STA478 Final Project : Top14 Rugby Analysis

Sakina Lord

2025-12-09

Introduction

The chosen dataset contains rugby match results and player data from the French Top 14 league. Every professional rugby match I have attended in person has been part of this league, which includes Union Bordeaux-Bègles, the team from my host city in France. Compared to more widely studied competitions such as HSBC Sevens or the Six Nations, the Top 14 is relatively underexplored. Personal interest also influenced this choice, as I played rugby union during the summer and it remains my favorite "ball sport."

This project uses two datasets: one containing 18 seasons of French **Top 14 Rugby Union results** , and another with **per-player information**. These datasets are referred to as `matches` and `players` respectively. The third significant dataset used in this analysis is the `joined` dataset which expands the `matches` data so that each row is one player and match combination, and joins this expanded dataset with `players` to add that individual's play statistics.

Data evaluation

Discussion of Variables

Top 14 The Top 14 dataset contains per-game data for 3,338 matches over 18 seasons involving 26 teams. In the raw dataset, all variables are character type. During cleaning, I converted *home_score* and *away_score* to numeric types as well as factored all the remaining character types. I also created:

- *delayed_game* — 1 if the match was postponed due to COVID-19, 0 otherwise
- *score_diff* — the score difference, positive if the home team won, negative if the away team won
- *winning_team* — indicates which team won

Additional key variables include *Season* (the two-year season span), *date* (match day), *home_team* and *away_team*, and *home_1-23* and *away_1-23* (the players for each team in that match).

Top 14 seasons are played over weekends denoted as *day* J1 through J26, with each team having 13 home and 13 away games. The top six teams progress to the final phase, where 3rd-6th place teams compete in *Barrages*. The winner of the 3rd/6th place match faces the 2nd-ranked team, and the winner of the 4th/5th place match faces the 1st-ranked team in the *Semi*, both played at neutral venues. The *Finale* is contested between the winners of the semifinals at a neutral stadium, which may affect how the model interprets home and away designations. The *Access Match* pits the 13th-ranked team against the second division champion for a place in the next Top 14 season. These differences in tournament days were important later in calculating Elo scores for each team.

Players The Players dataset contains per-player information, including *birthdate*, *position*, and *team* for a given *competition* and *year*. It also tracks performance metrics such as *total_points_scored*, *matches_played*, *tries_scored*, and *matches_started*.

Data originates from **itsrugby.fr**, so I was able to compare the short column names in Kaggle and rename fields with something more descriptive. *player_id*, scraped from each player's URL, was cleaned to contain only the player name in uppercase, aligning it with the Top 14 dataset when creating the `joined` set. `joined` contains:

- Character variables which were factored: *position*, *team*
- Character variables which were removed due to redundancy: *player_id*, *birthdate*, *competition*
- Numeric variables: all performance stats and *birth_year* obtained from *birthdate*

Players data aggregates per-competition metrics, whereas the Top 14 dataset contains per-game statistics. This motivates a potential join between the two datasets. Before filtering for Top 14 matches, the dataset included 5,968 players across 26 years and 86 competitions, totaling 51,211 rows. Some entries contain erroneous data, such as a player listed in a 2097 competition (*player_id* thomas-lainault-3896).

Data Cleaning

A key decision was whether to use only Top14 player data or include matches from other leagues. I decided to only used data on the Top14 league so I could focus on match data as well. Filtering `players` for only the Top14 competition reduces the player dataset from 51,211 entries to 4,355, removing players from Irish, English, Welsh, and Scottish leagues such as the Six Nations. When converting scores to numeric type, I encountered "Delayed" and "TBD" values. TBD scores correspond to future matches and are removed, as were the "Delayed" matches because they were not important to the analysis.

A major challenge in merging the datasets is that many Top 14 players do not appear in the Players dataset. The raw datasets only share player names as identifiers, but in different formats. Top 14 contains over 4,000 unique players, while the cleaned Players dataset contains only 1,048, leaving individual statistics missing for 3,154 players. This gap exists across all seasons from 2005–2022, so truncating by season is not an option. Missing two-thirds of player data presents a substantial limitation if player stats are used for modeling. In the final `joined` set, 102,854 player/match combinations had completely missing player statistics data.

Methods

Due to the large number of missing data points, a majority of the work in this project was spent researching and implementing imputation. I used the `mice` package which does Multivariate Imputation by Chained Equations by way of a Gibbs sampler. The`mice` package contains within it multiple methods of imputation depending on the type of missing data, such as whether it is binary, an ordered or unordered categorical, or numeric, and the distribution of the nonmissing data of that variable. The imputation in this analysis is primarily predictive mean modeling which takes samples from the data to build a subset of complete values whose response matches that of the observations with missing values. It then randomly selects a value from this subset to fill the missing value in the target observation. Predictive mean matching (`pmm`) is effective for a wide range of data and, since it is based on data already present in the set, will not produce imputations outside the observed data range which can be both good in that the values are plausible and bad in that the values may miss an important set of values that might lie outside the range. Thus, predictive mean matching is not appropriate for values MNAR (Missing Not at Random) where the missingness is related to the unobserved data. I was able to use it in this analysis as my values are missing completely at random (MCAR) as the missingness was not related to the missing values; they were missing due to simply not being chosen in the scrape when the data was collected.

I first tested a fandom forest versus predictive mean matching approach on the five missing *birth_year* values in the `players` dataset and found that the random forest approach gave imputations whose distribution better matched that of the available data.

After diagnosing the missingness and looking at the distributions and types of available data to select the method of imputation in the `joined` dataset, I determined `pmm` to be the most appropriate method for most of my numerical data. The categorical variables *team* and *position* were imputed using a random forest and Polytomous logistic regression, respectively. Polytomous logistic regression is used for categorical variables with more than 3 levels (*position* has 10) and is essentially a multinomial logistic regression. Random Forest imputation was said to do better than polytomous logistic regression for categories with a large number of levels and low sparsity; *teams* has 26 levels and is full in the available data.

After imputing the data to fill in the 102,854 rows with missing data, I created a single tree to try to predict the score difference using `joined_complete` which is the`joined` data completed using the imputed values.

Analysis results

Cross validation was used to select the predictive model as well as tune all final models. The Elastic Net model to predict `home_win` was tuned internally using `cv.glmnet()` with an α of 0.5. This cross validation found the accuracy of predictions to be ~ 73%, as was also the case for LASSO, Ridge, and Random forest, but their accuracies were a little lower. Though their coefficients were small, the imputed values aggregated to be used as per-match team statistics were present in the Elastic Net model, as seen in the table of Elastic Net Coefficients. The Random Forest predictive model was selected for the highest accuracy in the initial 5x2 K-fold cross validation and was later tuned to select `mtry` and `ntree` using 5-fold cross validation across a grid of 4 potential `mtry` values and 10 potential forest sizes. Tuning these values brought the average predictive accuracy of the model from just over 73% using 4 predictors and a forest size of 5000 to ~74.8% as seen in the Accuracy table. The other tree-based models in the initial model selection cross validation models showed higher variance across folds than the `glmnet`based models which might be due to the high number of predictors when a large amount of the data are factors with many levels.

Apart from the predictive Random Forest, tree-based models did not do very well throughout this analysis. The single trees produced relied solely on one predictor, as seen in and had few terminal nodes which pooled many potential outcomes, resulting likely in high error rates as seen in figures 5 and 7. The pruned `home_wins` prediction tree in fig 15 only contains one split based on the probability of a home win calculated using the Elo values of that match. The imputation based on random forests, however did well as can be seen in the Observed vs Imputed: `birth_year` in the difference of how the imputed `birth_year` values using the `rf`method follows the original data distribution better than those produced using `pmm`.

Discussion of final models and analysis.

Predictive Model: Random Forest

To choose a predictive model, I ran 10 iterations of 5x2 K-fold cross validation on nine different possible models to predict the binary outcome of whether or not the home team will win. The models analysed included GLM, LASSO, Ridge, Elastic Net, Single Tree, Random Forest, BAgged Random Forest, Boosted Random Forest, and BART. For this analysis, I added in an Elo score for each team calculated using all seasons leading up to a match. For each match, I added the probability of a home win calculated using the home team and away team Elo scores as well as incorporated aggregate team stats including the average birth year and total player statistics (*matches_played*, *tries_scored*, etc.) for each team's per-match lineup.

Based on the K-fold cross validation, the regular Random Forest model performed with the highest accuracy. The Elastic Net model was not far behind. Further improvements were be made to the random forest to tune the number of predictors and number of trees hyperparameters to further improve the model. This tuning

revealed that the ideal number of predictor variables to choose from at each split, `mtry`, is 10 predictors and the ideal forest size, `ntree`, is 9500 .

Inferential Model: Single Tree

For my inferential model, I initially built a single tree to predict *score_diff* from the `joined_complete`. The most complex tree produced is seen in fig 5. It is only split on the different teams and did not use any other predictor. Upon cross validation, it was found that a tree of size 6, the most complex tree, was the best predictor of *score_diff*. A pruned tree of 3 is in fig 7 and is shown to have a larger MSE than the more complex tree. A major limitation of this model is the large bins to predict score difference itself because a large set of teams will result in the same *score_diff* even if the teams are on different ends of values that average to the mean *score_diff* at their terminal node. However, this model does help to show which teams tend to have the largest gap in score. The values at the terminal nodes can help to inform which teams have historically been stronger or weaker than their opponents as well as to what magnitude.

I inadvertently created another inferential model when building the predictive model. I did not expect the Elastic Net model to do almost as well as a Random Forest in prediction of `home_win`. I included it in the cross validation to potentially see how much more a random forest would improve upon the GLM options. The most important variables in the Elastic Net model was the intercept and the probability of a home win, calculated using the Elo score. The intercept is the most significant coefficient, which could indicate a baseline higher probability of a home win. This makes sense given the tendance of the data towards home wins which indicates a home advantage. Based on this model, home teams tend to win the most in day 4 and both Narbonne and CA Brive win much more often on their own turf.

Conclusions

In this project, the most important lessons came from managing extensive missingness and learning how to perform large-scale imputation. The initial knit of the document took nearly four hours, driven by repeated MICE imputations and multiple rounds of cross-validation across complex models. Through this process, I gained a deeper understanding of how R handles factored variables, particularly when interpreting the largest coefficients in the Ridge and Elastic Net models. I also saw firsthand that modeling offers multiple pathways to similar goals, whether evaluating predictor importance, tuning hyperparameters, or selecting among competing model classes.

Working with datasets containing substantial sparsity proved especially challenging. In the `players` dataset, where events such as red cards or kick conversions are rare, imputation tended to oversmooth these patterns, filling in zeros where real variability should exist. The `joined` dataset—where roughly 70% of rows lacked multiple fields which seemed almost impossible to impute, and to some extent that concern was justified. I worried the resulting values would be unusable, but diagnostic checks suggested that at least some structure was preserved. Even so, imputation at this scale likely alters the data in ways that limit inferential validity and predictive reliability. The fact that the best models still achieved accuracy above 70% was surprising and somewhat reassuring, even if that performance is modest given the uncertainty introduced by so much estimated data.

In the future, I would like to incorporate additional methods such as VIF analysis and boosted random forest models. I initially considered VIF when examining the missingness mechanism of `birth_year`, but cross-validation ultimately made the Ridge penalty selection straightforward without it. Still, VIF could have clarified multicollinearity among predictors in the Elastic Net model, possibly improving its performance. Boosted models may also provide better predictive power and more nuanced variable importance measures. Overall, this project highlighted both the limitations and the unexpected strengths of modeling in the presence of extreme missingness.

Sources

Sas, W. (21 July, 2025). *Elo Calculator*. Omni Calculator https://www.omnicalculator.com/sports/elo

Van Buuren, S. (2018). *Flexible Imputation of Missing Data*. https://stefvanbuuren.name/fimd/sec-pmm.html

Van Buuren, S. (27 May, 2025). *Multivariate Imputation by Chained Equations.* https://cran.r-project.org/web/packages/mice/refman/mice.html#mice

Enders, C. (2024). *Modern Missing Data Analysis*. CenterStat. [Lecture Notes] https://centerstat.org/wp-content/uploads/2024/04/MISS-Notes.pdf

ISLR Version 2

Graphics Appendix

Imputation

There is a large amount of data missing in `Joined` (102,854 rows worth!) as well as 5 `birth_year` values in `players`.

Players: Imputing a small set (5 missing data points) Using Random Forest

Diagnose Missingness

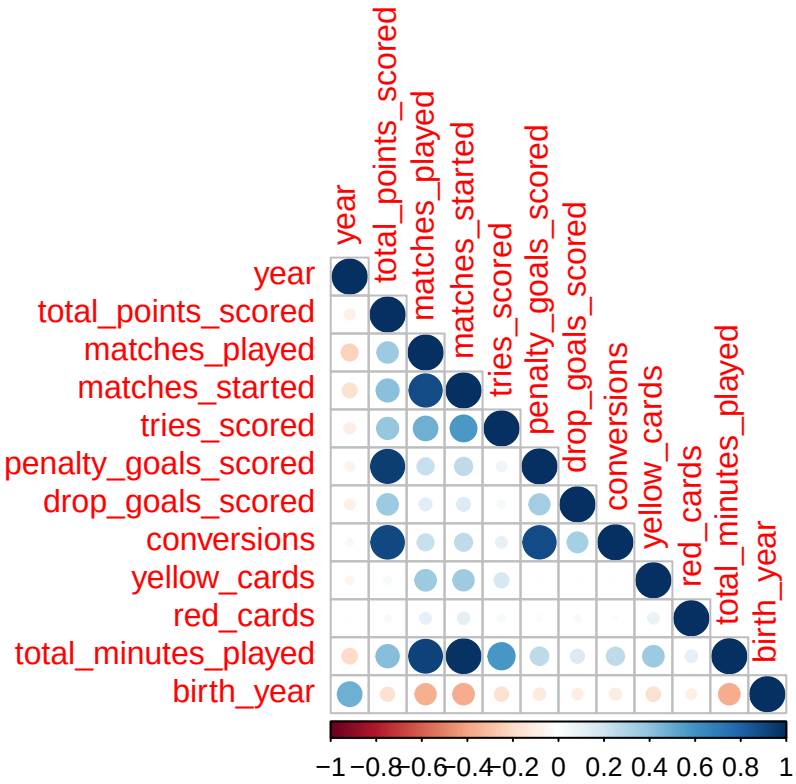


Figure 1: Players Corrplot

There a lot of correlation between multiple factors, which makes sense. The time spent on the field is directly related to how many games a player is in or starts. Players also cannot make points off the field. Conversions, penalty goals, drop goals, and tries are all ways to score, so it follows that they are related to the total points scored.

```
##
##  Pearson's Chi-squared test with simulated p-value (based on 2000
##  replicates)
##
## data:  table(players_factors$position, players_factors$team)
## X-squared = 337.04, df = NA, p-value = 0.006497
```

```
##
##  Pearson's Chi-squared test with simulated p-value (based on 2000
##  replicates)
##
## data:  table(players_factors$position, players_factors$player_name)
## X-squared = 39195, df = NA, p-value = 0.0004998
```

```
##
##  Pearson's Chi-squared test with simulated p-value (based on 2000
##  replicates)
##
## data:  table(players_factors$team, players_factors$player_name)
## X-squared = 59081, df = NA, p-value = 0.0004998
```

Based on these chi-sq tests on the bivariate relationships between factored variables, player name, position, and team are highly correlated. The p-value is very low, suggesting there is evidence these values are not unrelated and are more correlated factors than not. All this correlation indicates a ridge or elastic net regression approach might be best for diagnosing the type of missingness. A model based on all other predictors to try and predict the birth year of a player will show how much those predictors are used in predicting our response: whether or not `birth_year` is missing.

```
## Best Ridge Lambda: 0.00214
```

```
## Best Elastic Net Lambda: 0.00214
```

Both Ridge and Elastic Net have small lambdas, indicating they penalize the model very little. The regression is likely very similar to its least squares counterpart. However, there is a lot of multicollinearity, so this still might be an overfit model. This is not of much concern because we want to look at how well the other predictors predict whether `birth_year` is missing, so the focus will be on the significance of the predictor coefficients.

Looking at Ridge and Elastic Net model coefficients

Since the coefficients are very small, MAR (Missing at Random) is not plausible. The other predictors are not strong indicators of `birth_year`, so imputation will take the assumption of either MCAR (Missing Completely At Random) or MNAR (Missing Not at Random). The type of missingness helps to determine what values we choose to use to impute a value, what type of imputation, and how we interpret imputed values. In MCAR, Missingness is unrelated to any other variables meaning dropping these values from the set will not create bias in the dataset. Wih MNAR, the missingness depends on the unobserved data itself (here, that is `birth_year`) and standard imputation will create biased results. With MAR, missingness depends on observed variables, not the missing values themselves so imputation using the other variables works and standard imputation will not give biased results. The missing `birth_year` are likely MCAR as there is no motivation for why these particular players have not reported their dates of birth, so we could just drop the values without introducing bias. However, this small set might be cool to try some small imputation on.

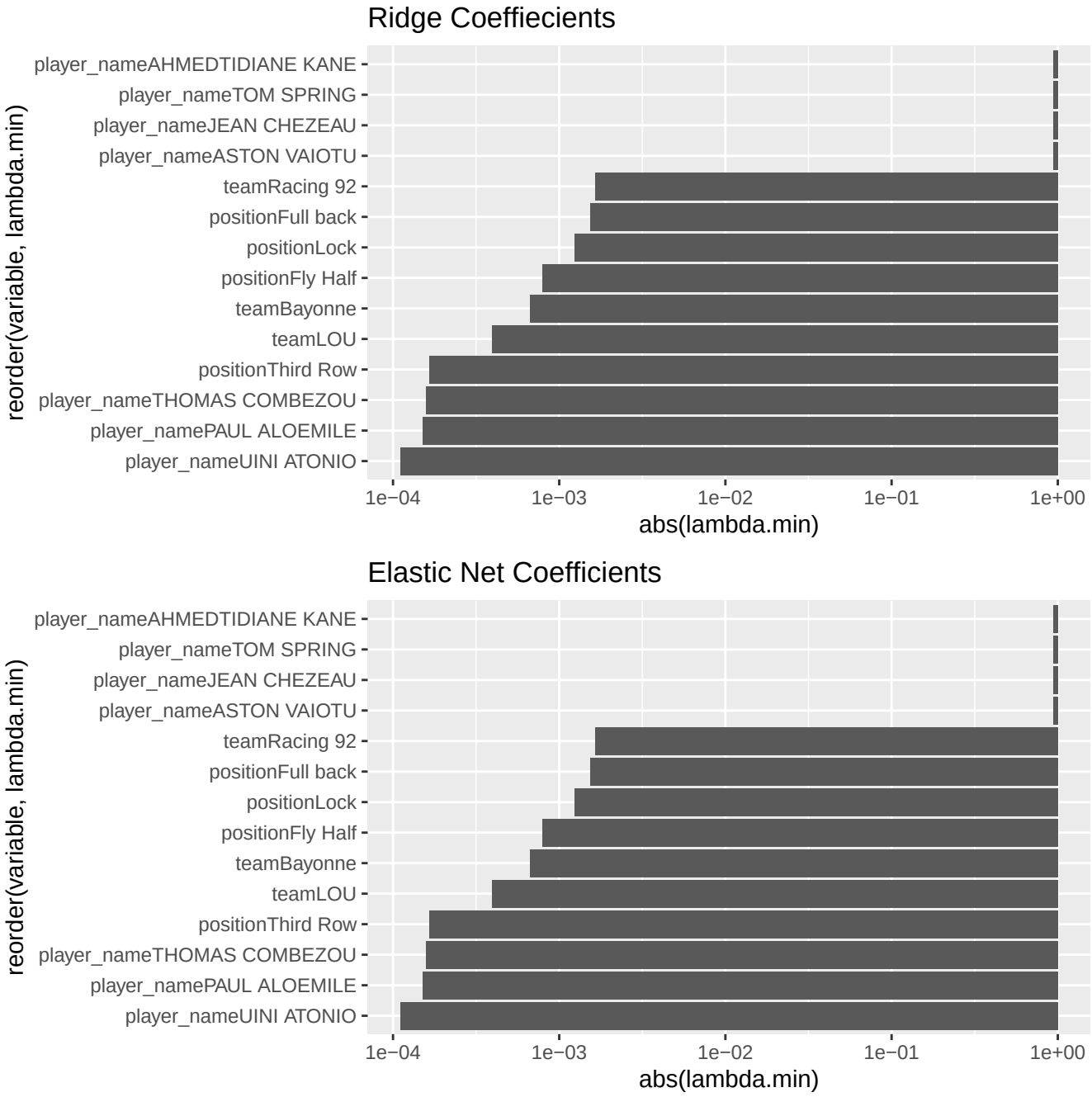


Figure 2: Coefficients greater than 10^{-4} of Ridge and Elastic net models predicting missing birth year. Log scaled

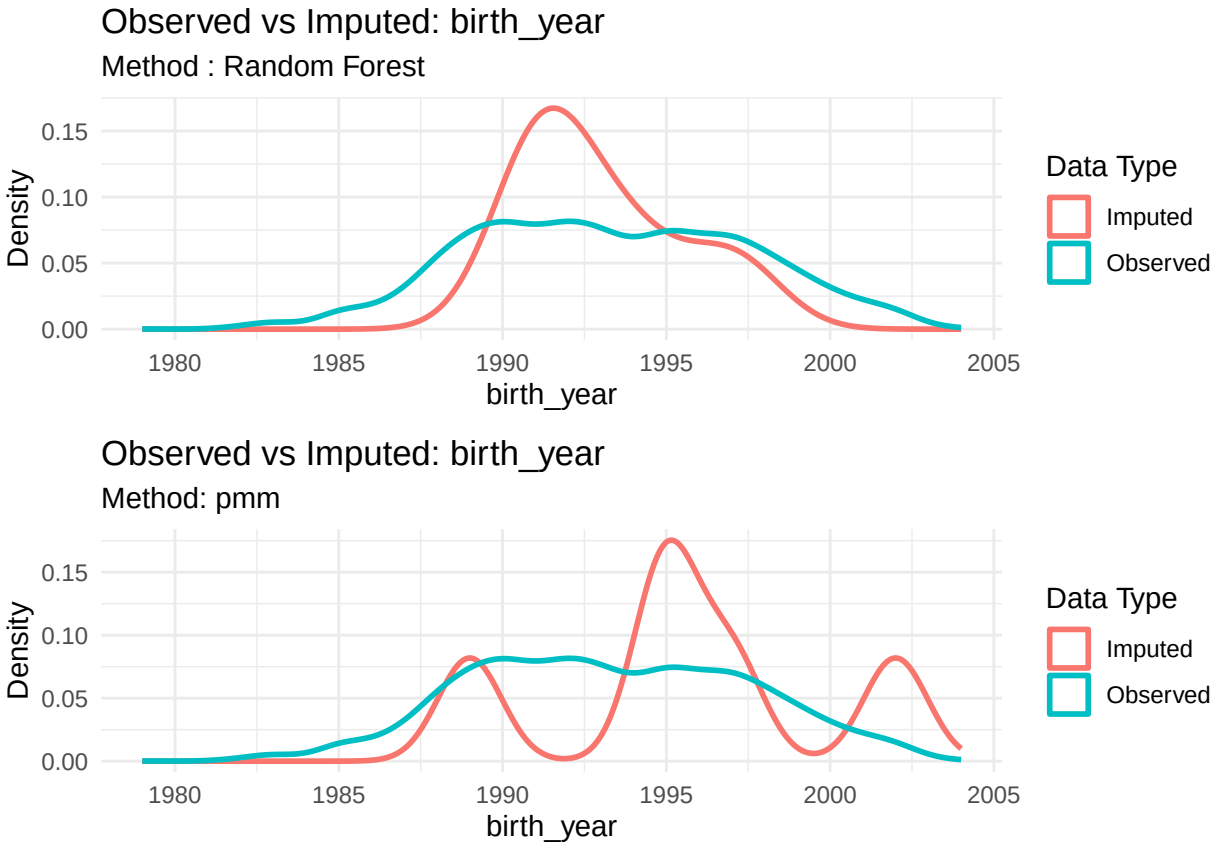
Imputation using package MICE: Multiple Imputation by Chained Equations

Found using ChatGPT. Documentation here: https://cran.r-project.org/web/packages/mice/refman/mice.html

Mice uses the Gibbs sampler to perform multiple imputation using a specified modelling method. Here we will try and compare the `pmm` (Predictive mean matching) and `rf` (Random Forest) imputation methods.

Distribution Analysis

Taking a peek at the original versus the imputed data, the distributions are in relatively similar areas. The shapes are unlikely to be very similar due to the low number of imputed data, but they do rise in similar regions.

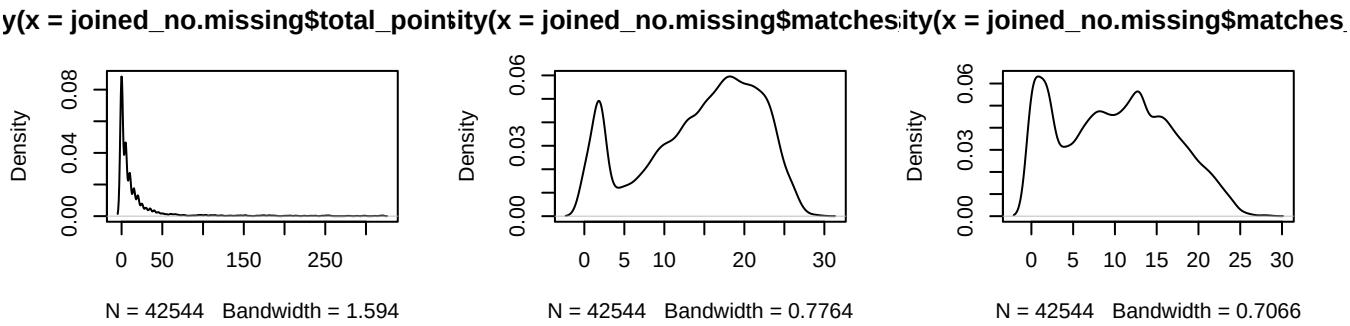


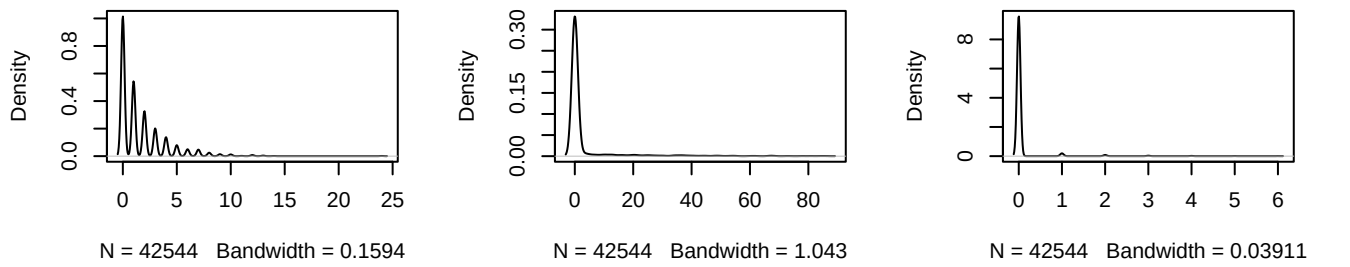
The Random Forest method seems to be a better estimator of values than those from the Predictive Mean Matching method. A complete `players` dataset can be made by taking the current `players` set and filling the missing `birth_year` values in with these imputed values from the Random Forest.

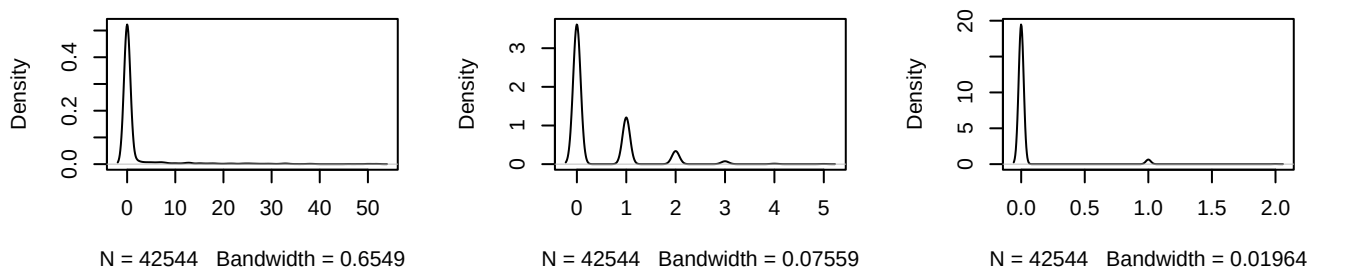
On Joined Data

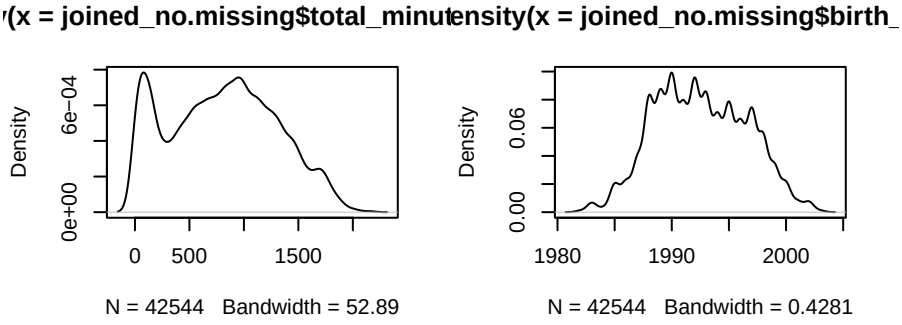
After joining the `Players` and `Match` data, I was missing player statistics for 102,854 different player/year combinations, in addition to the extra birth years missing from `players`. This is a much larger impute, considering this makes up 70.74% of observations Since there is such a large amount missing, I expect the imputed values to not be very accurate individually, but I hope they give a good description of the data globally.

These distributions, in addition to the data type of each predictor, will inform what type of imputation is used.





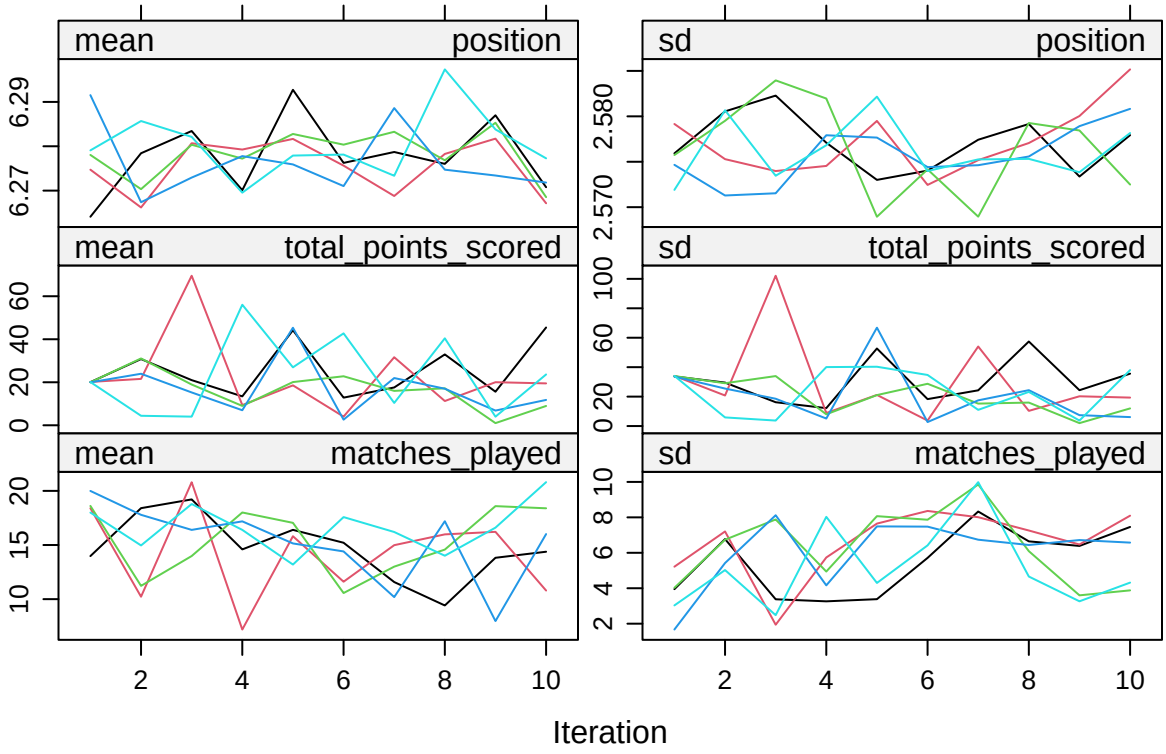


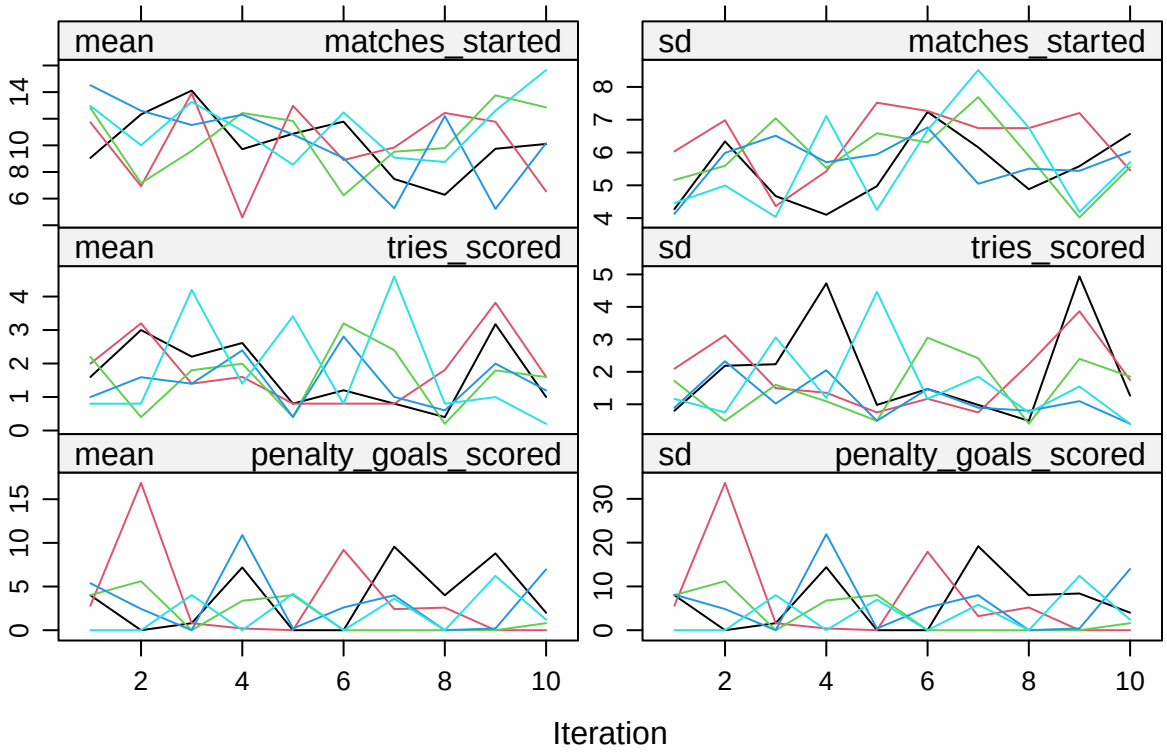


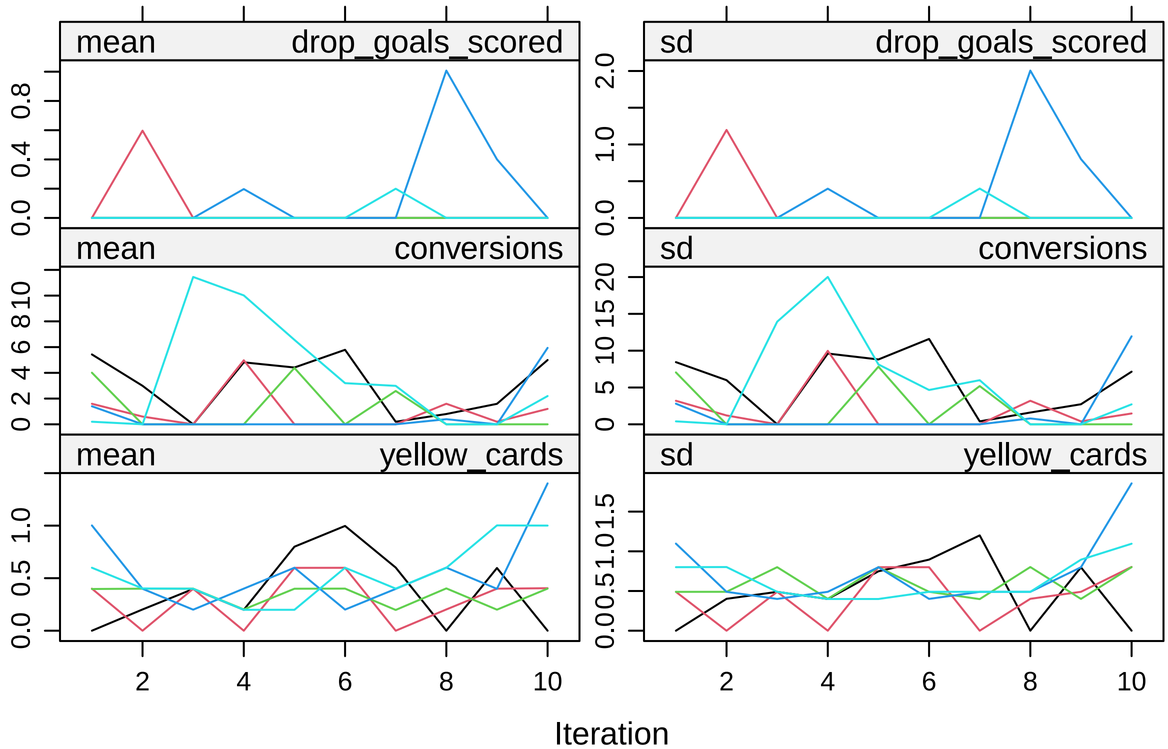
Based on these distributions, the majority of predictors will be filled using `pmm` because they are not normally distributed and numeric. The most normal-ish distribution is seen in `birth_year`, so `norm` (Bayesian linear regression) is used. Position is a categorical variable with 10 levels and will be imputed using `polyreg` (Polytomous logistic regression) which is a multinomial model. Team is also categorical, but has a 26 levels which benefits more from a `rf` (Random Forest) approach.

Validation

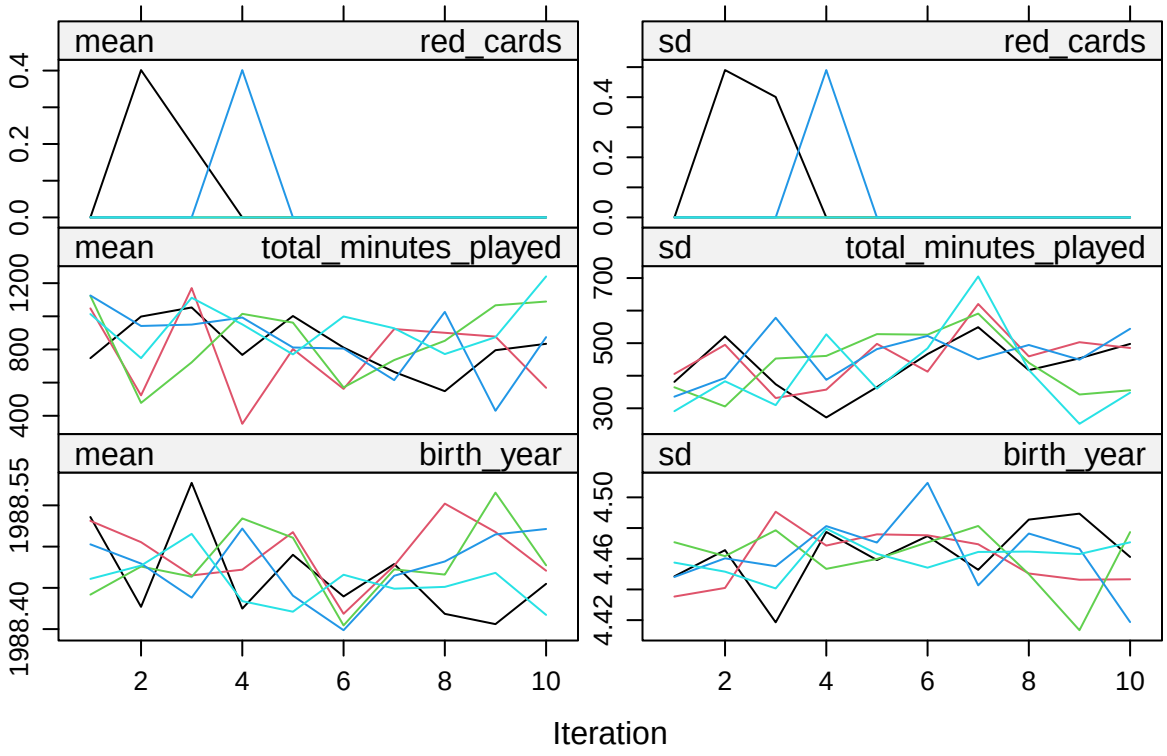
Mice uses Gibbs sampling, so one method of assessing the imputations is to see if the chains converged.




mean
matches_started
sd
matches_started
mean
tries_scored
sd
tries_scored
mean
penalty_goals_scored
sd
penalty_goals_scored
Iteration

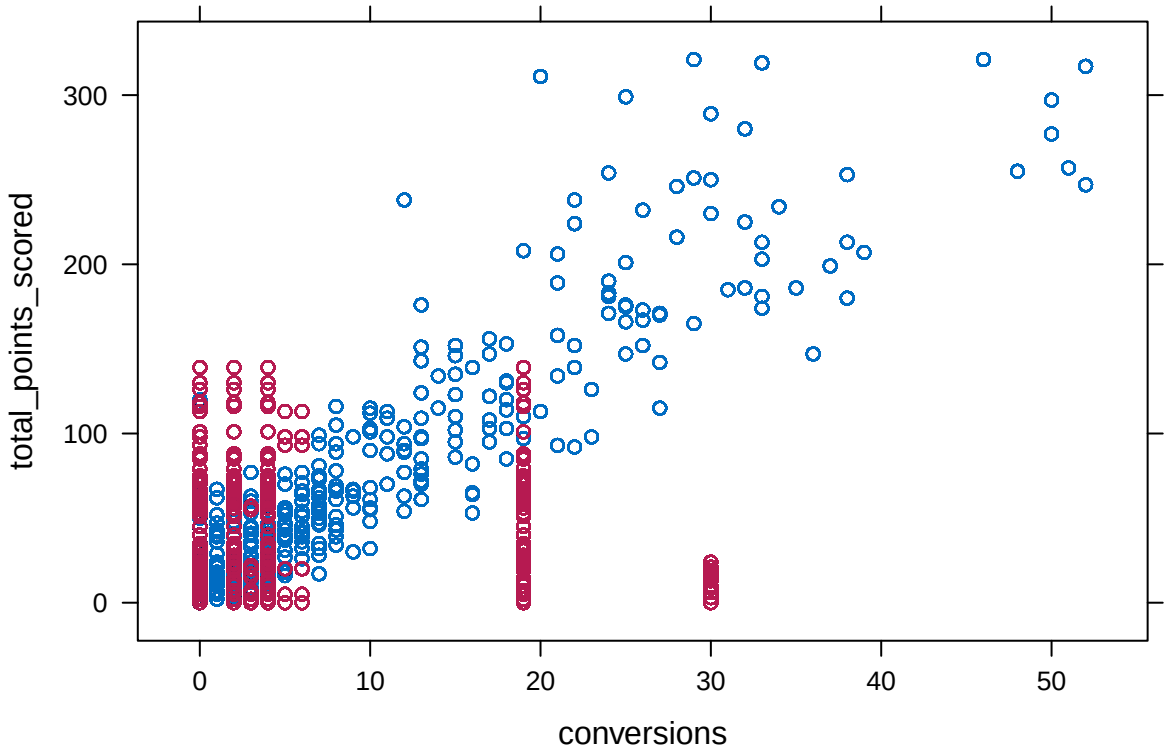


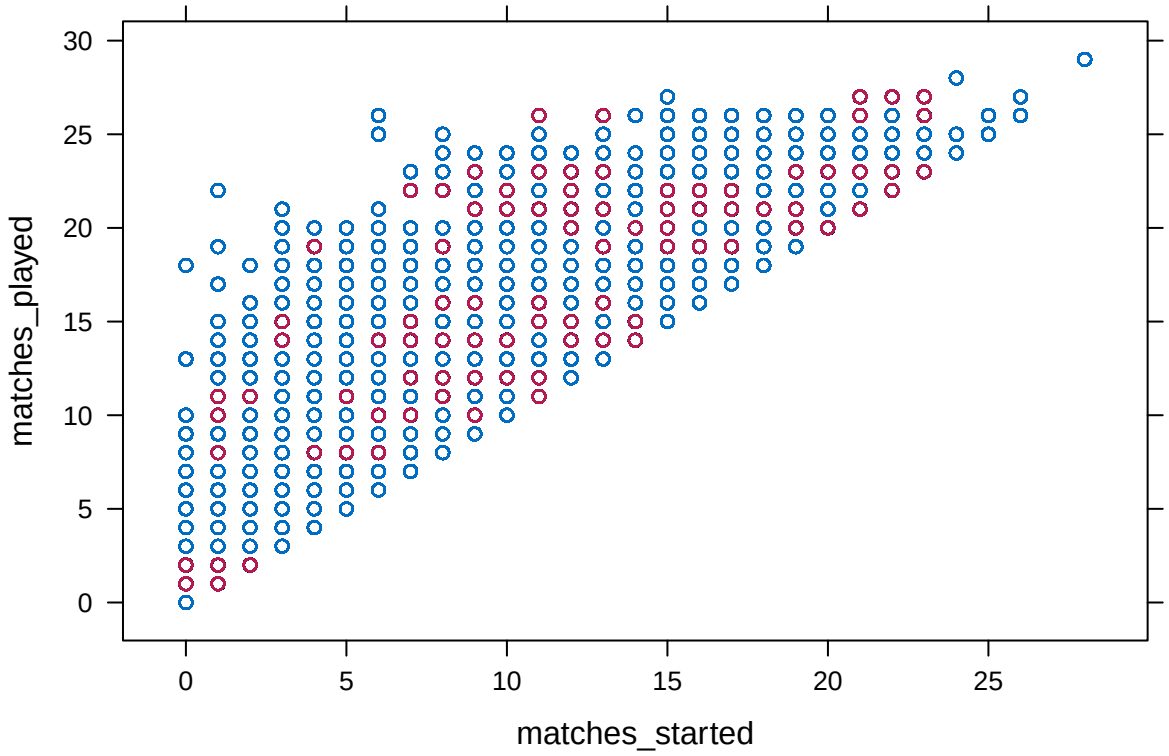
mean
drop_goals_scored
sd
drop_goals_scored
mean
conversions
sd
conversions
mean
yellow_cards
sd
yellow_cards
Iteration

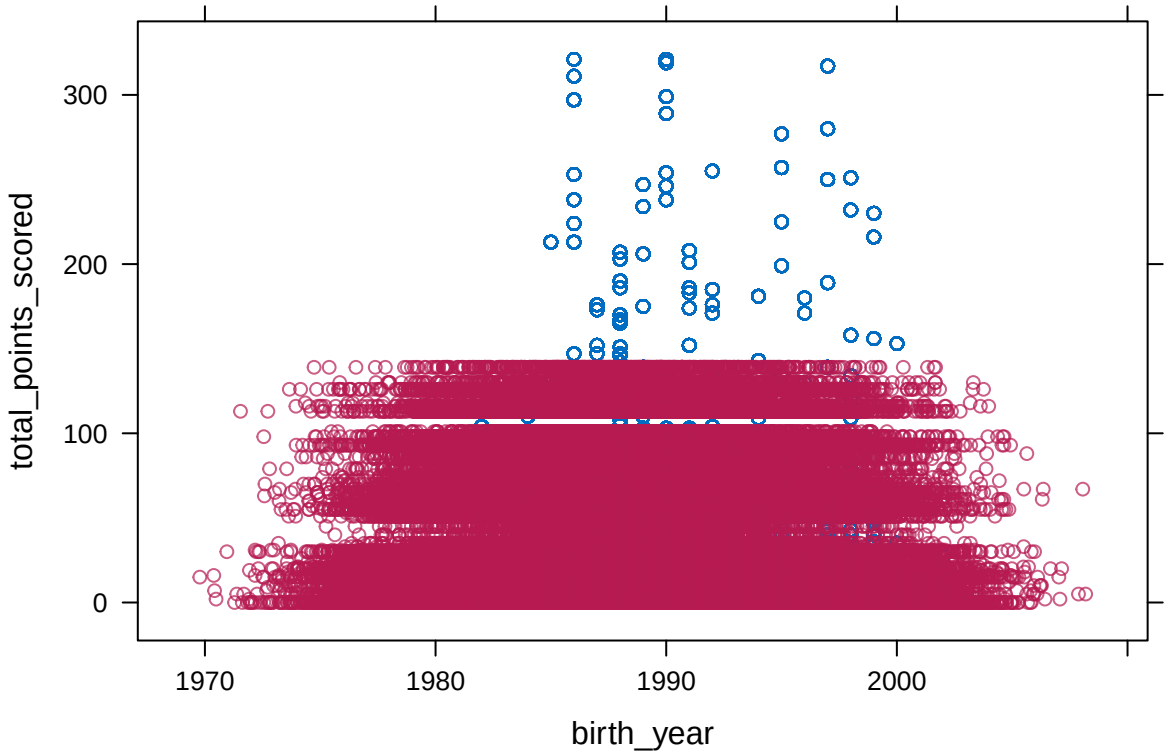


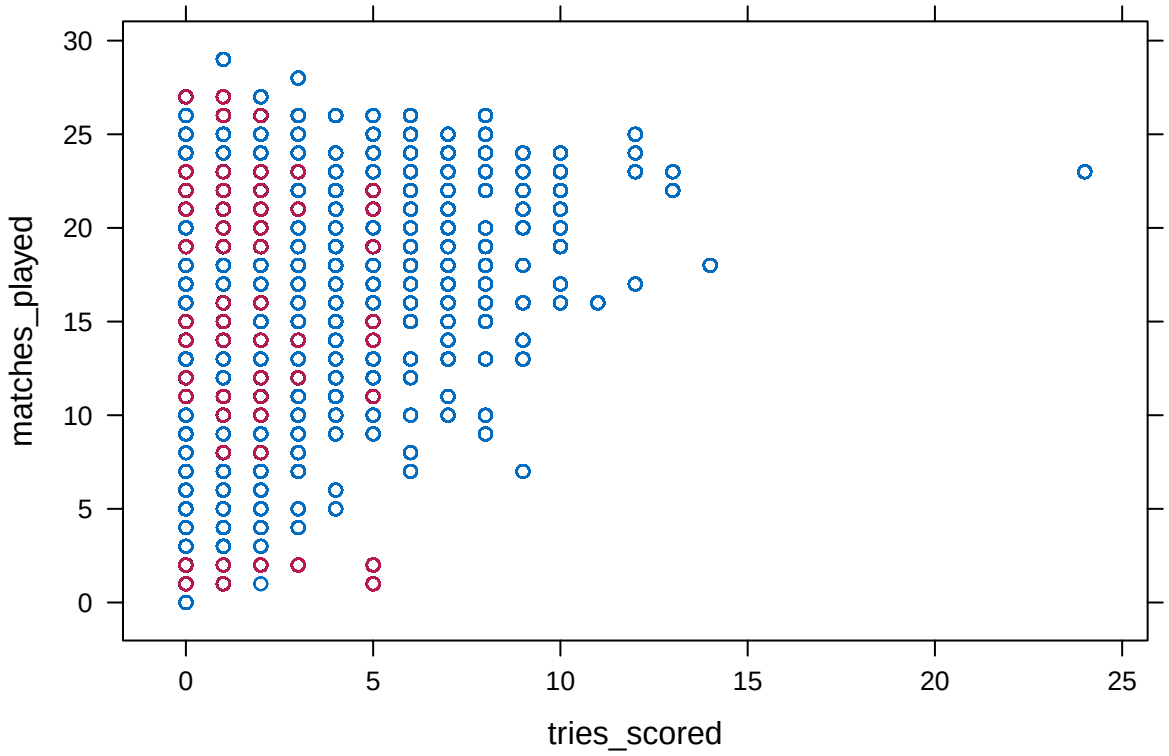
Most all of the chains mix well and are not getting stuck. Some exceptions are those for `red_cards` and `drop_goals_scored` which may be due to the high number of zeroes in the data. Drop goals are not a common way to score in Rugby and red cards are not desired because they limit play; it makes sense that these values would be zero for most players.

Another way to assess the 'goodness' of the imputations is to check that relationships between variables are maintained. Here are a few of the bivariate relationships assessed.




matches_played
matches_started
0
5
10
15
20
25
30


total_points_scored
300
200
100
0
1970
1980
1990
2000
birth_year



In these graphs, well-imputed data, in red, is will be mixed well into the original data, the blue, with a similar relationship between the rpedictors and no impossible values. This idicaties the imputed values maintain the bivariate relationship. `matches_started` versus `matches_played` has the imputed values mixed in well while `total_points_scored` and `conversions` they do not, indicating imputed values may have introduced bias or created misleading data. I continue with the data in the set because I would like to try predictions, even if 70% of the data is hypothetical and not all of it is great.

Look at change in Correlation structure

The initial correlations are not much different than that of the dataset completed using the imputed data. This also helps show where predictor relationships were maintained. The imputed values look good here since the correlation structure is similar.

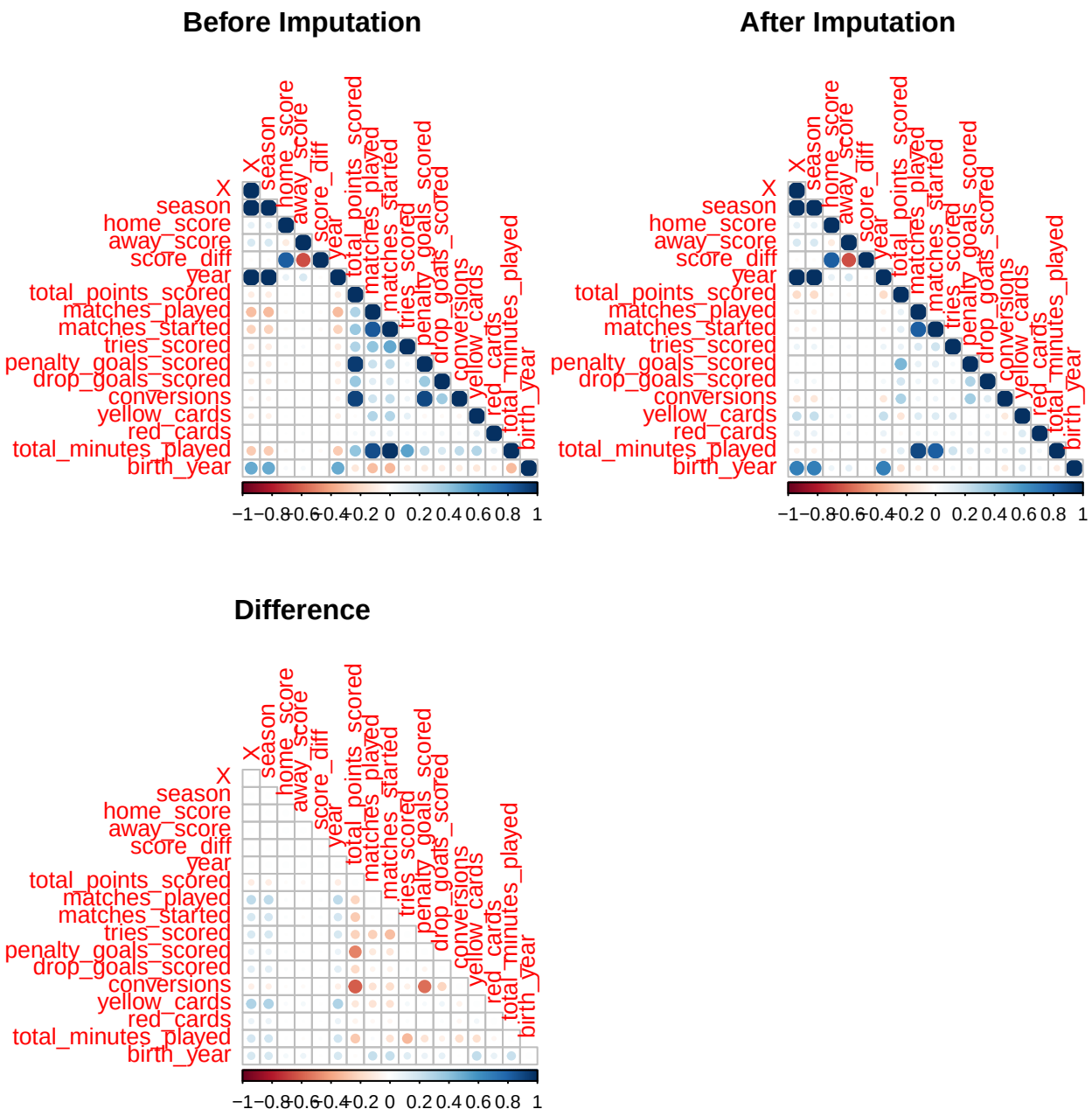
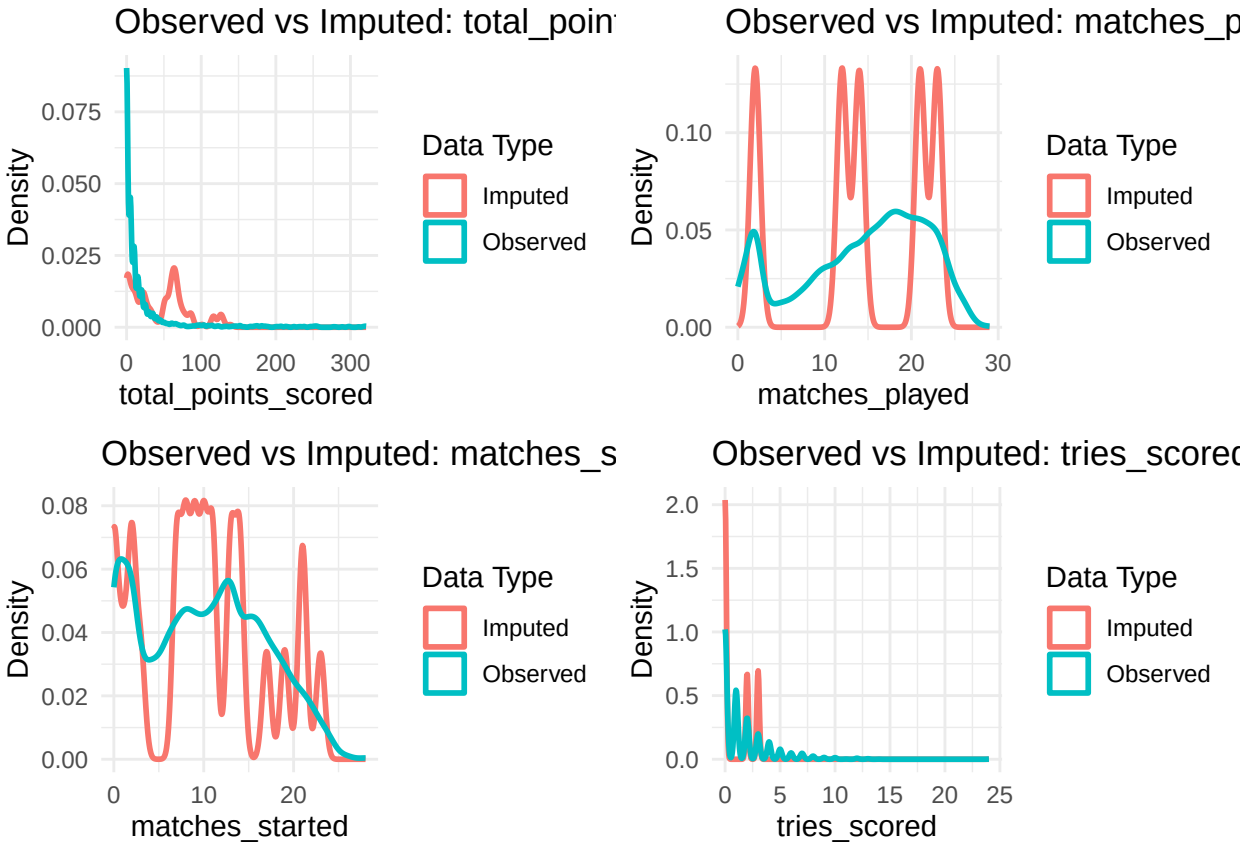
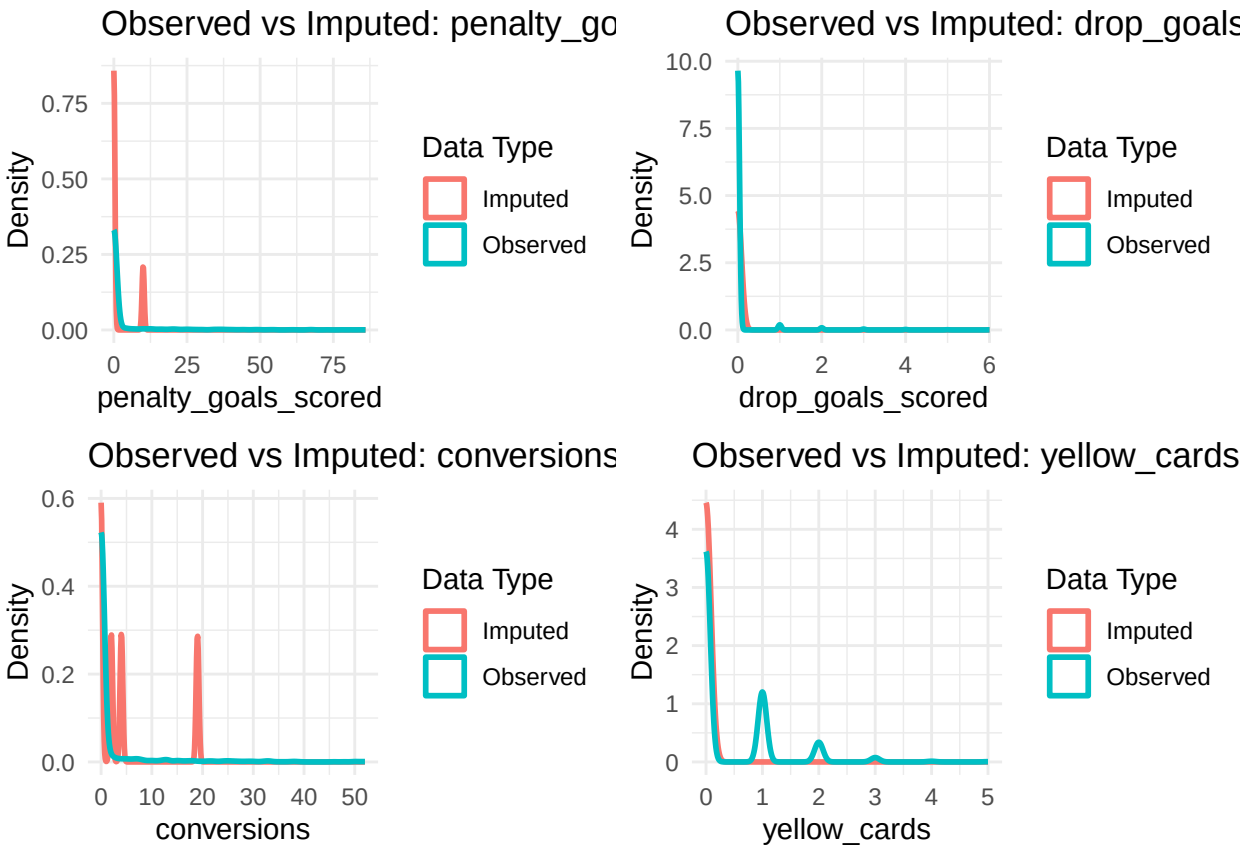


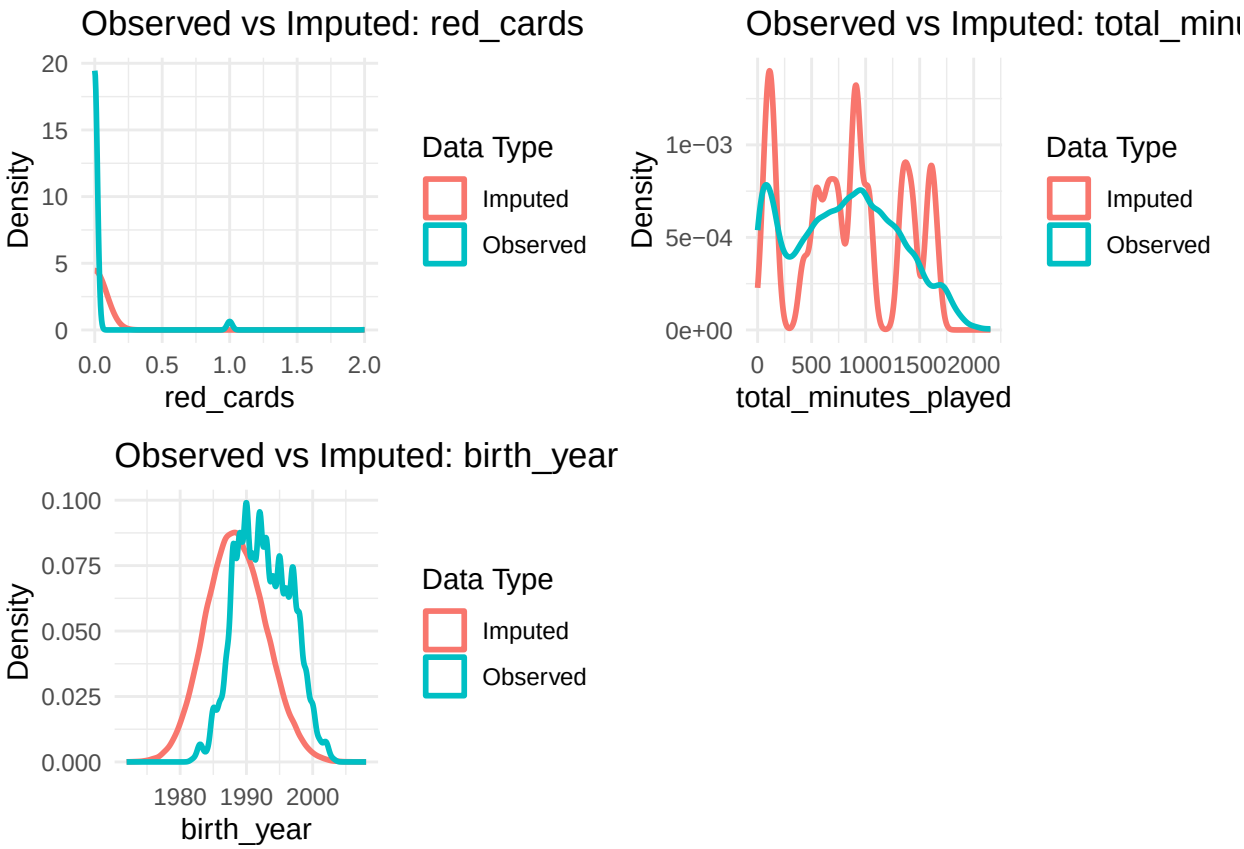
Figure 3: Comparison of Correlation structure of joined data

Visuals: Diagnosing Imputed Values

Here we check that the distributions of the original and the imputed data are similar.




Observed vs Imputed: penalty_go
Density
0.75
0.50
0.25
0.00
0
25
50
75
penalty_goals_scored
Data Type
Imputed
Observed
Observed vs Imputed: drop_goals
Density
10.0
7.5
5.0
2.5
0.0
0
2
4
6
drop_goals_scored
Data Type
Imputed
Observed
Observed vs Imputed: conversions
Density
0.6
0.4
0.2
0.0
0
10
20
30
40
50
conversions
Data Type
Imputed
Observed
Observed vs Imputed: yellow_cards
Density
4
3
2
1
0
0
1
2
3
4
5
yellow_cards
Data Type
Imputed
Observed



The imputation looks like it preserved the distributions of most of the variables. Notably, the imputation was able to find the peaks in the `yellow_cards` data and follows the distribution `birth_year` nicely, even if the average value is a bit lower, though this could mean the missing values were, on average, from earlier seasons so the players would have been born earlier. `matches_played`, `matches_started`, and `total_minutes_played` have imputations that peak in ways inconsistent with data already in the set and therefore might not have returned the most reliable potential values.

Some of the imputation is good, some is not as great in part due to the large amount of data that was missing and some variables with a large amount of zeroes. The dataset can now be completed by filling in the rest of the `joined` dataset with the imputed values, knowing it is not the most accurate set but it is now complete.

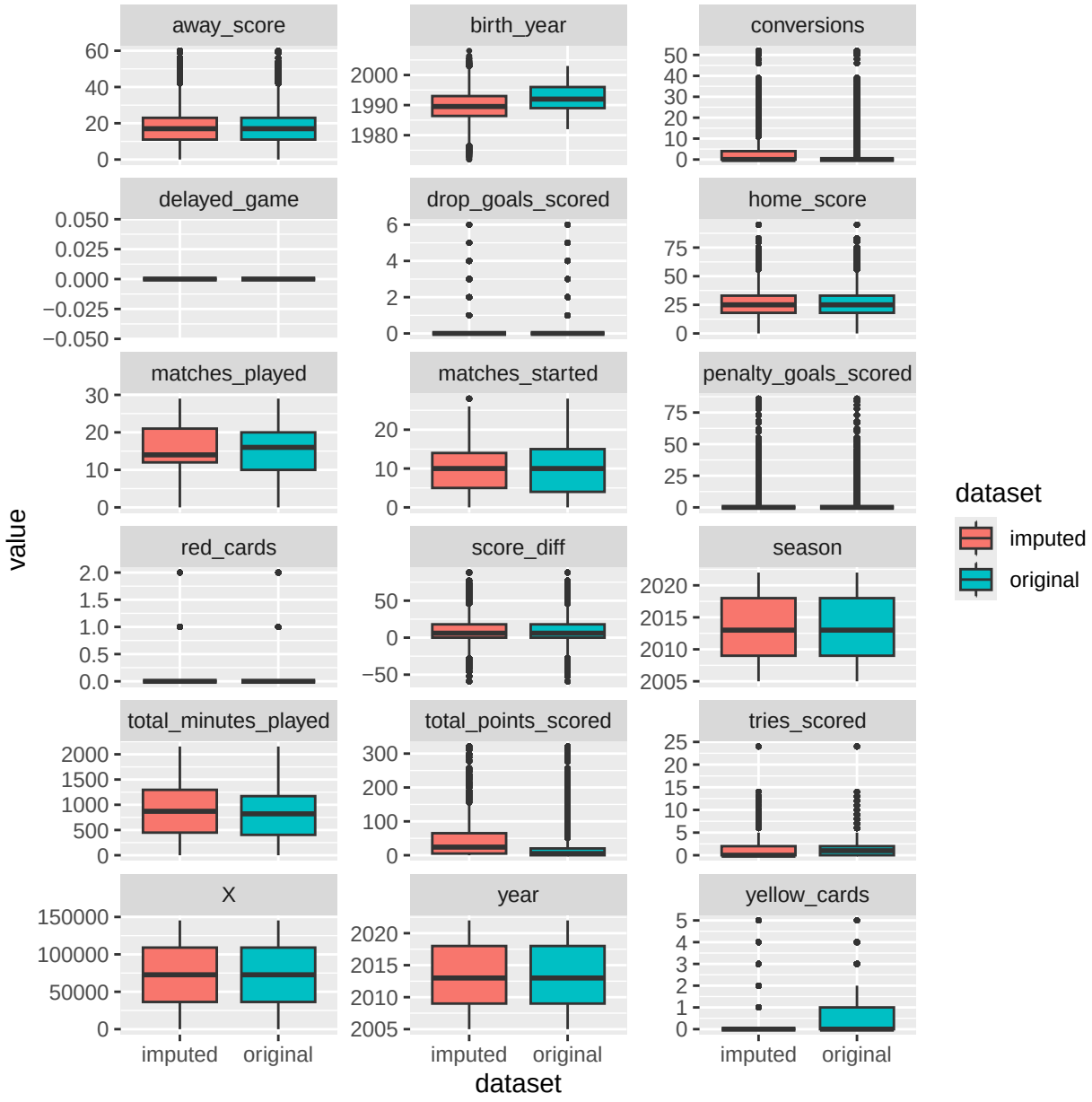


Figure 4: Data spread comparison between original joined data and joined_complete data

One thing to note is the data spread of the original `joined` data matches well with that of the `joined_complete` data filled in with imputed values.

Interpretable: Single Regression Tree for the score differential

This tree predicts the score difference using the completed data. When restricting the dataset to not include values used to compute the `score_diff`, we see that `away_team` and `home_team` are the only predictors. This indicates that certain teams tend to have a larger score difference in their matches than others. We can use cross validation to see if a less complex tree is a better predictor of `score_diff`.

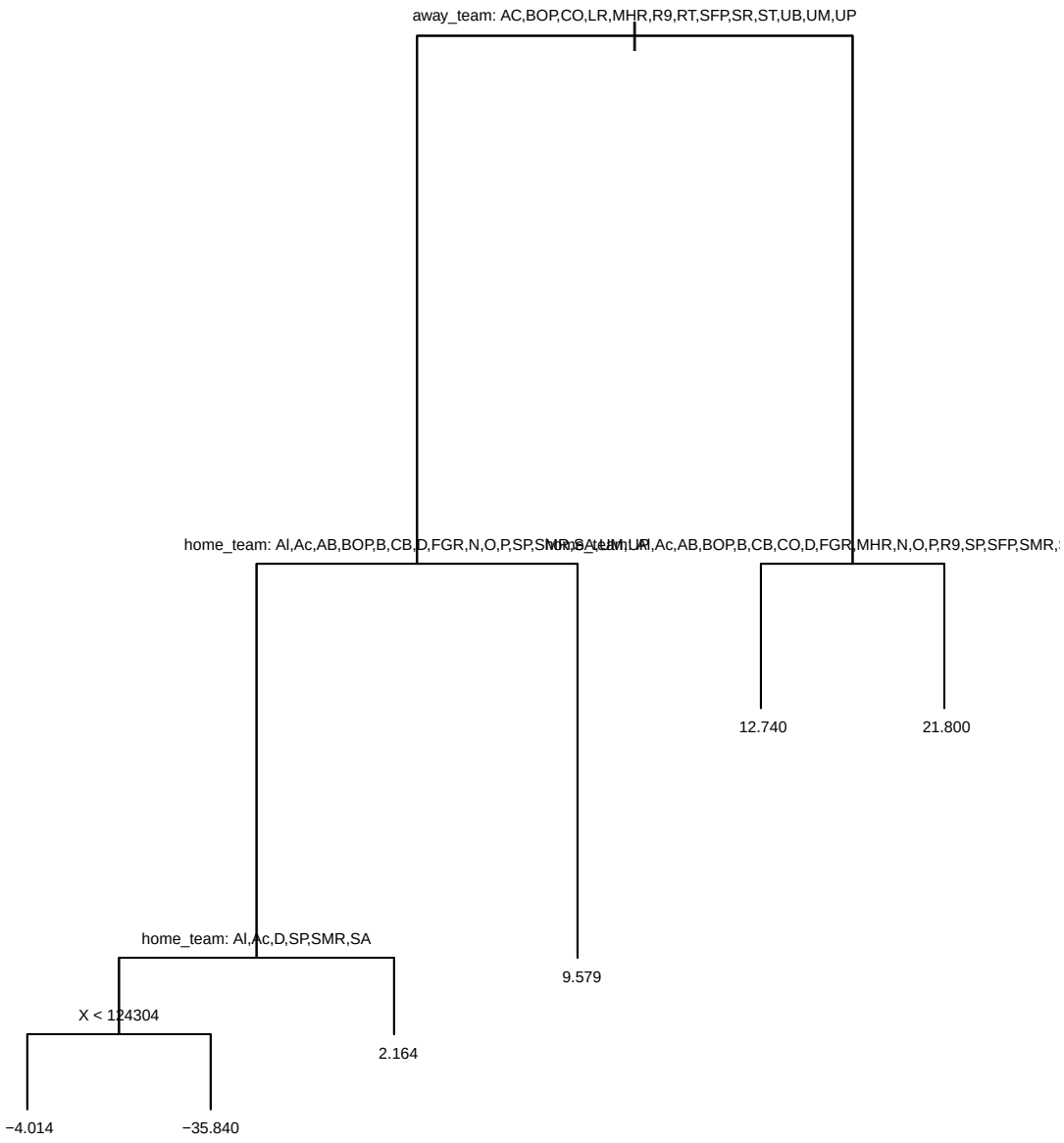


Figure 5: Score_diff estimation tree

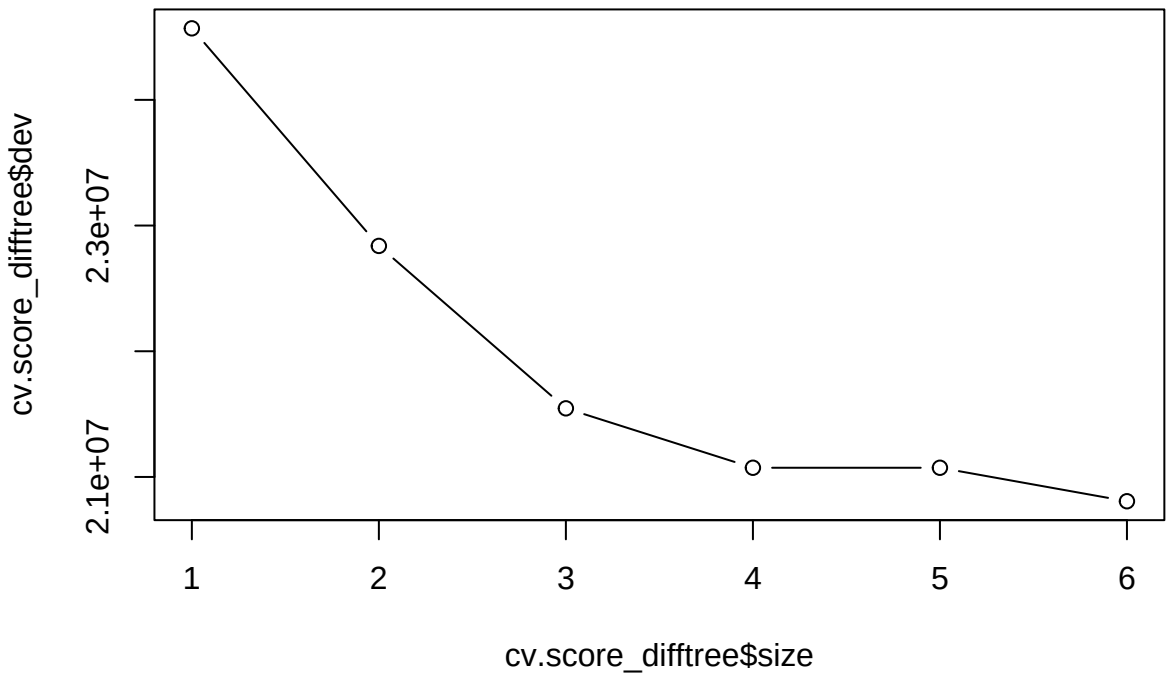


Figure 6: CV of score_diff tree size

The most complex tree is the best one, according to cross validation, so no pruning is needed. However, in evaluating the predictions using the test data, I add a pruned tree to show the 4 node tree is the best predictor.

Whole Tree MSE:	209.8561
Pruned Tree MSE:	219.2003

The pruned tree has a higher MSE value, indicating the whole tree is a better predictor.

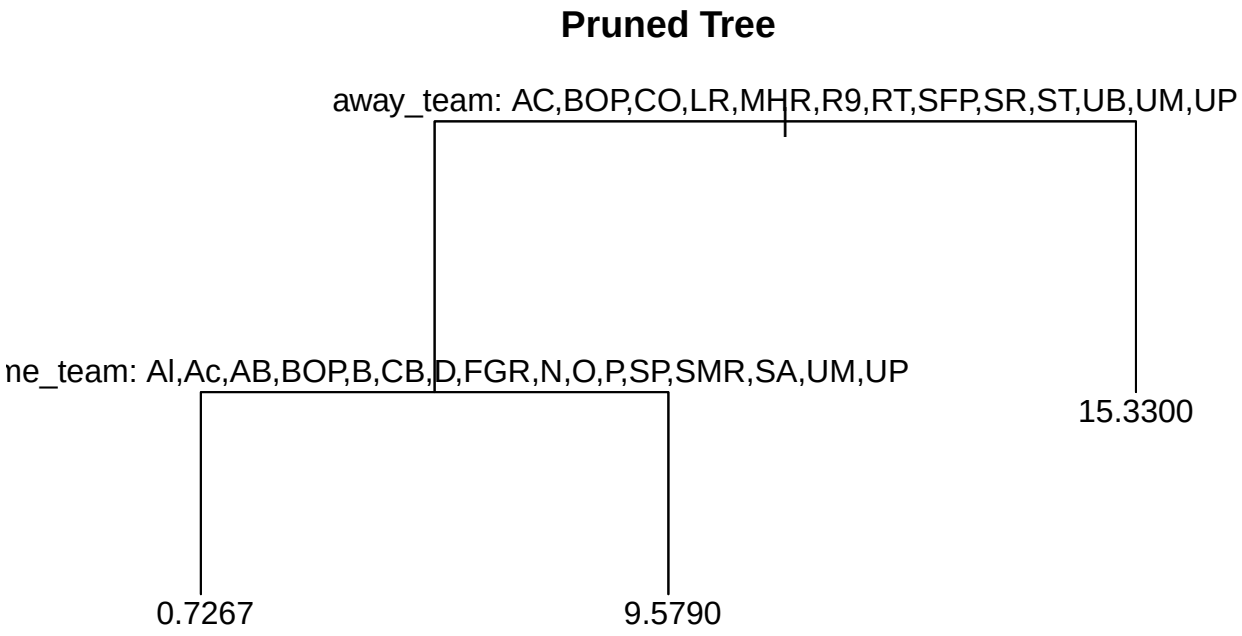


Figure 7: Pruned Tree

The pruned tree with one less split than the original removed the split on the right.

Betting metric: Predictive Random Forest using Completed data

First, we calculate the ELO. The expected win probability of team A is given by

$$P_A = \frac{1}{1 + 10^{\frac{R_B - R_A}{400}}}$$

Where R_A and R_B are the ratings for teams A and B, respectively. Each time a match is played, these ratings change depending on the outcome. After a match, the updated ratings are given by

$$R'_A = R_A + K \times (S_A - P_A)$$

Where S_A is the actual outcome for team A. S_A is 1 if team A wins, -0.5 for a draw, and 0 for a loss. K is a sensitivity parameter. Lower values move ratings less and higher values move them more. In Top14 where there are ~26 matches per season, a K value of 20-30 is reccomended, so I will use 25.

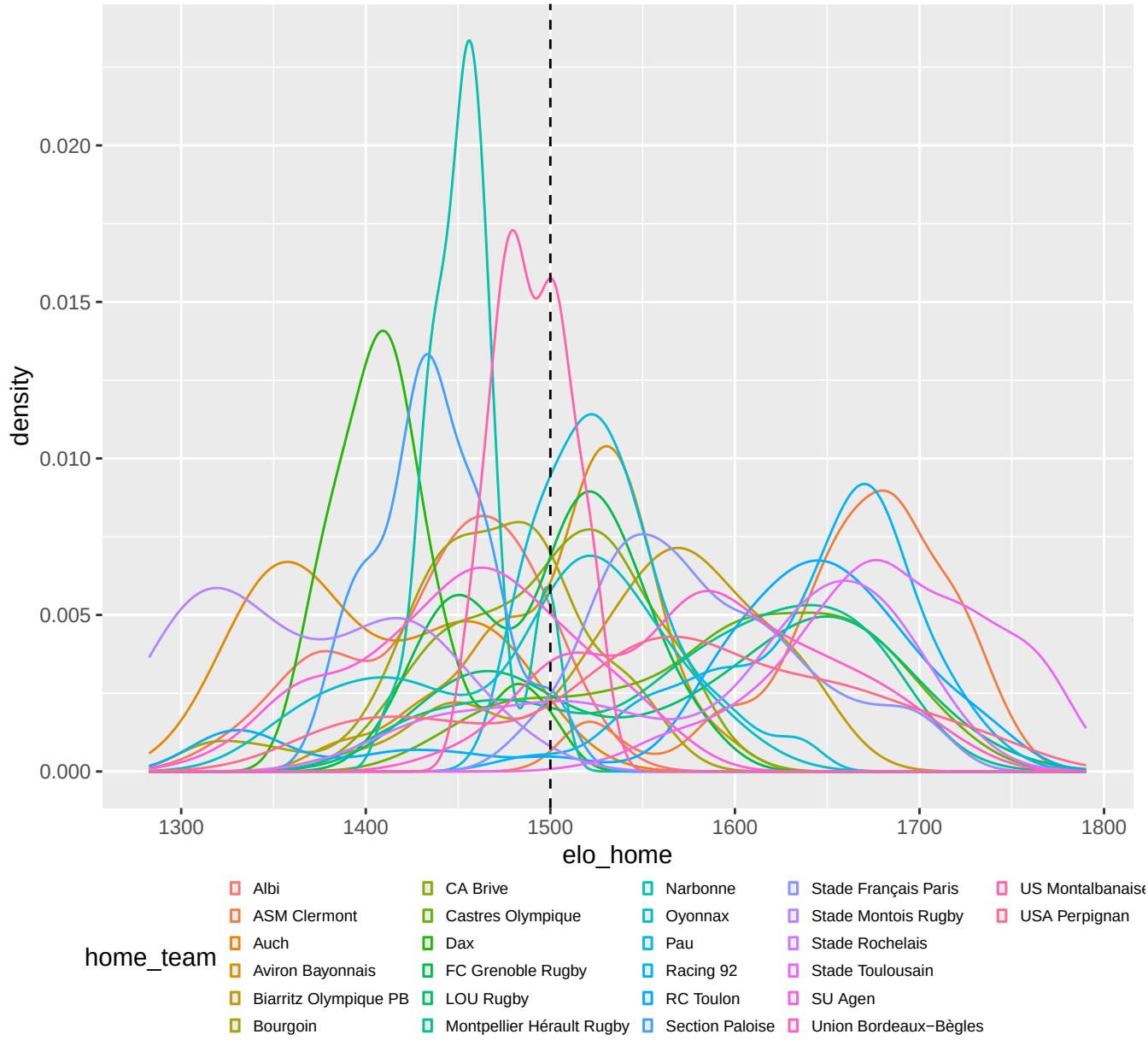


Figure 8: Density of Elo Score by team

Visual of the home team's Elo value density. Each line is one team. Elo distributions with peaks on the right of the dotted line at `elo_home` = 1500 are more favored to win. These include Stade Toulousain, RC Toulon, and ASM Clermont who have generally been winning teams. On the left, Auch and CA Brieve are prominent which makes sense as they are not the best performing teams.

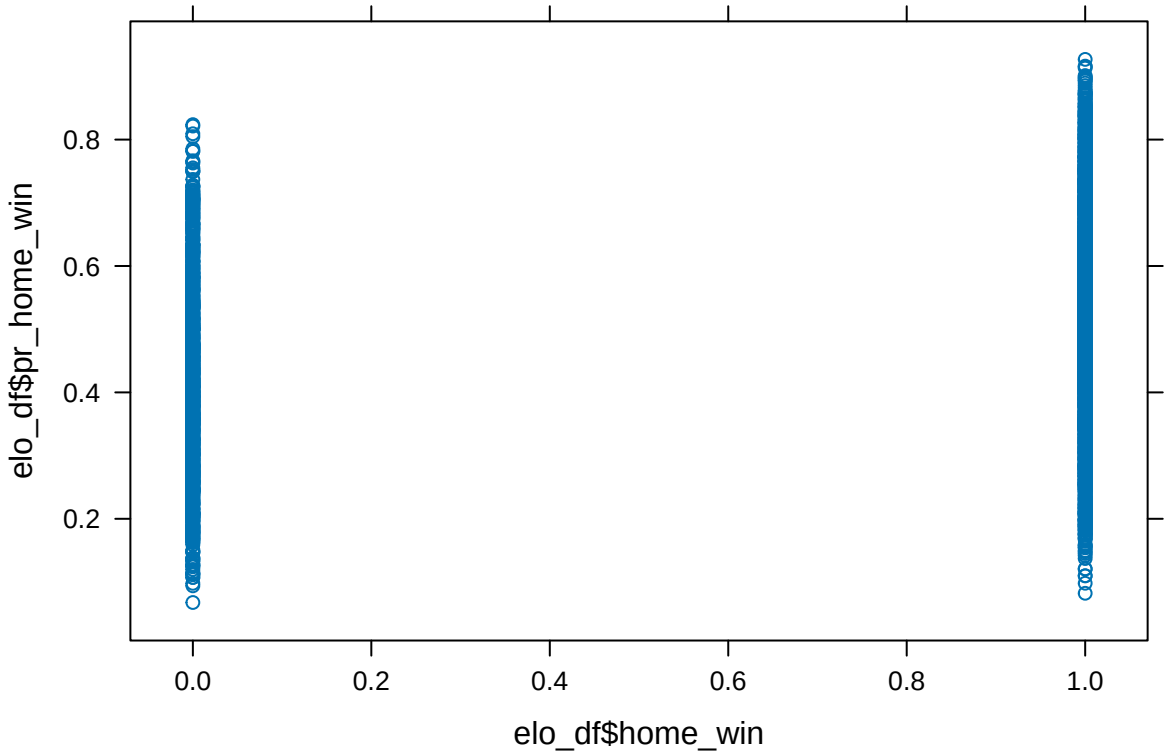


Figure 9: Elo-based home win prediction vs true home wins

So far, just using the Elo as a predictive win metric does not seem like a good idea. We will create more potentially predictive metrics using the player information from the `joined_complete` dataset to generate whole team statistics for each individual match lineup.

team	season	day	avg_birth_year_away	tries_in_match_away
Castres Olympique : 236	Min. :2005	J1 : 126	Min. :1982	Min. : 0.00
Stade Français Paris : 232	1st Qu.:2009	J13 : 126	1st Qu.:1987	1st Qu.:19.00
ASM Clermont : 230	Median :2013	J2 : 126	Median :1990	Median :25.00
Stade Toulousain : 230	Mean :2013	J6 : 126	Mean :1990	Mean :26.25
Montpellier Hérault Rugby: 228	3rd Qu.:2018	J7 : 126	3rd Qu.:1992	3rd Qu.:32.00
RC Toulon : 204	Max. :2022	J11 : 125	Max. :1997	Max. :72.00
(Other) :1841	NA	(Other):2446	NA	NA

home_win	elo_home	elo_away	pr_home_win
Min. :0.0000	Min. :1283	Min. :1291	Min. :0.06756
1st Qu.:0.0000	1st Qu.:1500	1st Qu.:1504	1st Qu.:0.36521
Median :1.0000	Median :1573	Median :1575	Median :0.49436
Mean :0.7195	Mean :1571	Mean :1575	Mean :0.49473
3rd Qu.:1.0000	3rd Qu.:1652	3rd Qu.:1655	3rd Qu.:0.62300
Max. :1.0000	Max. :1790	Max. :1807	Max. :0.92692

The new dataset includes team stats aggregating those of the lineups for both the home and away teams.

	0	1
Number of Observations	898	2303
% of Dataset	28.05	71.95

The data has about a 70/30 split of wins to losses which is not extremely imbalanced, but is a little skewed towards wins. This makes sense, as this is just the home team win rate. It also indicates teams are more likely to win on their home turf.

Varaiable Importance

It looks like `pr_home_win` and `day` are the most important predictors, followed by `elo_home`, `elo_away`, and `team`. Elo seems to be a helpful predictor in these models.

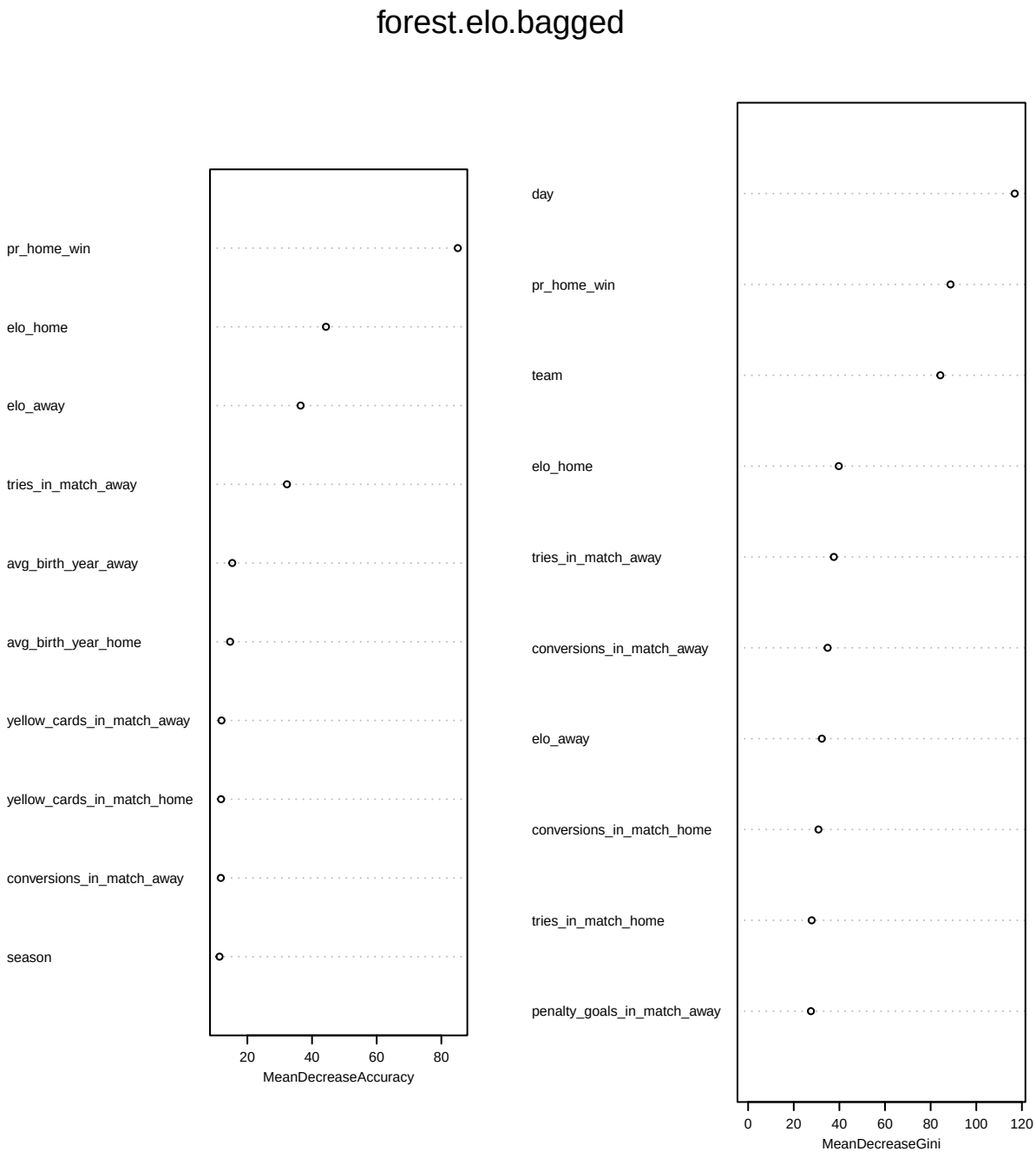


Figure 10: Variable Importance for BAgged RF predicting home_win

The regular random forest has the same set of most important variables as the bagged model. For building these models, I could have stripped away all the unimportant variables to limit the noise in predictions or just make a cleaner model.

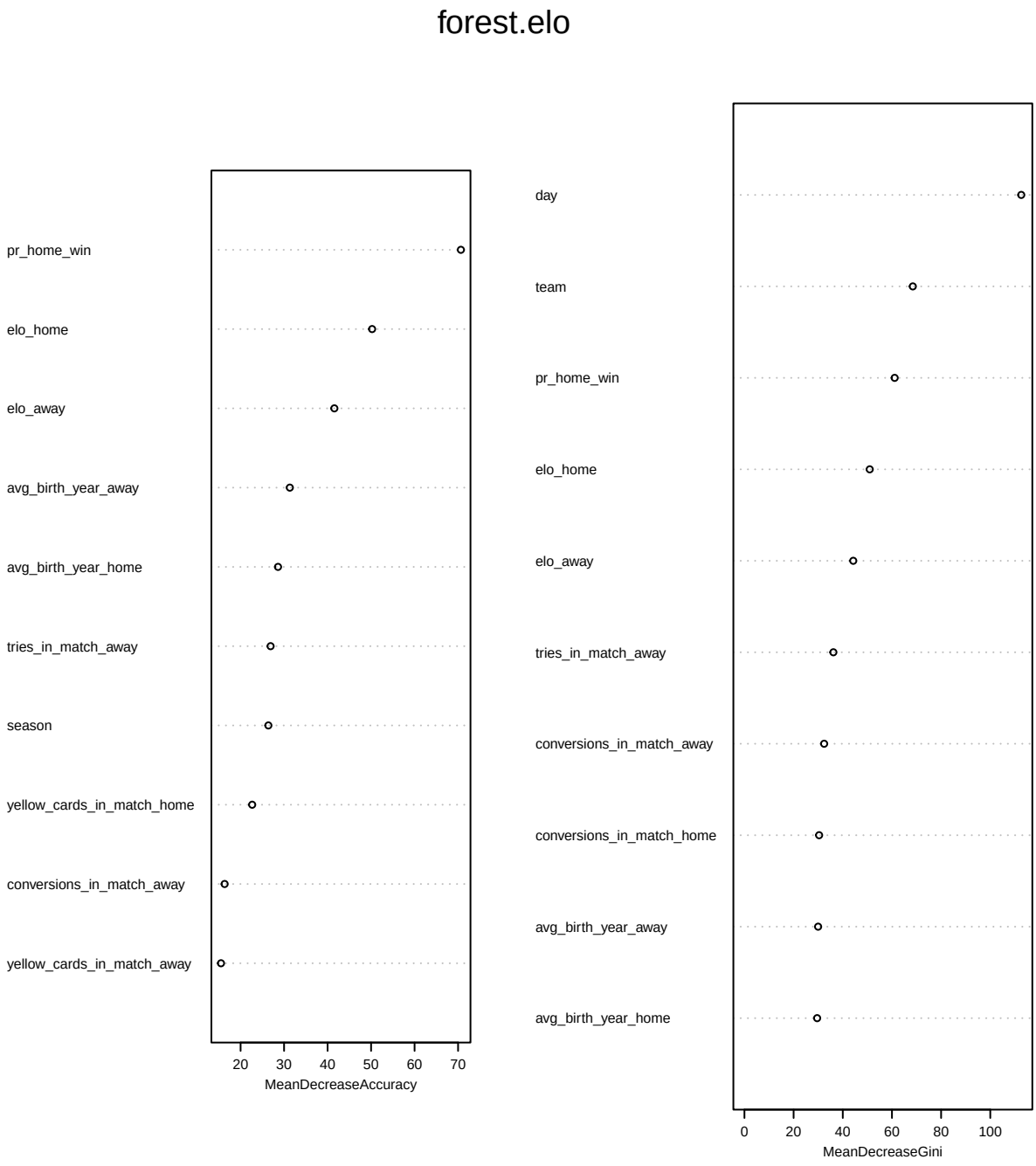


Figure 11: Variable Importance for vanilla RF predicting home_win

Pruning the Single Tree

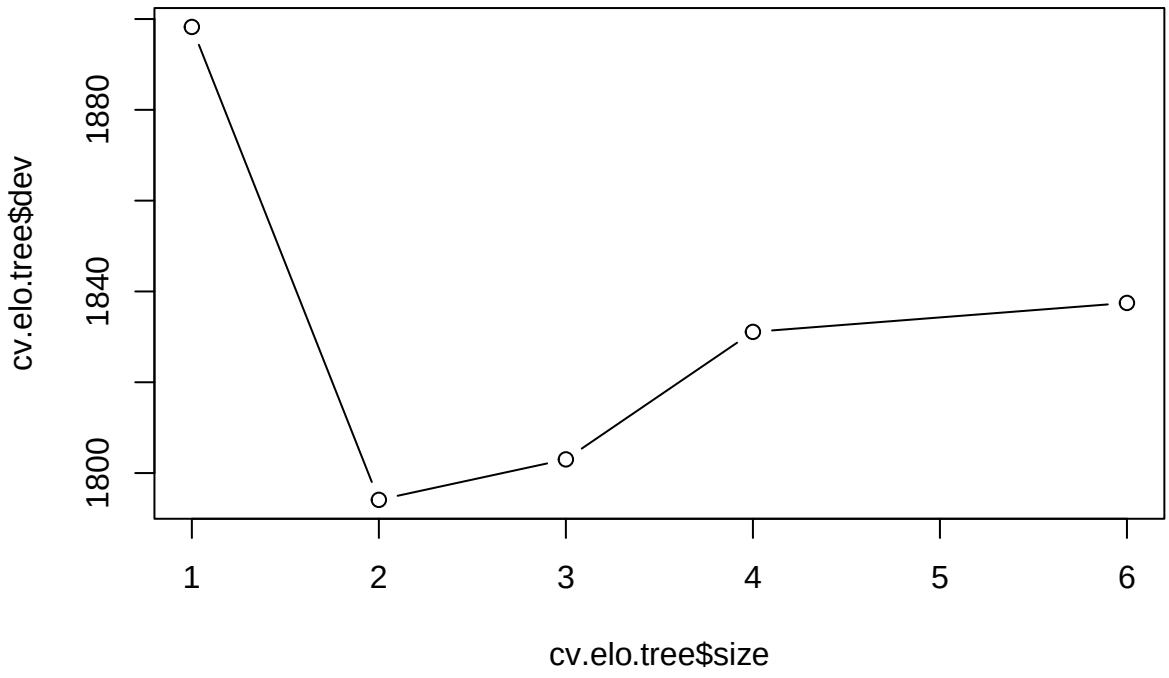


Figure 12: CV for single tree predicting home_win

The best predictive single tree is a tiny 2 split according to the cross validation.

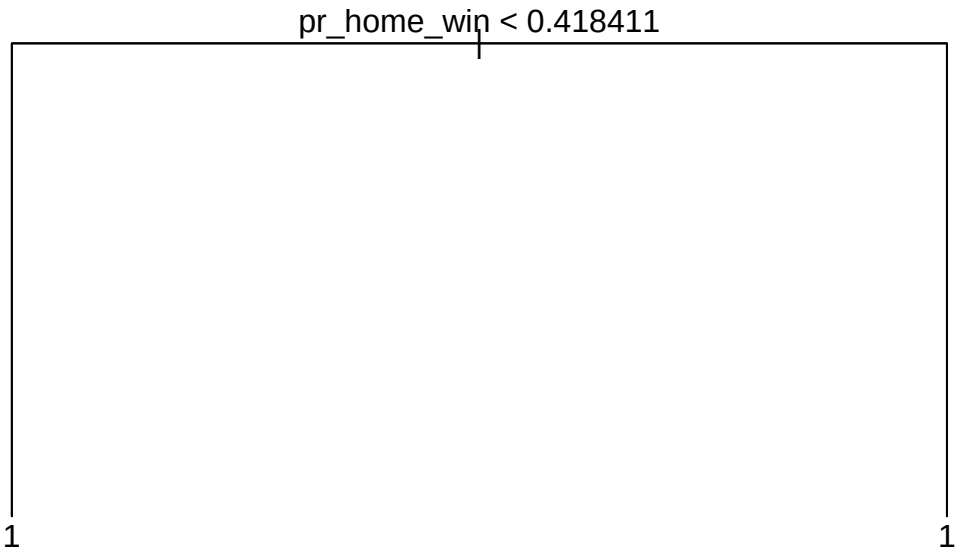


Figure 13: Pruned single tree predicting home_win

This is the pruned tree, it is not very impressive and is based solely on the probability of a home win calculated using the home team and away team Elo values.

Model	Accuracy
Random Forest	0.73448
LASSO	0.73380
Elastic Net	0.73347
Ridge	0.73232
Bagged Random Forest	0.73106
Pruned Single Tree	0.71965
Whole Single Tree	0.71431
GLM	0.70169
Elo	0.61326

Out of all the models cross validated to predict `home_wing`, the Random Forest and Elastic net models are the best at predicting a home win with about 73% accuracy.

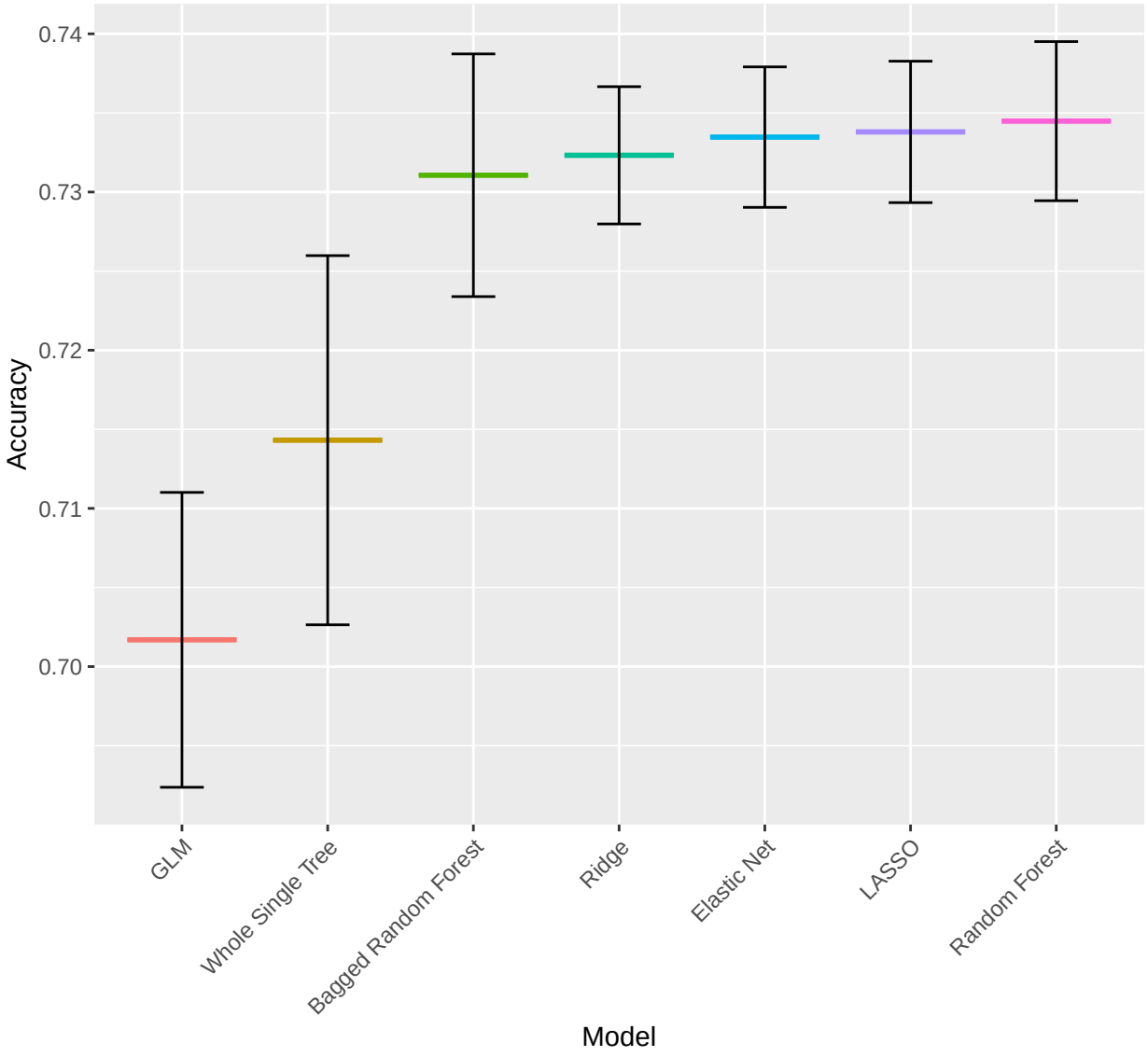


Figure 14: Accuracies of cross validated models predicting home_win

All of these models do better than the raw Elo score, with the Random forest and Elastic Net models being the most effective. The random forest could be improved with boosting or cross validation to tune hyperparameters such as the number of predictors used, `mtry`, and number of trees in the forest, `ntrees`. Elastic Net is already cross validated to use the best lambda. A deeper dive might use these methods to improve the random forest. I will just tune the hyperparameters.

Improving Random Forest and Choosing Best parameters

nmtry	ntree	accuracy
10	9500	0.7485929
10	6500	0.7453066
10	9500	0.7446809
12	8500	0.7442151
16	6000	0.7442151

Based on a 2-fold cross validation accross 5 different potential parameter amounts and 9 potential tree sizes, it was determined that an `mtry` value of – parameters and a forest size (`ntree` value) of – is preferred. This produces an increase of –% over the untuned model using `mtry` = 4 and `ntree` = 5000.

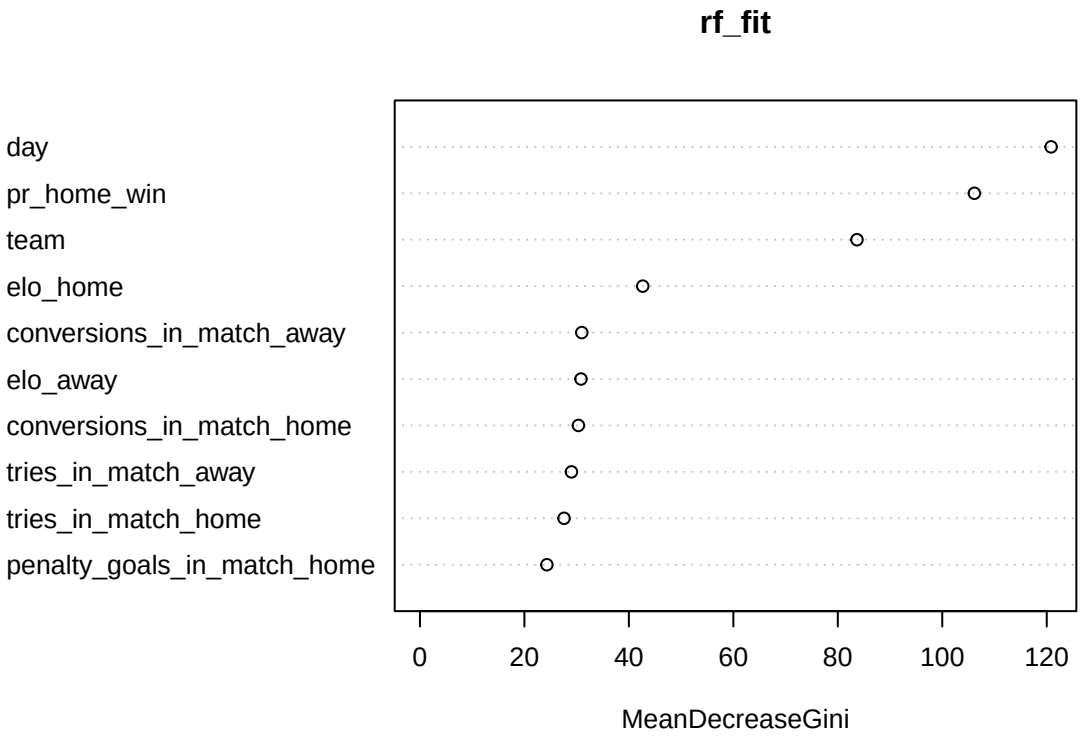


Figure 15: Variable importance for the tuned random forest

Looking at Elastic Net

	Coefficient
pr_home_win	3.0429531
dayJ4	0.2233166
dayJ19	0.1830333
teamAuch	0.1259688
dayJ25	0.0493225
drop_goals_in_match_home	0.0230430
tries_in_match_home	0.0045040
elo_home	0.0012228

The Elastic Net model, like the trees, uses mostly the probability of a home win, calculated using the Elo score. The intercept is the most significant coefficient, which could indicate a baseline higher probability of a home win which makes sense given the tendance of the data towards home wins. Based on this model, home teams tend to win the most in day 4 and both Narbonne and CA Brive win much more often on their own turf.

Code Appendix

```
players_raw <- read.csv("clean_data/players_clean.csv")
matches_raw <- read.csv("clean_data/top14_clean.csv")
joined_raw <- read.csv("clean_data/joined.csv")
matches_long_raw <- read.csv("clean_data/top14_clean_long.csv")
```

```
# Player data // 5 birth years are missing -> Imputation
players <- players_raw %>%
  mutate(team = as.factor(team)) %>%
  mutate(position = as.factor(position)) %>%
  mutate(player_name = as.factor(player_name)) %>%
  mutate(birth_year = as.numeric(str_sub(birthdate, -4))) %>%
  select(-competition, -player_id, -birthdate)

# Match Data
matches <- matches_raw %>%
  select(-date, -stadium) %>%
  mutate(season = as.factor(season),
  day = as.factor(day),
  home_team = as.factor(home_team),
  away_team = as.factor(away_team),
  winning_team = as.factor(winning_team)) %>%
  mutate(across(matches("^home_[0-9]+$"),as.factor)) %>%
  mutate(across(matches("^away_[0-9]+$"),as.factor))

# By-Player Match data (long)
matches.long <- matches_long_raw %>%
  select(-date, -stadium) %>%
  mutate(season = as.factor(season),
         day = as.factor(day),
         home_team = as.factor(home_team),
         away_team = as.factor(away_team),
         winning_team = as.factor(winning_team),
         side = as.factor(side),
         player_name = as.factor(player_name))

# Joined
joined <- joined_raw %>%
  rename(position = position.y) %>%
  mutate(team = ifelse(side == "home", home_team, away_team)) %>%
  mutate(season = as.numeric(str_sub(season, end = 4)),
                              day = as.factor(day),
                              home_team = as.factor(home_team),
                              away_team = as.factor(away_team),
                              winning_team = as.factor(winning_team),
                              side = as.factor(side),
                              player_name = as.factor(player_name),
                              birth_year = as.numeric(str_sub(birthdate, -4)),
                              team = as.factor(team),
                              position = as.factor(position)) %>%
  select(-stadium, -player_id, -birthdate, -position.x, -competition, -date)
```

Imputation

Diagnose Missingness

```
# Create missingness indicator, drop 'conversions'
players.impute <- players %>%
  mutate(missing_birthyr = as.integer(is.na(players$birth_year)))

# Diagnose missingness
# corrplot
players_numeric <- players.impute %>%
  select(where(is.numeric), -missing_birthyr, -X)
cor_matrix <- cor(players_numeric, use = "complete.obs")
corrplot(cor_matrix, type = "lower")
```

```
# Chi sq association test for categorical -> ISSUE player name has one per group:
  # approx may be incorrect
players_factors <- players.impute %>%
  select(where(is.factor))
PT <- chisq.test(table(players_factors$position, players_factors$team),
                 simulate.p.value = TRUE)
PN <- chisq.test(table(players_factors$position, players_factors$player_name),
                 simulate.p.value = TRUE)
TN <- chisq.test(table(players_factors$team, players_factors$player_name),
                 simulate.p.value = TRUE)

print(PT)
print(PN)
print(TN)
```

```
# RIDGE
# remove birthyear (this is what we want to impute)
data <- players.impute %>% select(-birth_year)
x <- model.matrix(missing_birthyr~., data = data)[,-1]
y <- data$missing_birthyr

ridge_fit <-glmnet(x, y, alpha=0)
ridge_fit.cv <-cv.glmnet(x, y, alpha=0)

best_l <- ridge_fit.cv$lambda.min
cat("Best Ridge Lambda:", round(best_l, 5))

coefs_ridge <- coef(ridge_fit.cv, s = "lambda.min")
```

```
elastic_fit <-glmnet(x, y, alpha=0)
elastic_fit.cv <-cv.glmnet(x, y, alpha=0)

best_l <- elastic_fit.cv$lambda.min
cat("Best Elastic Net Lambda:", round(best_l, 5))
coefs_elastic <- coef(elastic_fit.cv, s = "lambda.min")
```

```r
fig1 <- ggplot(ridge_coef_df.cut, aes(x = reorder(variable, lambda.min),
                                      y = abs(lambda.min))) +
  geom_col() +
  scale_y_log10()+
  coord_flip() + labs(title = "Ridge Coeffiecients")

fig2 <- ggplot(elastic_coef_df.cut, aes(x = reorder(variable, lambda.min),
                                      y = abs(lambda.min))) +
  geom_col() +
  scale_y_log10()+
  coord_flip() + labs(title = "Elastic Net Coefficients")

plot_grid(fig1, fig2, ncol=1, align="v")
```

```r
imp_players_rf <- mice(players.impute,
                       m = 5,
                       method = "rf",
                       maxit = MICEMAXIT,
                       printFlag = FALSE)
#Looks at how each estimated value changed over each round of imputation
#imp_players_rf$imp$birth_year
```

```r
imp_players_pmm <- mice(players.impute,
                       m = 5,
                       method = "pmm",
                       maxit = MICEMAXIT,
                       printFlag = FALSE)
#Looks at how each estimated value changed over each round of imputation
#imp_players_pmm$imp$birth_year
```

```r
# Built by chatGPT
plot_imputation_density <- function(var_name, original_data, imp_obj, dataset = 1) {
  # Extract observed (non-missing) values
  observed <- original_data[[var_name]][!is.na(original_data[[var_name]])]

  # Extract imputed values for the first completed dataset by default
  if (!var_name %in% names(imp_obj$imp)) {
    stop(paste("Variable", var_name, "not found in imp_obj$imp"))
  }

  imputed <- imp_obj$imp[[var_name]][[dataset]]

  # Combine into a single data frame for plotting
  df_plot <- data.frame(
    value = c(observed, imputed),
    type = c(rep("Observed", length(observed)),
             rep("Imputed", length(imputed)))
  )

  # Make the density plot
  p <- ggplot(df_plot, aes(x = value, color = type)) +
    geom_density(linewidth = 1) +
    theme_minimal() +
```

```
    labs(
      x = var_name,
      y = "Density",
      title = paste("Observed vs Imputed:", var_name),
      color = "Data Type"
    )

  return(p)
}
```

```
# PLAYERS: BIRTH_DATE
p1 <- plot_imputation_density(var_name = "birth_year",
                          original_data = players.impute,
                          imp_obj = imp_players_rf) + labs(subtitle = "Method : Random Forest")

p2 <- plot_imputation_density(var_name = "birth_year",
                          original_data = players.impute,
                          imp_obj = imp_players_pmm) + labs(subtitle = "Method: pmm")

plot_grid(p1, p2, nrow = 2, ncol = 1, align = "v")
```

```
# Remove missing data to look at the distributions of variables
# so we can choose the imputation method
joined_no.missing <- joined %>% na.omit()

par(mfrow = c(4, 3))
plot(density(joined_no.missing$total_points_scored))
plot(density(joined_no.missing$matches_played))
plot(density(joined_no.missing$matches_started))
plot(density(joined_no.missing$tries_scored))
plot(density(joined_no.missing$penalty_goals_scored))
plot(density(joined_no.missing$drop_goals_scored))
plot(density(joined_no.missing$conversions))
plot(density(joined_no.missing$yellow_cards))
plot(density(joined_no.missing$red_cards))
plot(density(joined_no.missing$total_minutes_played))
plot(density(joined_no.missing$birth_year))
```

```
# initialize imputation
ini <- mice(joined, maxit = 0, print = FALSE)
methods <- ini$method
predictors <- ini$predictorMatrix

# Cols with missing data in joined:
# Position is an unordered categorical variable wih more than 3 levels
methods["position"] <- "polyreg"
# team is also an unordered categorical, but with a lot more levels
methods["team"] <- "rf"
# pmm is recommended for numeric variables; it preserves the distribution and does not assume linearity
# These predictors all have not normal distributions
methods["total_points_scored"] <- "pmm"
methods["matches_played"] <- "pmm"
methods["matches_started"] <- "pmm"
```

```
methods["tries_scored"] <- "pmm"
methods["penalty_goals_scored"] <- "pmm"
methods["drop_goals_scored"] <- "pmm"
methods["conversions"] <- "pmm"
methods["yellow_cards"] <- "pmm"
methods["red_cards"] <- "pmm"
methods["total_minutes_played"] <- "pmm"
# Bayesian linear regression for a numeric variable that is normally distributed
methods["birth_year"] <- "norm"

# Restrict predictors to avoid multicollinearity
predictors[] <- 0  # no predictors by default
predictors["total_points_scored", c("tries_scored","conversions","penalty_goals_scored","drop_goals_sco
predictors["matches_started", "matches_played"] <- 1
predictors["total_minutes_played", "matches_played"] <- 1
predictors["position", c("team","birth_year")] <- 1
predictors["team", c("position","birth_year")] <- 1
predictors["birth_year", c("season")] <- 1

# Block high-cardinality variable
predictors[, "player_name"] <- 0

imp_joined <- mice::mice(joined,
                         m = 5,
                         method  = methods,
                         predictorMatrix = predictors,
                         maxit = MICEMAXIT,
                         printFlag = FALSE)
```

```
plot(imp_joined)
```

```
# Imputed values are in red
# xyplot(imp_joined, total_points_scored ~ matches_played)
# xyplot(imp_joined, total_points_scored ~ tries_scored)
# xyplot(imp_joined, total_points_scored ~ matches_started)
# xyplot(imp_joined, total_points_scored ~ penalty_goals_scored)
# xyplot(imp_joined, total_points_scored ~ drop_goals_scored)

xyplot(imp_joined, total_points_scored ~ conversions)
xyplot(imp_joined, matches_played ~ matches_started)
xyplot(imp_joined, total_points_scored ~ birth_year)
xyplot(imp_joined, matches_played ~ tries_scored)

# xyplot(imp_joined, matches_played ~ conversions)
# xyplot(imp_joined, matches_played ~ yellow_cards)
# xyplot(imp_joined, matches_played ~ red_cards)
# xyplot(imp_joined, matches_played ~ total_minutes_played)
# xyplot(imp_joined, matches_played ~ birth_year)
#
# xyplot(imp_joined, tries_scored ~ matches_started)
# xyplot(imp_joined, tries_scored ~ penalty_goals_scored)
# xyplot(imp_joined, tries_scored ~ drop_goals_scored)
# xyplot(imp_joined, tries_scored ~ conversions)
```

```r
# xyplot(imp_joined, tries_scored ~ yellow_cards)
# xyplot(imp_joined, tries_scored ~ red_cards)
# xyplot(imp_joined, tries_scored ~ total_minutes_played)
# xyplot(imp_joined, tries_scored ~ birth_year)
```

```r
# Correlation structure
cor_before <- joined %>%
  select(where(is.numeric)) %>%
  select(where(~ sd(.x, na.rm = TRUE) != 0))%>%
  cor(use = "pairwise.complete.obs")

cor_after  <- imp_joined %>%
  complete() %>%
  select(where(is.numeric)) %>%
  select(where(~ sd(.x, na.rm = TRUE) != 0))%>%
  cor(use = "pairwise.complete.obs")

cor_diff <- cor_after - cor_before

par(mfrow = c(2, 2), cex.lab = 0.9)

corrplot(cor_before, type = "lower")
title("Before Imputation", line = 1)

corrplot(cor_after, type = "lower")
title("After Imputation", line = 1)

corrplot(cor_diff, type = "lower")
title("Difference", line = 1)
```

```r
# Cols with missing data in joined:
colNames <- c("total_points_scored",
              "matches_played",
              "matches_started",
              "tries_scored",
              "penalty_goals_scored",
              "drop_goals_scored",
              "conversions",
              "yellow_cards",
              "red_cards",
              "total_minutes_played",
              "birth_year")
# Numeric
for (i in 1:length(colNames)){
  figName <- paste("fig", i, sep = "")
  # Assign from claude
  assign(figName, plot_imputation_density(var_name = colNames[i],
                         original_data = joined,
                         imp_obj = imp_joined))
}
```

```r
plot_grid(fig1, fig2, fig3, fig4)
plot_grid(fig5, fig6, fig7, fig8)
plot_grid(fig9, fig10, fig11)
```

```r
joined_complete <- complete(imp_joined)

joined_long <- joined %>%
  select(where(is.numeric))%>%
  pivot_longer(cols = everything(),
               names_to = "variable",
               values_to = "value") %>%
  mutate(dataset = "original")

joined_comp_long <- joined_complete %>%
  select(where(is.numeric))%>%
  pivot_longer(cols = everything(),
               names_to = "variable",
               values_to = "value") %>%
  mutate(dataset = "imputed")

compare_df <- bind_rows(joined_long, joined_comp_long)
```

```r
# Plotting from CHatGPT
ggplot(compare_df, aes(x = dataset, y = value, fill = dataset)) +
  geom_boxplot(outlier.size = 0.5) +
  facet_wrap(~ variable, nrow = 6, ncol = 3, scales = "free_y")
```

```r
joined_complete.lim <- joined_complete %>% select(-home_score, -away_score,
                                                  -player_name, -winning_team)

n <- nrow(joined_complete.lim)
index <- sample(n, floor(n*.6667))
train <- joined_complete.lim %>% slice(index)

tree.joined <- tree(score_diff~., data = train)

test <- joined_complete.lim %>% slice(-index)

#predict(tree.joined)

#summary(tree.joined)
```

```r
plot(tree.joined)
text(tree.joined, pretty = .5, cex=0.5)
title("Score-diff Estimation Tree")
```

```r
cv.score_difftree <- cv.tree(tree.joined)
plot(cv.score_difftree$size , cv.score_difftree$dev , type = "b")
```

```r
yhat <- predict(tree.joined , newdata = test)
joined.test <- test$score_diff
# plot(yhat , joined.test)
```

```r
# abline (0, 1)
whole.mse <- mean (( yhat - joined.test)^2)

clipped.tree <- prune.tree(tree.joined, best=3)
yhat <- predict(clipped.tree , newdata = test)
joined.test <- test$score_diff
# plot(yhat , joined.test)
# abline (0, 1)
pruned.mse <- mean (( yhat - joined.test)^2)

cbind(c("Whole Tree MSE:", "Pruned Tree MSE:"), rbind(round(whole.mse, 4), round(pruned.mse,4)))%>% kabl
```

```r
plot(clipped.tree)
title("Pruned Tree")
text(clipped.tree, pretty = .05)
```

```r
joined_bet <- joined_complete %>% mutate(home_win = ifelse(home_score > away_score, 1, 0))
matches_bet <- matches %>% mutate(home_win = ifelse(home_score > away_score, 1, 0))
```

```r
# Code from chatGPT
elo_calc <- function(df,
                         K_regular = 20,
                         K_semi = 30,
                         K_final = 40,
                         K_barrage = 25,
                         K_access = 30,
                     # Generally accepted base Elo
                         init = 1500,
                         home_adv = 50,
                         margin_factor = 0.20) {
  # Order data by season
  df <- df %>% arrange(season, X)

  # Get team names
  teams <- unique(c(df$home_team, df$away_team))
  elo <- setNames(rep(init, length(teams)), teams)

  # Set holding variables for elo home and elo away
  elo_home <- numeric(nrow(df))
  elo_away <- numeric(nrow(df))

  # Helper to classify match type
  match_type <- function(day, winning_team, away_team, home_team) {
    d <- as.character(day)
    if (grepl("Final", d, ignore.case = TRUE)) return("final")
    if (grepl("Semi", d, ignore.case = TRUE))  return("semi")
    if(grepl("Access", d, ignore.case = TRUE)) return("access")
    if(grepl("Barrages", d, ignore.case = TRUE)) return("barrage")
    return("regular")
  }
# For each row in the dataframe
  for (i in seq_len(nrow(df))) {
    # Get team name
```

```
    home_name <- as.character(df$home_team[i])
    away_name <- as.character(df$away_team[i])

    # get initial Elo
    Rh <- elo[home_name]
    Ra <- elo[away_name]

    # Determine match class
    mtype <- match_type(df$day[i], df$winning_team[i], away_name, home_name)
    K <- ifelse(mtype == "final", K_final,
                ifelse(mtype == "semi", K_semi, K_regular))

    # Home advantage (add to home Elo before computing expected probability)
    # Makes sense to add because the mean home score is higher than away
    Rh_eff <- Rh + home_adv

    # Expected probabilities
    Ph <- 1 / (1 + 10^((Ra - Rh_eff)/400))
    Pa <- 1 - Ph

    # Actual result
    if (df$home_score[i] > df$away_score[i]) {
      Sh <- 1; Sa <- 0
    } else if (df$home_score[i] < df$away_score[i]) {
      Sh <- 0; Sa <- 1
    } else {
      Sh <- 0.5; Sa <- 0.5
    }

    # Score margin multiplier (rugby appropriate)
    margin <- abs(df$home_score[i] - df$away_score[i])
    margin_mult <- 1 + margin_factor * log(1 + margin)

    # Save baseline values for model features
    elo_home[i] <- Rh
    elo_away[i] <- Ra

    # Rating updates (use effective K)
    elo[home_name] <- Rh + (K * margin_mult) * (Sh - Ph)
    elo[away_name] <- Ra + (K * margin_mult) * (Sa - Pa)
  }

  # store as part of df
  df$elo_home <- elo_home
  df$elo_away <- elo_away

  return(df)
}
```

```
# Calculate and add Elo
elo_df <- elo_calc(matches_bet) %>%
  mutate(season = as.numeric(str_sub(season, end = 4)))
```

```r
# Visualise
elo_df %>% ggplot() + geom_density(aes(x=elo_home, color = home_team))+
  geom_vline(xintercept = 1500, lty = 2) + theme(legend.position = "bottom",
                                                 legend.text = element_text(size = 7),
                                                 legend.box.spacing = unit(0, "pt"),
                                                 legend.key.width = unit(5, "pt"),
                                                 legend.key.height = unit(5, "pt"))
```

```r
# For each match, calculate Pr(home win)
elo_df <- elo_df %>% mutate(pr_home_win = (1 / (1 + 10^((elo_away - elo_home)/400))))

#how accurate is our win probability?
xyplot(elo_df$pr_home_win ~ elo_df$home_win)
```

```r
# Get team stats for that match
team_stats <- joined_complete %>%
  group_by(team, season, day) %>%
  summarise(avg_birth_year = mean(birth_year),
            tries_in_match = sum(tries_scored),
            penalty_goals_in_match = sum(penalty_goals_scored),
            drop_goals_in_match = sum(drop_goals_scored),
            conversions_in_match = sum(conversions),
            yellow_cards_in_match = sum(yellow_cards),
            red_cards_in_match = sum(red_cards),
            .groups = "drop")

# Using the data.table library, because this join was too big for my computer
# data.table handles joins quicker than dyplr and uses less memory
# data.table suggestion and syntax from ChatGPT
elo_df <- as.data.table(elo_df)
team_stats <- as.data.table(team_stats)

# Set key for team_stats on ("team", "season", "day")
setkey(team_stats, team, season, day)

# Join elo_df with team_stats for home team
matches_elo <- team_stats[
  elo_df,
  on = .(team = home_team, season, day),
  allow.cartesian = FALSE
]

# Add suffix "_home" to team_stats columns to differentiate from away
home_cols <- setdiff(names(team_stats), c("team", "season", "day"))
setnames(matches_elo, home_cols, paste0(home_cols, "_home"))

# Join AGAIN with team_stats for the away team
matches_elo <- team_stats[
  matches_elo,
  on = .(team = away_team, season, day),
  nomatch = 0,
  allow.cartesian = FALSE
]
```

```r
# Add suffix "_away" to team_stats columns
away_cols <- setdiff(names(team_stats), c("team", "season", "day"))
away_cols <- away_cols[away_cols != "team"]   # avoid double-renaming
setnames(matches_elo, away_cols, paste0(away_cols, "_away"))
```

```r
matches_elo %>% select(1:5) %>%
  summary() %>% kable() %>%
  kable_styling(full_width = FALSE)
```

```r
matches_elo %>% select(71:74) %>%
  summary() %>% kable() %>%
  kable_styling(full_width = FALSE)
```

```r
count_table <- table(matches_elo$home_win)

cbind(c("Number of Observations", "% of Dataset"),
      rbind(as.integer(count_table),
      c(round((count_table[1]/(count_table[1]+count_table[2]))*100, 2),
        round((count_table[2]/(count_table[1]+count_table[2])*100), 2)))) %>% kable()
```

```r
# Remove high factor variables and values used to calculate whether the home team won
matches_elo_lim <- matches_elo %>% select(-(matches("^(home|away)_[0-9]+$")),
                                          - winning_team,
                                          - score_diff,
                                          - home_score,
                                          - away_score,
                                          - X,
                                          - i.team) %>%
  mutate(home_win = as.factor(home_win))
```

```r
# 5x2 K-fold CV
iterations <-10
repeats = 5
num_folds = 2

# Initialize storage vectors for test errors
test.error.glm <- test.error.lasso <- test.error.ridge <-
  test.error.enet <- test.error.bagged_rf <- test.error.rf <-
  test.error.tree <- test.error.boost <-  numeric()

#Initialize values
n <- nrow(matches_elo_lim)
p <- ncol(matches_elo_lim)-1


# Fix factor levels in test to match train (Acces match only has 4 observations)
  # based on debugging from claude
matches_elo_lim <- matches_elo_lim %>%
  filter(day != "Access Match")

for(i in 1:iterations){
  for (j in 1:repeats){
```

```r
folds <- caret::createFolds(matches_elo_lim$home_win, k= num_folds)
for (k in 1:num_folds){
    # Split train and test
    index <- folds[[k]]
    train <- matches_elo_lim %>% slice(-index)
    test <- matches_elo_lim %>% slice(index)

    # Build models (Comparing a single tree, glm, and bagged random forest)
    glm.elo <- glm(home_win~., data = train, family = "binomial")

    # trees! From ISLRv2
    tree.elo <- tree(home_win~., data = train)
    forest.elo.bagged <- randomForest(home_win~., data = train,
                              mtry= p,
                              importance = TRUE,
                              ntree = 5000)
    forest.elo <- randomForest(home_win~., data = train,
                              mtry = floor(sqrt(p)),
                              importance = TRUE,
                              ntree = 5000)
    # boost.elo <- gbm(home_win ~., data = train,
    #                  distribution = "bernoulli",
    #                  n.trees = 5000, interaction.depth = 4)

    # Try to improve using penalization: glmnet
        # predictors matrix
    x <- model.matrix(home_win ~ ., train)[,-1]
      # Response vector
    y <- train$home_win
      # train models
    lasso_model <- cv.glmnet(x, y, alpha = 1, family = "binomial")
    ridge_model <- cv.glmnet(x, y, alpha = 0, family = "binomial")
    enet_model <- cv.glmnet(x, y, alpha = 0.5, family = "binomial")

    # Get accuracies
      #GLM
    yhat.glm.probs <- predict(glm.elo, newdata = test)
    yhat.glm <- ifelse(yhat.glm.probs > 0.5, 1, 0)
    test.error.glm <-c(test.error.glm, mean(yhat.glm == test$home_win))

      # bagged rf
    yhat.bag <- predict(forest.elo.bagged , newdata = test, type = "class")
    test.error.bagged_rf <-c(test.error.bagged_rf,
                             mean (yhat.bag == test$home_win))

      # Regular rf
    yhat.rf<- predict(forest.elo , newdata = test, type = "class")
    test.error.rf <-c(test.error.rf, mean (yhat.rf == test$home_win))

      # Single Tree
    yhat <- predict(tree.elo , newdata = test, type = "class")
    test.error.tree <- c(test.error.tree, mean (yhat == test$home_win))
```

```
        #   # Boosted rf
        # yhat.boost <- predict(boost.elo, newdata=test, type="class")
        # test.error.boost <- c(test.error.boost,
        #                       mean(yhat.boost == test$home_win))

          # glmnet
        x_test <- model.matrix(home_win ~ ., test)[,-1]

        yhat_prob_lasso <- predict(lasso_model, newx = x_test,
                                   type = "response", s = "lambda.min")
        yhat_prob_ridge <- predict(ridge_model, newx = x_test,
                                   type = "response", s = "lambda.min")
        yhat_prob_enet <- predict(enet_model, newx = x_test,
                                  type = "response", s = "lambda.min")

        yhat_prob_lasso <- ifelse(yhat_prob_lasso > 0.5, 1, 0)
        yhat_prob_ridge <- ifelse(yhat_prob_ridge > 0.5, 1, 0)
        yhat_prob_enet <- ifelse(yhat_prob_enet > 0.5, 1, 0)

        test.error.lasso <- c(test.error.lasso,
                              mean(yhat_prob_lasso == test$home_win))
        test.error.ridge <- c(test.error.ridge,
                              mean(yhat_prob_ridge == test$home_win))
        test.error.enet <- c(test.error.enet,
                             mean(yhat_prob_enet == test$home_win))
    }
  }
}
```

```
varImpPlot(forest.elo.bagged, n.var = 10, cex=0.5)
```

```
varImpPlot(forest.elo, n.var = 10, cex=0.5)
```

```
# Cross validation indicates a tree of size 2 is the preferred model
cv.elo.tree <- cv.tree(tree.elo)
plot(cv.elo.tree$size , cv.elo.tree$dev , type = "b")
```

```
# So prune tree and get accuracy
tree.elo.clipped <- prune.tree(tree.elo, best = 2)
yhat <- predict(tree.elo.clipped , newdata = test, type = "class")
pruned.tree.accuracy <- mean(yhat == test$home_win)

plot(tree.elo.clipped)
text(tree.elo.clipped, pretty = .05)
```

```
# calculate Elo accuracy
yhat.elo <- ifelse(elo_df$pr_home_win > 0.5, 1, 0)
elo.accuracy <- mean(yhat.elo == elo_df$home_win)

# Build into pretty table
accuracies_df <- cbind(c("Whole Single Tree", "Pruned Single Tree",
                         "Random Forest", "Bagged Random Forest",
```

```
                          "GLM", "LASSO", "Ridge", "Elastic Net", "Elo"),
      rbind(round(mean(test.error.tree), 5),
            round(pruned.tree.accuracy, 5),
            round(mean(test.error.rf), 5),
            round(mean(test.error.bagged_rf), 5),
            round(mean(test.error.glm), 5),
            round(mean(test.error.lasso), 5),
            round(mean(test.error.ridge), 5),
            round(mean(test.error.enet), 5),
            round(elo.accuracy, 5)
            ),
      rbind(round(sd(test.error.tree), 5),
            round(pruned.tree.accuracy, 5),
            round(sd(test.error.rf), 5),
            round(sd(test.error.bagged_rf), 5),
            round(sd(test.error.glm), 5),
            round(sd(test.error.lasso), 5),
            round(sd(test.error.ridge), 5),
            round(sd(test.error.enet), 5),
            round(elo.accuracy, 5)
            )) %>% as.data.frame()

accuracies_df_pretty <- accuracies_df %>%
  rename(Model = V1, Accuracy = V2, s.d. = V3 ) %>%
  mutate(Model = as.factor(Model),
         Accuracy = as.numeric(Accuracy),
         s.d.= as.numeric(s.d.)) %>%
  arrange(desc(Accuracy))

accuracies_df_pretty %>% select(Model, Accuracy) %>% kable()
```

```
accuracies_df_pretty %>% filter(Accuracy != s.d.)%>%
  mutate(Model = reorder(Model, Accuracy)) %>%
  ggplot(aes(x = Model, y = Accuracy)) +
  geom_boxplot(aes(color = Model)) +
  geom_errorbar(aes(ymin = Accuracy - s.d., ymax = Accuracy + s.d.), width = 0.3) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        legend.position = "none")
```

```
# K-fold CV, still on the matches_elo_lim dataset
# There are 21 predictors
ntree_grid <- (seq(6000, 10000, by = 500))
mtry_grid <- c(seq(10,18,by=2))
num_folds = 2

accuracies.matrix <- data.frame()

for (i in mtry_grid){
  for (j in ntree_grid){
    accuracies <- numeric()

    folds <- caret::createFolds(matches_elo_lim$home_win, k= num_folds)
```

```
    for(k in 1:num_folds){
      #cat(i, " ", j, "Fold: ", k, "\n")
      # split train and test
              # Split train and test
      index <- folds[[k]]
      train <- matches_elo_lim %>% slice(-index)
      test <- matches_elo_lim %>% slice(index)

      rf_fit <- randomForest(home_win ~ ., data = train,
                             mtry = i,
                             ntree = j)

      preds <- predict(rf_fit, newdata = test, type = "response")
      accuracy <- mean(preds == test$home_win)
      accuracies <- c(accuracies, accuracy)

    }
   accuracies.matrix <- rbind(accuracies.matrix, cbind(i, j, accuracies))
  }
}
```

```
accuracies.matrix%>% rename(nmtry = i,
                            ntree = j,
                            accuracy = accuracies)%>%
  arrange(desc(accuracy)) %>% head(5) %>% kable()
```

```
varImpPlot(rf_fit, n.var = 10, cex=0.8)
```

```
enet.coefs <- coef(enet_model, s="lambda.min") %>% as.matrix()

as.data.frame(enet.coefs) %>%
  filter(lambda.min >0) %>%
  arrange(desc(lambda.min)) %>%
  rename(Coefficient = lambda.min) %>% kable()
```