

Our model shows good performance according to the performance metrics, ROC curve, and AUC, indicating that it can be used for making predictions on new, unseen data. However, before deploying the model, we must validate it using test data to ensure it works well.However, we need to test the model on such data before finalizing it. Currently, the model is fitted with actual outcome results, but we may need to adjust or optimize it after the initial testing.

Conclusion

In conclusion, our analysis showed that the number of tiebreaker points won by the both the players and the surface type (Clay) were crucial in determining the match duration. Matches played on Clay tend to last longer, and if the player wins many tiebreaker points, the match is expected to be longer. On the other hand, if both players have high ace points, the match duration is likely to be slower. With an R-square value of 92%, this model can be used for future predictions of match duration. In terms of predicting the winner, except double fault all the other variables were found to be important factors. A player who wins more tiebreakers and has fewer double faults has a higher chance of winning the match. These models can be applied to future matches by inputting the players' per-match historical performance data in the framework.

References

- Starner.J. (2017,5th January). Regularization Part 1: Ridge (L2) Regression. https://www.youtube.com/watch?v=Q81RR3yKn30 (https://www.youtube.com/watch?v=Q81RR3yKn30)
- Starner.J. (2017,17th January). Regularization Part 2: Lasso (L1) Regression. https://www.youtube.com/watch?v=NGf0voTMlcs (https://www.youtube.com/watch?v=NGf0voTMlcs)
- Starner.J. (2017,29th January). Ridge, Lasso and Elastic-Net Regression in R. https://www.youtube.com/watch?v=ctmNq7Fgbvl (https://www.youtube.com/watch?v=ctmNq7Fgbvl)
- Zach. (2021, 26th August). When to Use Ridge & Lasso Regression. https://www.statology.org/when-to-use-ridge-lasso-regression/ (https://www.statology.org/when-to-use-ridge-lasso-regression/)
- Lee. S. (2021, 3oth October).lambda.min, lambda.1se and Cross Validation in Lasso https://www.r-bloggers.com/2021/10/lambda-min-lambda-1se-and-cross-validation-in-lasso-binomial-response/ (https://www.r-bloggers.com/2021/10/lambda-min-lambda-1se-and-cross-validation-in-lasso-binomial-response/).
- Cho.D. (n.d). Logistic Regression: Feature Selection Methods. https://rpubs.com/ohcsnad/feature_selection_methods (https://rpubs.com/ohcsnad/feature_selection_methods)
- Starmer.J. (2022, 23th January). Logistic Regression in R, Clearly Explained!!!!. https://www.youtube.com/watch?v=C4N3_XJJ-jU (https://www.youtube.com/watch?v=C4N3_XJJ-jU)
- Yihui. (n.d). The kableExtra package. https://bookdown.org/yihui/rmarkdown-cookbook/kableextra.html (https://bookdown.org/yihui/rmarkdown-cookbook/kableextra.html)