Predicting NFL wins using Machine Learning

Angus Watters

August 03, 2022

# **Introduction**

My goal with this project was to try to answer the age old sports question of **“So, who is supposed to win this game?”**

I wanted to see if I could use historic information about a team’s offensive and defensive performances to garner an accurate prediction on whether or not they will win their upcoming game.

All data ingestion, processing, analysis, and modeling was done in R, and I leaned heavily on the tidyverse and tidymodels ecosystems to get this done.

# **Data Retrieval**

I pulled NFL play-by-play data together using the nflFastR R package. I then did some data cleaning and wrangling tasks to get the data into a usable format.

I started by getting the full play-by-play data sets from 1999-2021. Each season there are some ~50,000 plays run, and for each play, there are 372 columns of data giving information about what happened on that play.

**Note:** While exploring the data and creating the features I wanted to use in the model, I created a host of utility helper functions that work off of the play-by-play output data from the nflFastR package. Those helper functions can be found [here](https://github.com/anguswg-ucsb/nfl_wins/blob/main/utils/utils.R).

The function below takes in the NFL play-by-play returned from the nflFastR::nflfastR::load\_pbp() function, and aggregates the data to game level team statistics.

In the end I used data from the 1999-2021 seasons, however I completed omitted the entire 2021 season to use as a holdout set of data to test my models against. Another thing to note is that I ended up using only data from the perspective of the home team.

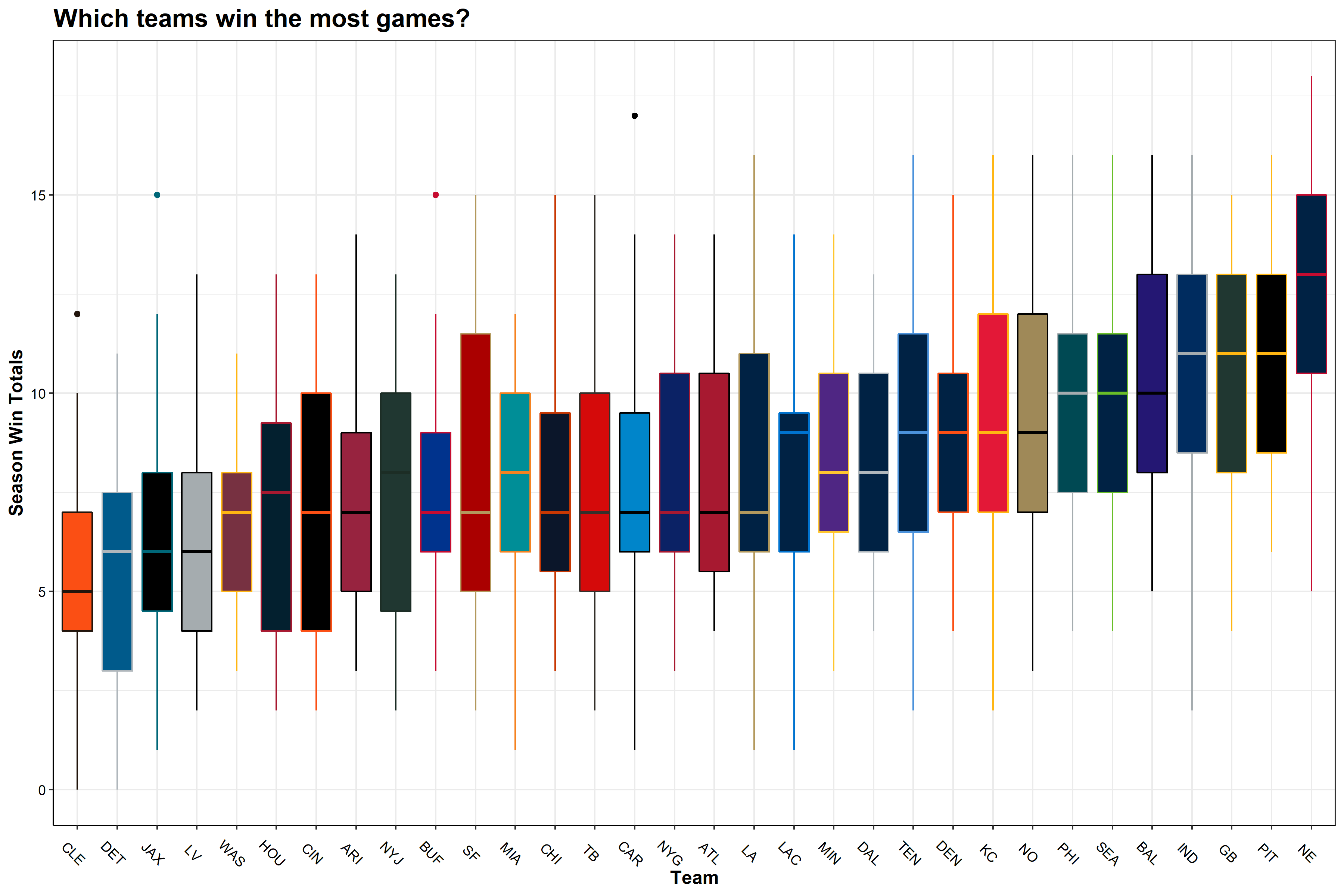
Initially I had been running models with 2 data points for each game, one being the home team perspective and the other being the away team perspective. I pretty quickly ran into some major overfitting in my models and realized **it was a better idea to only use data from the home team.**

# **Exploratory Data Analysis**

Let’s dig into some of the data and see what we can learn about the NFL and the data we are working with.

## Season win totals

First thing I wanted to see was how many games on average each NFL franchise won per season between 1999 and 2021. The box plot below shows the average season win totals on the Y axis and the NFL franchises on the x axis with the x axis arranged by mean annual win total. The boxes are colored to match the color scheme for each NFL team.



**TLDR;** Patriots good, Browns bad (sorry Cleveland!)

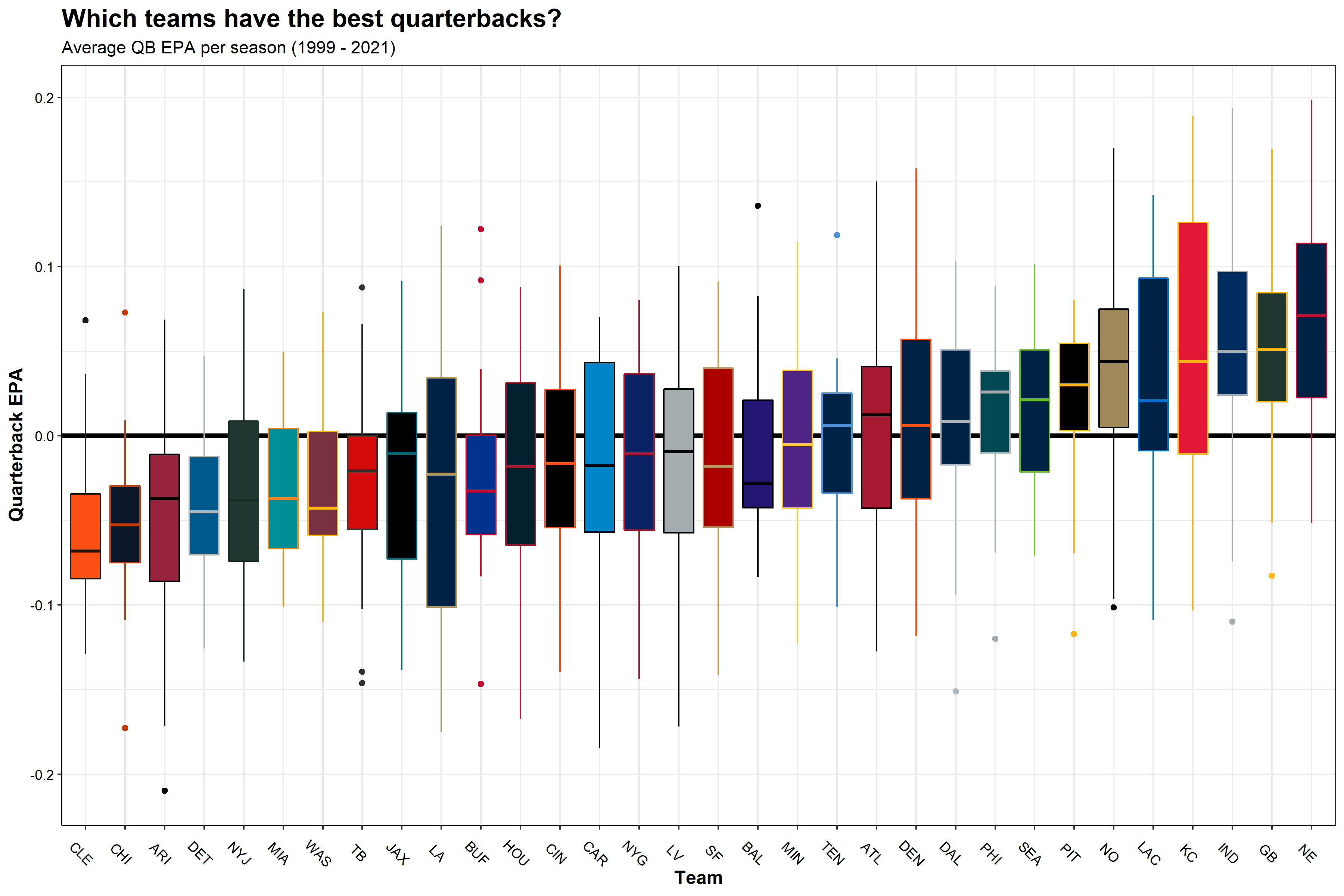
This confirms what I already know from my experience watching the NFL. The New England Patriots are leading the league with an average of 12.6 wins per season. The Patriots are followed by the Steelers, Packers, Colts, Ravens, all teams that have seen consistent success over the last 23 years. At the bottom of the league are the Browns, Lions, Jaguars, and the Raiders.

## QB performances

Next I wanted to see which, NFL team’s had the best quarterbacks over the last 23 seasons. Quarterback EPA is a metric that determines how likely a team is to score points as a result of a play, and more specifically the quarterbacks effect on that play.

**If a QB does something good (long completion, run for a first down), then there team is more likely to score points on the ensuing play, thus Expected points were added (EPA is positive). On the other hand if a QB does something bad (incompletion, takes a big sack), then there team is less likely to score points on the ensuing play and expected points were removed (EPA is negative).**

Below is a plot showing the for each team, the average QB EPA across the all seasons.



**TLDR;** Tom Brady good, bad

This makes sense, teams like the Patriots, Packers, and Colts are at the top of the league in terms of average QB EPA seeing as though these teams had great quarterbacks for most of these years, Tom Brady, Brett Favre/Aaron Rodgers, and Peyton Manning/Andrew Luck, respectively. This plot matches very closely with the above season win totals boxplot. Safe to say, good quarterback play positively impacts winning games.

## Relating end of season win totals to team metrics

Next thing I wanted to see what the end of season win totals for each team across all 23 seasons, looked like when plotted against some average team level statistics for that season. The plot below shows 6 team metrics with the X axis displaying end of season win totals and the Y axis being the season average for each team.

**Note:** The Y axis values are not meant to be compared across plots as they are in different units. The purpose of this figure is to show the relationship between team metrics and end-of-season win totals From top left to bottom right:

Variable

Y Axis

% Time of possesion

Average percent of the game the team had possession of the ball

QB EPA

EPA value w/ negative values indicating a negative impact on scoring

Score Differential

Average Score Differential

Scoring Drive

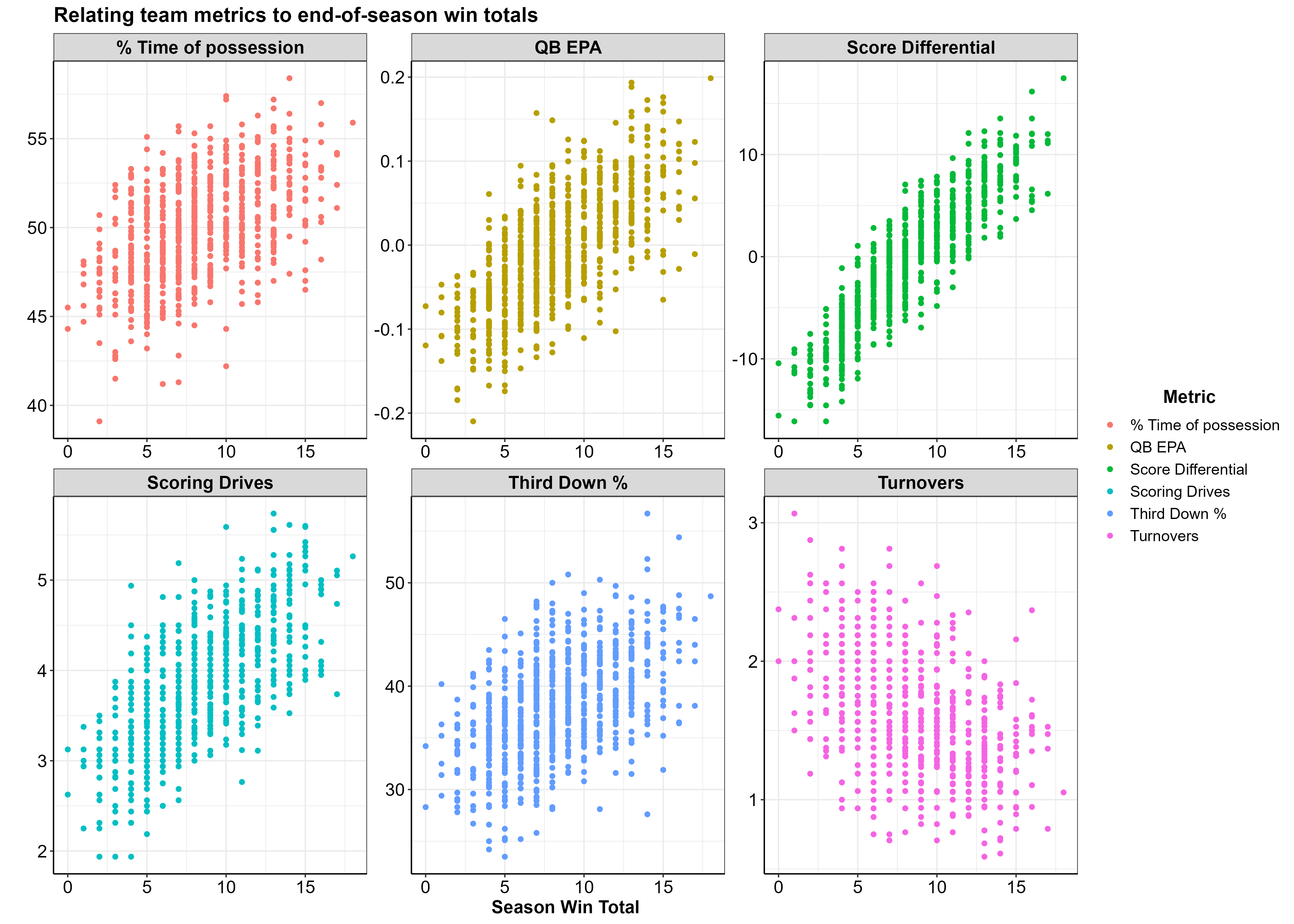
Number of scoring drives per game

Third Down %

Average percent of third downs converted to first downs

Turnovers

Number of turnovers per game

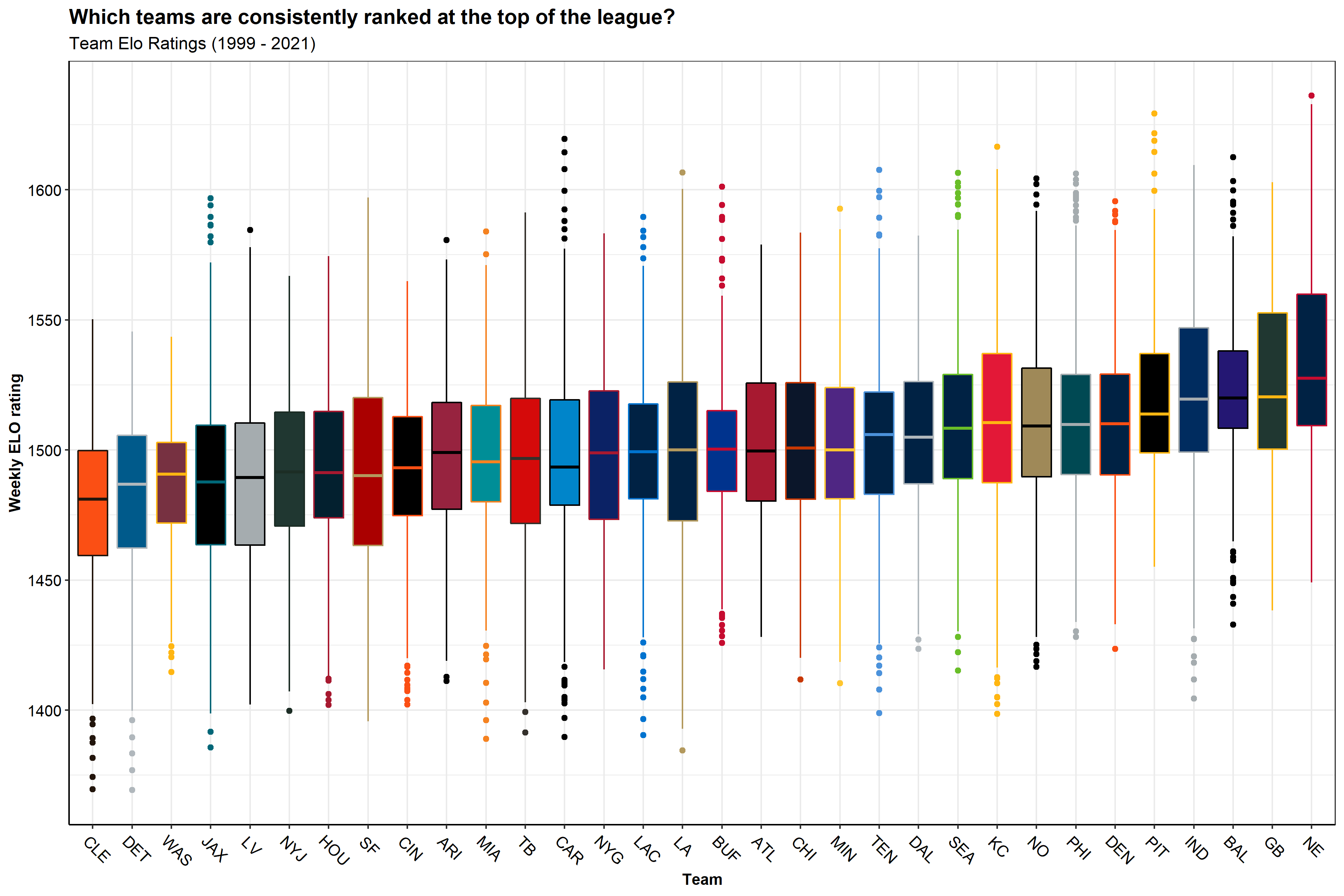


Important takeaways from these plots:

* Greater % time of possession, the more total wins at the end of the season
* Higher QB EPA, the more total wins at the end of the season
* More positive score differential, the more total wins at the end of the season
* Higher number of scoring drives per game, the more total wins at the end of the season
* Higher third down %, the more total wins at the end of the season
* Lower number of turnovers per game, the more total wins at the end of the season

## Elo Rating

I created an Elo rating system for each season to create a metric that keeps track of a teams rank relative to the rest of the league. Elo rating systems were first created to rate chess players and are now commonly used in many sports such as American Football, baseball, basketball, etc. Special thanks to the creators of the [elo](https://eheinzen.github.io/elo/) package, your package made my life a lot easier. Here is more information on [Elo Rating Systems](https://en.wikipedia.org/wiki/Elo_rating_system) and its inventor [Arpad Elo](https://en.wikipedia.org/wiki/Arpad_Elo).



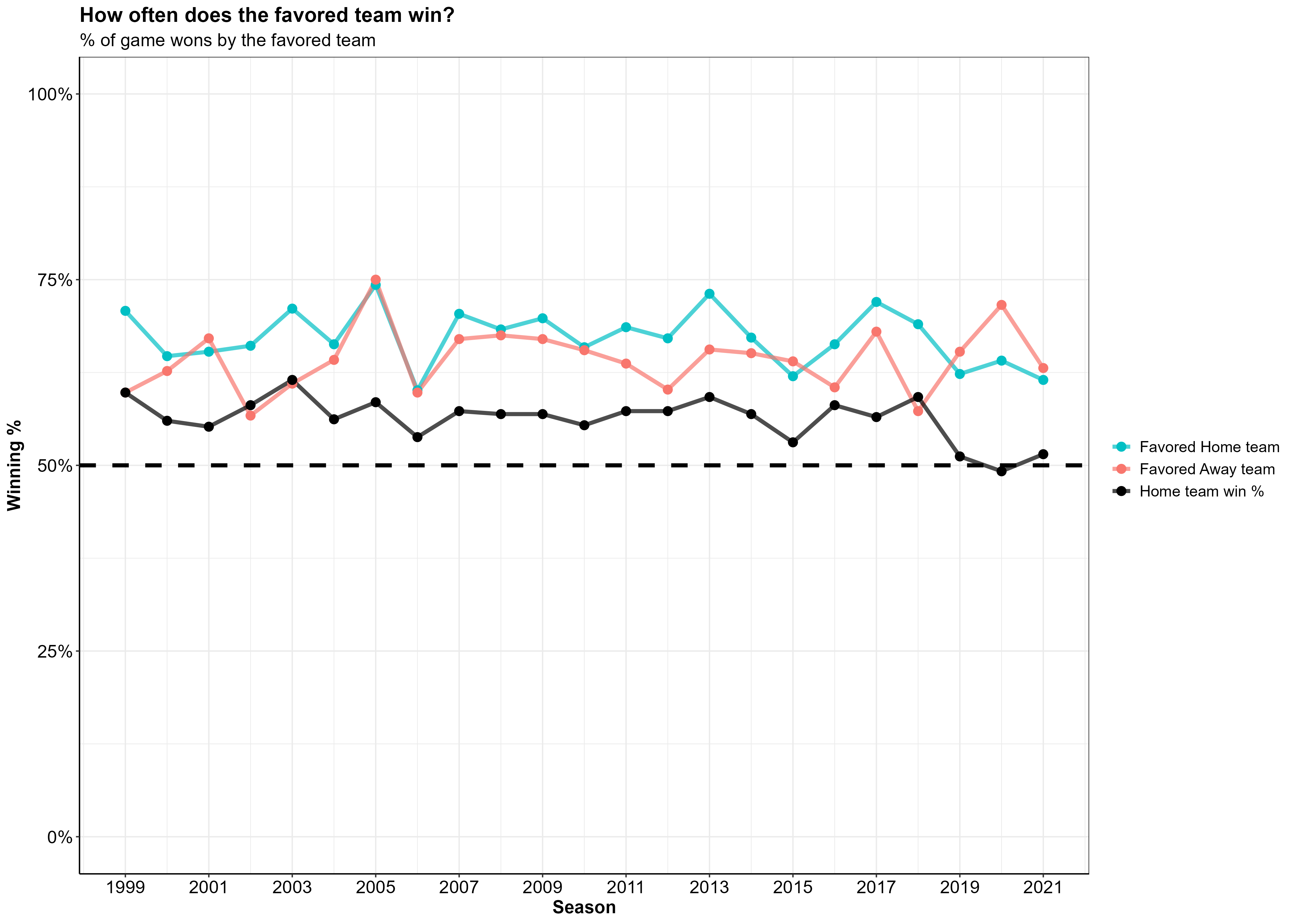
Sadly for Browns fans, the same take home message as we saw before… Patriots good, Browns bad :’(

## Favored teams

Because I am going to make predictions on the outcome of NFL games I thought it best to see how the teams that were favored to win their game (according to Las Vegas) actually did in those games. Specifically, I wanted to see how often does a favored home team win? and a favored away team? and how often do home teams win overall?

I’m curious about how often the home team wins, furthermore, what percent of the time does the team favored by Vegas win? And how often does a favored home team win? and a favored away team?

The black line indicates the percent of games won by the home team per season. The aqua blue line show the percent of games that favored home teams won and the coral color line shows the percent of games that favored away teams won.



Overall, it looks like a favored home team is winning ~ 5-10% more of the games they’re favored in than a favored away team does. Anything else would be suprising seeing as a though a favored away team is at an inherit disadvantage in there game compared to a favored home team (because they’re ***not*** the home team).

# **Feature engineering and selection**

I decided I wanted to focus the model features on **offensive metrics** and a handful of **other off-field factors** that I suspected might influence a teams likelihood to win their upcoming game. For all the metrics I describe below, in order to capture both the home team and the away team, I created two sets of the same metric, one for the home team and one for the away team.

**Note:** Later on you will see features whose names start with **“opp\_”**, which just means that feature is representing the away team.

## Offense

Originally, I had created many more features (including time of possession as a percent of the game, points scored in each quarter, turnovers by the defense and many more) but as I made some initial models I found that all of these were not necessary. The features you see below are the features that had the most predictive power after I trimmed away some of the fat.

Variable

Model Variable

Type

Description

Score Differential

score\_diff

Numeric

Average score differential throughtout the game

Third Down Conversion Rate

third\_down\_pct

Numeric

Percent of 3rd downs converted by offense

Turnovers

turnovers

Numeric

Total Turnovers by the offense (Fumbles lost + Interceptions

QB EPA

qb\_epa

Numeric

Average Quarterback EPA across the whole game

Scoring Drive

score\_drives

Numeric

Number of drives in a game that result in a score (touchdown/fieldgoal)

## Off field factors

I included a handful of other information known prior to the game such as the number of days of rest between games, the team’s overall, home, and away winning percentages, and the team’s ranking via an Elo rating.

Variable

Model Variable

Type

Description

Rest days

rest\_days

Numeric

Numeric variable for the amount of rest the team had between games

Win %

win\_pct

Numeric

Percent of all games won

Home Win %

home\_win\_pct

Numeric

Percent of home games won

Away Win %

away\_win\_pct

Numeric

Percent of away games won

Elo rating

elo

Numeric

Elo rating per week in each season, relative to rest of the league

Once all of the above variables were wrangled, cleaned, and aggregated from the NFL play-by-play data, I was left with a data frame that had 1 row for every home game played during the season, with two columns for each of the variables shown in the tables above (1 for the home team and 1 for the away team).

nfl\_data %>%   
 head(10)

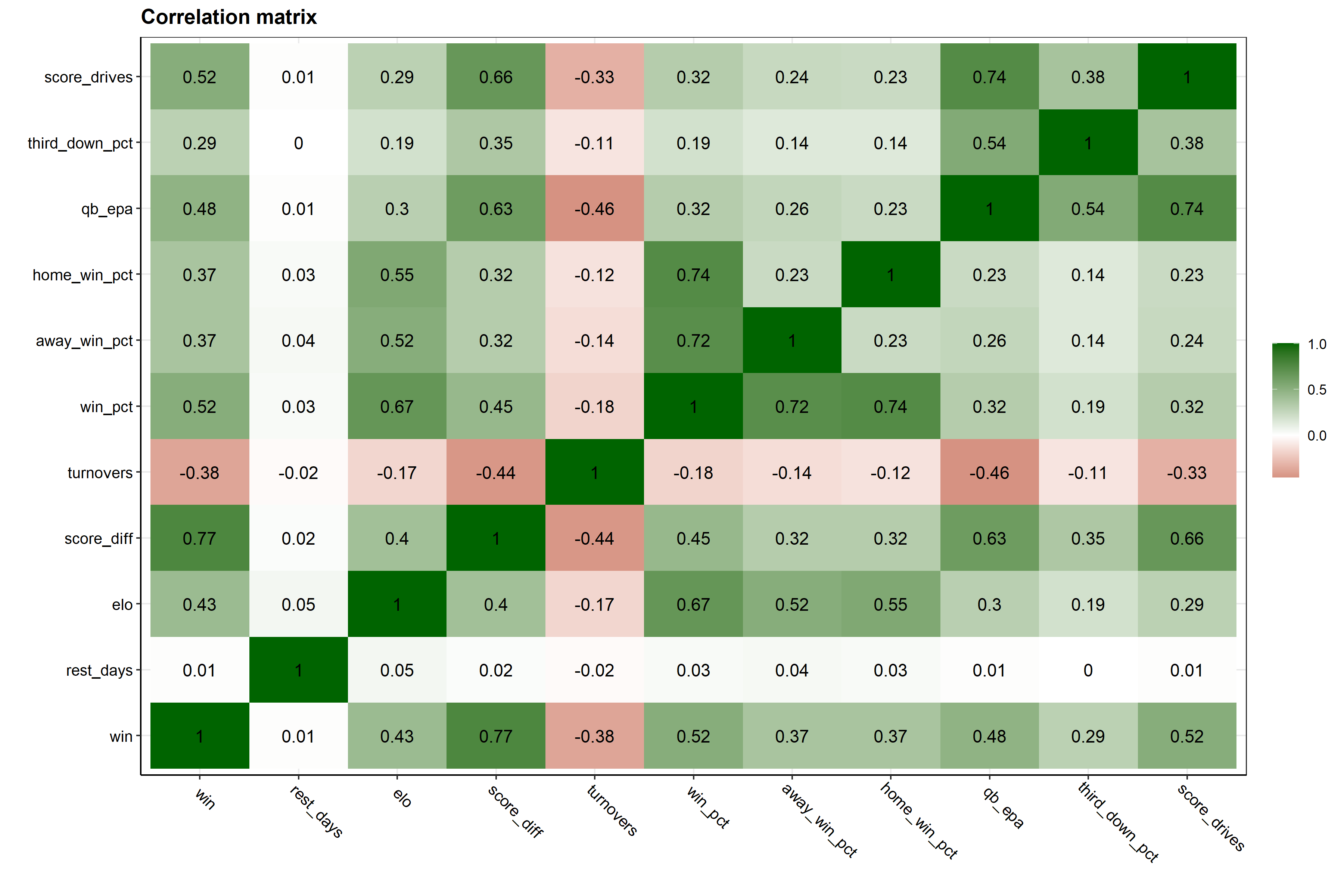
## # A tibble: 10 × 26  
## season week game\_id team oppon…¹ win rest\_…² opp\_r…³ elo opp\_elo  
## <dbl> <dbl> <chr> <chr> <chr> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 1999 1 1999\_01\_ARI\_P… ARI PHI 1 7 7 1472. 1472.  
## 2 1999 2 1999\_02\_ARI\_M… ARI MIA 0 7 7 1462. 1462.  
## 3 1999 3 1999\_03\_SF\_ARI ARI SF 0 8 8 1490 1490   
## 4 1999 4 1999\_04\_ARI\_D… ARI DAL 0 6 6 1481. 1481.  
## 5 1999 5 1999\_05\_NYG\_A… ARI NYG 1 7 7 1500. 1500.  
## 6 1999 6 1999\_06\_WAS\_A… ARI WAS 0 7 7 1490. 1490.  
## 7 1999 8 1999\_08\_NE\_ARI ARI NE 0 14 14 1481. 1481.  
## 8 1999 9 1999\_09\_ARI\_N… ARI NYJ 0 7 7 1463. 1463.  
## 9 1999 10 1999\_10\_DET\_A… ARI DET 1 7 7 1491. 1491.  
## 10 1999 11 1999\_11\_DAL\_A… ARI DAL 1 7 7 1501. 1501.  
## # … with 16 more variables: score\_diff <dbl>, opp\_score\_diff <dbl>,  
## # turnovers <dbl>, opp\_turnovers <dbl>, win\_pct <dbl>, away\_win\_pct <dbl>,  
## # home\_win\_pct <dbl>, opp\_win\_pct <dbl>, opp\_away\_win\_pct <dbl>,  
## # opp\_home\_win\_pct <dbl>, qb\_epa <dbl>, opp\_qb\_epa <dbl>,  
## # third\_down\_pct <dbl>, opp\_third\_down\_pct <dbl>, score\_drives <dbl>,  
## # opp\_score\_drives <dbl>, and abbreviated variable names ¹​opponent,  
## # ²​rest\_days, ³​opp\_rest\_days  
## # ℹ Use `colnames()` to see all variable names

## Corrolation

Let’s take a look at how our game level summary data relates to winning a game.

Below is a correlation matrix that highlights the relationship between variables in the data. We are most interested in the bottom row of this plot, which indicates the correlation between a team winning there game and all the other variables in the data.

Darker green colors indicate positive correlations while darker red colors indicate negative correlations.



We see that Elo rating has a relatively strong positive correlation with winning, which makes a lot of sense, a higher Elo rating (higher ranked team) is more likely to end up with a win. The average score differential for a team also makes good sense, if a team on average has a higher score differential (they score more points than are scored against them) then it’s logical that they would end up winning more games. The turnover variable has a strong negative correlation with winning, so if a team on average has fewer turnovers, they are more likely to end up winning the game.

At this point in my analysis, I was working with ***a posteriori*** data, so data that was collected as a product of what happened on the field that week. If I want to make any useful predictions about the coming week, I would need to conjur up some ***a priori*** data.

## Cumulative Averages

To capture how well a team is doing throughout each season, I created lagged cumulative means for all of the above variables. Such that, for each week a team plays a game, the cumulative mean of all the ***preceeding*** weeks of data are calculated for every variable leading into the upcoming slate of games.

Below is the general method I used to get these lagged Cumulative averages values. Note, for some variables like winning percentage, the lagged cumulative averages was ***not*** calculated and rather only the lagged value were used. In the example code below, the lagged winning percentage, and the lagged cumulative average QB EPA is calculated for the Arizona Cardinals 2014 season.

nfl %>%   
 dplyr::filter(season == 2014, team == "ARI") %>%   
 dplyr::select(season, week, team, win, win\_pct, qb\_epa) %>%   
 dplyr::group\_by(season, team) %>%   
 dplyr::arrange(season, week, .by\_group = T) %>%   
 dplyr::mutate(  
 across(c(win\_pct), ~dplyr::lag(.x), .names = "{col}\_lag"),  
 across(c(qb\_epa), ~dplyr::lag(dplyr::cummean(.x)), .names = "{col}\_lag")  
 ) %>%   
 dplyr::mutate(across(c(win\_pct:qb\_epa\_lag), round, 2)) %>%   
 dplyr::ungroup() %>%   
 dplyr::relocate(season, week, team, win, win\_pct, win\_pct\_lag, qb\_epa, qb\_epa\_lag)

## # A tibble: 17 × 8  
## season week team win win\_pct win\_pct\_lag qb\_epa qb\_epa\_lag  
## <dbl> <dbl> <chr> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 2014 1 ARI 1 1 NA -0.1 NA   
## 2 2014 2 ARI 1 1 1 0.03 -0.1   
## 3 2014 3 ARI 1 1 1 0.2 -0.04  
## 4 2014 5 ARI 0 0.75 1 -0.16 0.04  
## 5 2014 6 ARI 1 0.8 0.75 -0.01 -0.01  
## 6 2014 7 ARI 1 0.83 0.8 0.09 -0.01  
## 7 2014 8 ARI 1 0.86 0.83 -0.03 0.01  
## 8 2014 9 ARI 1 0.88 0.86 -0.02 0   
## 9 2014 10 ARI 1 0.89 0.88 -0.11 0   
## 10 2014 11 ARI 1 0.9 0.89 -0.06 -0.01  
## 11 2014 12 ARI 0 0.82 0.9 -0.34 -0.02  
## 12 2014 13 ARI 0 0.75 0.82 -0.09 -0.05  
## 13 2014 14 ARI 1 0.77 0.75 0 -0.05  
## 14 2014 15 ARI 1 0.79 0.77 -0.14 -0.05  
## 15 2014 16 ARI 0 0.73 0.79 -0.25 -0.05  
## 16 2014 17 ARI 0 0.69 0.73 0.09 -0.07  
## 17 2014 18 ARI 0 0.65 0.69 -0.4 -0.06

If you take a look at ARI’s week 5 game (row 5), the **win\_pct\_lag** column for week 5 is represented by the teams **win\_pct** from the previous week, week 4 in this case, or in other words, the team’s winning percentage ***leading into*** the week 5 match up. On the other hand, the **qb\_epa\_lag** column was calculated using the mean of **qb\_epa** from weeks 1-4.

I did this lagging process across all of my variables such that for each row with a game outcome (win/loss), that team has variables representing their performances up until that point in the season.

Now our data is set up so that all the information used to inform our prediction is data that would be available to us ***prior*** to the upcoming week of games. This is important because information such as the number of turnovers during the game is not information we would have prior to the game, and thus can’t be used as a predictor in our model, we can only use historic data to inform our predictions.

# **Modeling**

I decided I wanted to run my data across a panel of 6 different models and see how they all perform against each other

* [Logistic Regression](https://parsnip.tidymodels.org/reference/details_logistic_reg_glmnet.html)
* [K-nearest neighbors](https://parsnip.tidymodels.org/reference/details_nearest_neighbor_kknn.html)
* [Gradient Boosted Decision Trees](https://parsnip.tidymodels.org/reference/details_boost_tree_xgboost.html)
* [Multilayer Perceptron](https://parsnip.tidymodels.org/reference/details_mlp_nnet.html)
* [Radial basis function Support Vector Machines](https://parsnip.tidymodels.org/reference/details_svm_rbf_kernlab.html)
* [Polynomial Support Vector Machines](https://parsnip.tidymodels.org/reference/details_svm_poly_kernlab.html)

## Data Budget/Splits

The first step was to split up our data into testing and training splits (75% training, 25% testing). When I split my data, I chose to stratify the data by the binary win/loss value to ensure there was the same proportion of wins and losses in our training and testing data splits.

# Set random seed  
set.seed(234)  
  
# Partition training/testing data, stratified by win/loss  
nfl\_split <- rsample::initial\_split(nfl\_df, strata = win)  
  
# training data split - 75%  
nfl\_train <- rsample::training(nfl\_split)  
  
# testinng data split - 25%  
nfl\_test <- rsample::testing(nfl\_split)

## <Training/Testing/Total>  
## <4115/1373/5488>

## Data Preprocessing

Using the recipes package, I made ID variables for information about the game and when it happened. I then applied some recipe steps to my training data. Because I decided to only use the home teams data, this meant that there was going to be more wins than losses in my dataset because the home team tends to win more often then the away team (56% wins / 44% losses). To account for the slight imbalance of wins to losses in the data, I chose to use the themis package to upsample my data using the themis::step\_smote() function.

### Recipe steps

* step\_zv is applied to all predictors to remove all variables with only one variable (zero-variance)
* step\_normalize is used to normalize all the numeric predictors in the training data so that they all have a standard deviation of 1 and a mean of 0
* step\_smote is applied to our dependent **win** variable to fix the class imbalance due to having slightly more wins than losses in the data. The step\_smote function works by upsampling the minority class and generating new, synthetic data by using the nearest-neighbors of the minority class instances.

Below I create the two recipe steps that we will apply to our training data before modeling.

# Base recipe  
base\_recipe <-   
 recipes::recipe(  
 formula = win ~ .,   
 data = nfl\_train  
 ) %>%   
 recipes::update\_role(  
 game\_id, team, opponent, season, week, new\_role = "ID"  
 )   
  
# Normalize, SMOTE algo upsampling recipe  
norm\_smote\_recipe <-   
 base\_recipe %>%   
 recipes::step\_zv(recipes::all\_predictors()) %>%   
 recipes::step\_normalize(recipes::all\_numeric\_predictors())  
 themis::step\_smote(win, over\_ratio = 0.9, skip = T)  
   
# zero variance SMOTE algo upsampling recipe  
zv\_smote\_recipe <-   
 base\_recipe %>%   
 recipes::step\_zv(all\_predictors()) %>%   
 themis::step\_smote(win, over\_ratio = 0.9, skip = T)

## Cross-validation folds

I created 10 cross-validation folds from the training data. As I did with the initial training/testing split, I made sure to create **stratified resamples using the win/loss variable to ensure the same proportion of wins and losses appear in CV folds as the do in the original data.**

# Set random seed   
set.seed(432)  
  
# Cross-validation folds  
nfl\_folds <- rsample::vfold\_cv(nfl\_train, v = 10, strata = win)

## workflowsets

Using the workflowsets I created a workflow\_set object containing each of the data preprocessing recipes and corresponding model specifications.

# Workflow set of candidate models  
nfl\_wfs <-  
 workflowsets::workflow\_set(  
 preproc = list(  
 kknn\_rec = norm\_smote\_recipe,  
 glmnet\_rec = norm\_smote\_recipe,  
 xgboost\_rec = zv\_smote\_recipe,  
 nnet\_rec = norm\_smote\_recipe,  
 svm\_poly\_rec = norm\_smote\_recipe,  
 svm\_rbf\_rec = norm\_smote\_recipe  
 ),  
 models = list(  
 knn = knn\_spec,  
 glmnet = glmnet\_spec,  
 xgboost = xgboost\_spec,  
 nnet = nnet\_spec,  
 svm\_poly = svm\_poly\_spec,  
 svm\_rbf = svm\_spec  
 ),  
 cross = F  
 )

## Tuning Hyperparameters

The next step was to tune our hyperparameters for each of models. Using the tune package I applied the tune::tune\_grid() function across my workflowset object containing my model recipes and specifications. 20 random candidate parameters were created using the dials::grid\_latin\_hypercube() function from the dials package

### Parallel processing

This step is the most time consuming and resource intensive process of a machine learning workflow. So, to speed up the process I run the tuning process across multiple cores on my computer, making use of my computer’s multiple processors.

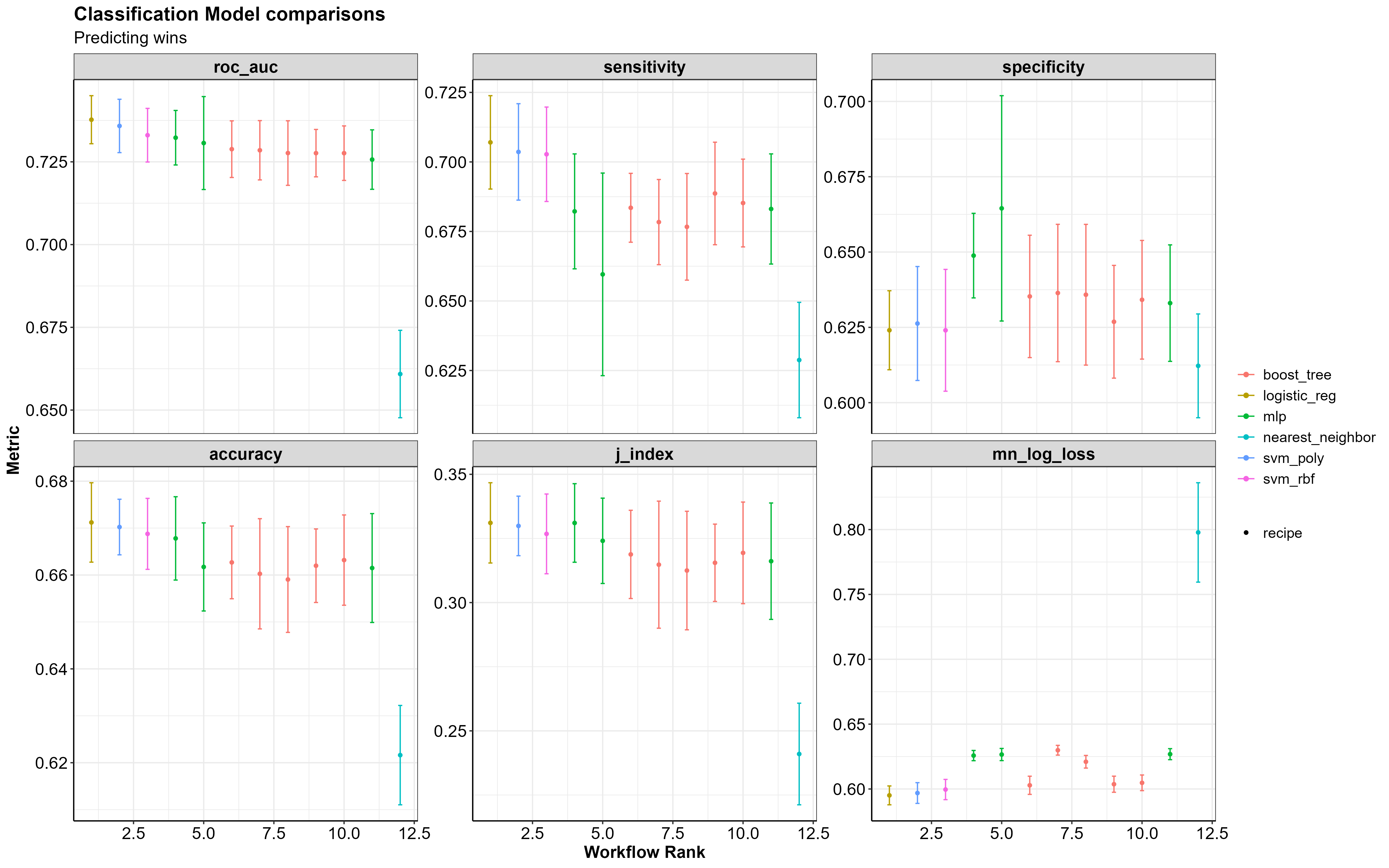
### Racing methods

To get another boost in processing time, I made use of the tune\_race\_anova from the finetune package, which makes use of a repeated measure ANOVA model to iterativily eliminates tuning parameters that are unlikely to yield the best results. Combining **parrallel processing** and **racing methods** can improve processing times by [~35 fold](https://finetune.tidymodels.org/reference/tune_race_anova.html).

# Choose metrics  
my\_metrics <- yardstick::metric\_set(roc\_auc, pr\_auc, accuracy, mn\_log\_loss)  
  
# Set up parallelization, using computer's other cores  
parallel::detectCores(logical = FALSE)  
modeltime::parallel\_start(6, .method = "parallel")  
  
# Set Random seed  
set.seed(589)  
  
# Tune models in workflowset  
nfl\_wfs <-  
 nfl\_wfs %>%  
 workflowsets::workflow\_map(  
 "tune\_grid",  
 resamples = nfl\_folds ,  
 grid = 20,  
 metrics = my\_metrics,  
 control = tune::control\_grid(  
 verbose = TRUE,  
 save\_pred = TRUE),  
 verbose = TRUE  
 )  
  
 # Stop parrallelization  
modeltime::parallel\_stop()

# **Model Evaluation**

Now we can compare how all of our models did compared to one another and make a decision as to which one we want to use to make predictions. The plots below shows how the 6 models performed on the set of resampling data.



If you take a look at the **ROC AUC** and **mean log loss plots** the best performing models are the Logistic Regression (logistic\_reg), the Support Vector Machines (svm\_poly/svm\_rbf), the Multilayer Perceptron (mlp) and the Gradient Boosted Decision Trees (boost\_trees).

I decided to set aside the Multilayer Perceptron model due to the slight increase in mean log loss relative to the other top performing models.

In particular, the boosted trees looks to perform at approximately the same level asthe mlp model when it comes from correctly predicting wins (sensitivity) and correctly predicting losses (specificity), without the increase in log loss.

A standard threshold for log loss when it comes to binary predictions is ~0.69 because anything higher than that and you aren’t beating the probability of a 50-50 guess. The table below shows the mean log loss for each model. The best log loss you can have is 0 and the larger the number gets the worse the model is at predicting the correct outcome.

The table belows shows the performance metrics for how each model.

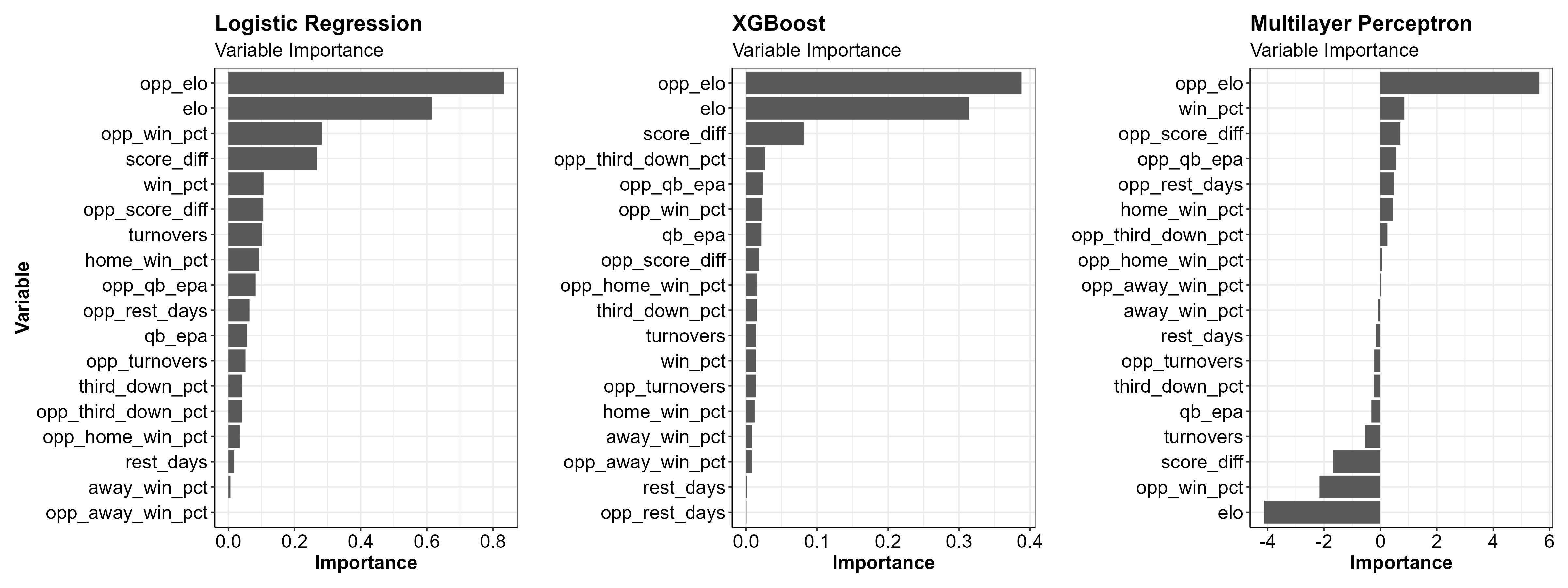
|  |  | **Resample Metrics** | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **model** | **rank** | **accuracy** | **mean\_log\_loss** | **roc\_auc** | **sensitivity** | **specificity** |
| logistic\_reg | 1 | 0.671 | 0.595 | 0.738 | 0.707 | 0.624 |
| svm\_poly | 2 | 0.670 | 0.597 | 0.736 | 0.704 | 0.626 |
| svm\_rbf | 3 | 0.669 | 0.600 | 0.733 | 0.703 | 0.624 |
| mlp | 4 | 0.668 | 0.626 | 0.732 | 0.682 | 0.649 |
| boost\_tree | 6 | 0.663 | 0.603 | 0.729 | 0.684 | 0.635 |
| nearest\_neighbor | 12 | 0.622 | 0.798 | 0.661 | 0.629 | 0.612 |

resamp\_metrics %>%   
 dplyr::arrange(mean\_log\_loss) %>%   
 ggplot() +  
 geom\_point(aes(x = reorder(model, mean\_log\_loss), y = mean\_log\_loss), size = 2) +  
 geom\_hline(yintercept = 0.69, size = 1.5)

## Variable Importance

Looking at at variable importance plots, generated from the vip package, can give us an idea of which variables are the most important to ours models.

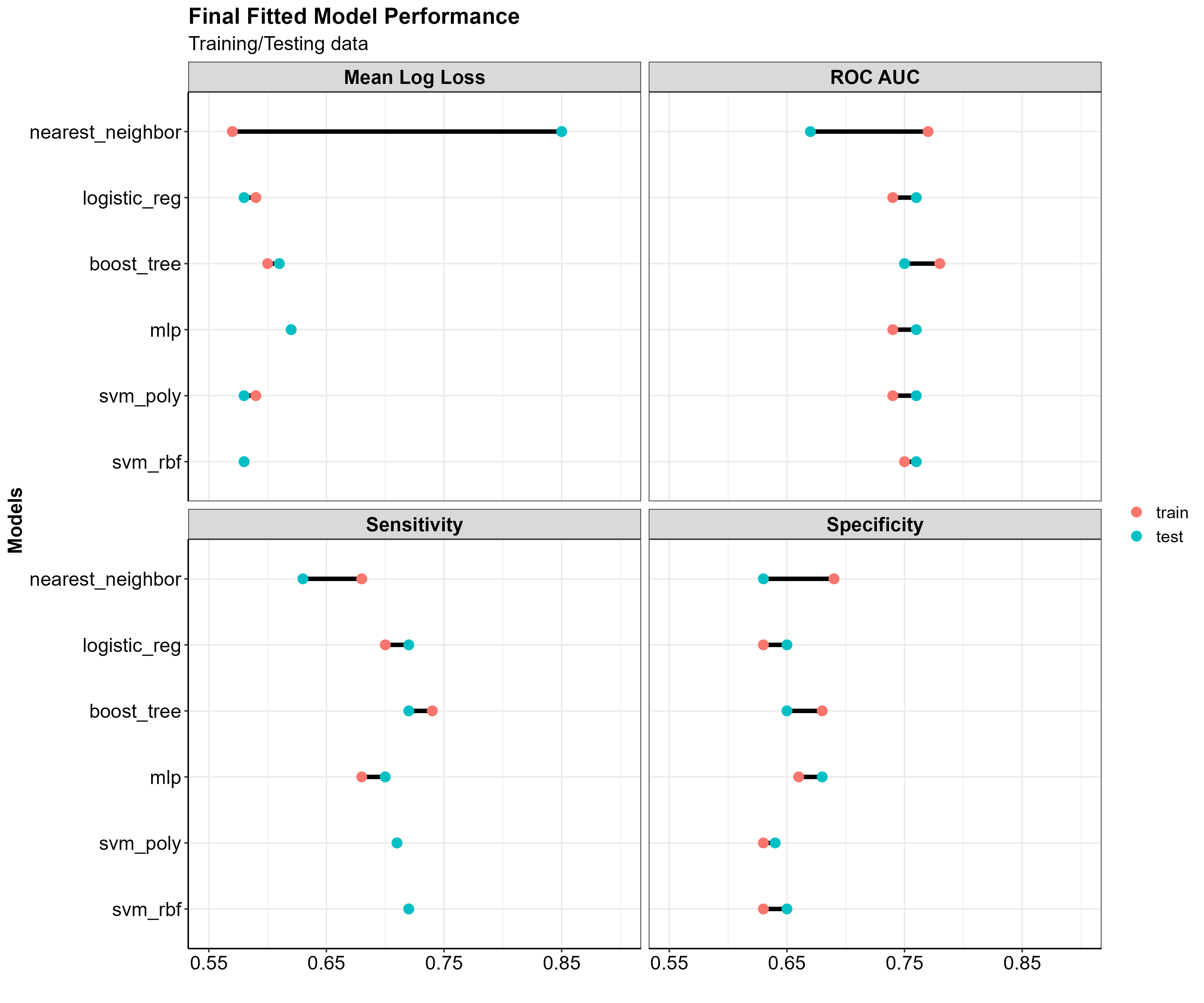
Below is the variable importance scores for the Logistic Regression, XGBoost, and Multilayer Perceptron models. If you want to read more about what makes up the variable importance score you can click here.



The most important variables in both models looks to be the team’s Elo ratings, the cumulative average score differential and the opponent’s winning percentage.

# **Model Performance on Test Data**

I then selected the tuning parameters for each model that had the best performance metrics on the data resamples. Using these model parameters, I refit each model for the last time using just the initial data split, fitting one last time on the training data and evaluating on the testing data. I then used the final fitted models to make predictions. Let’s check out the results!



And in table form…

|  |  | **Best model metrics** | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Data split** | **accuracy** | **mean\_log\_loss** | **roc\_auc** | **sensitivity** | **specificity** |
| nearest\_neighbor | train | 0.69 | 0.57 | 0.77 | 0.68 | 0.69 |
| nearest\_neighbor | test | 0.63 | 0.85 | 0.67 | 0.63 | 0.63 |
| logistic\_reg | train | 0.67 | 0.59 | 0.74 | 0.70 | 0.63 |
| logistic\_reg | test | 0.69 | 0.58 | 0.76 | 0.72 | 0.65 |
| boost\_tree | train | 0.71 | 0.60 | 0.78 | 0.74 | 0.68 |
| boost\_tree | test | 0.69 | 0.61 | 0.75 | 0.72 | 0.65 |
| mlp | train | 0.67 | 0.62 | 0.74 | 0.68 | 0.66 |
| mlp | test | 0.69 | 0.62 | 0.76 | 0.70 | 0.68 |
| svm\_poly | train | 0.68 | 0.59 | 0.74 | 0.71 | 0.63 |
| svm\_poly | test | 0.68 | 0.58 | 0.76 | 0.71 | 0.64 |
| svm\_rbf | train | 0.68 | 0.58 | 0.75 | 0.72 | 0.63 |
| svm\_rbf | test | 0.69 | 0.58 | 0.76 | 0.72 | 0.65 |

The nearest neighbor model sees a big drop in mean log loss between the training and testing data, besides this, the remaining models appear to perform similarly with both the training and testing data, which gives us some more confidence that these models would perform similarly in the real world.

## ROC Curves

Receiver Operator Characteristic curves, (aka ROC curves) are a way of understanding how well a model is predicting the True Positive events vs. the False Positive events.

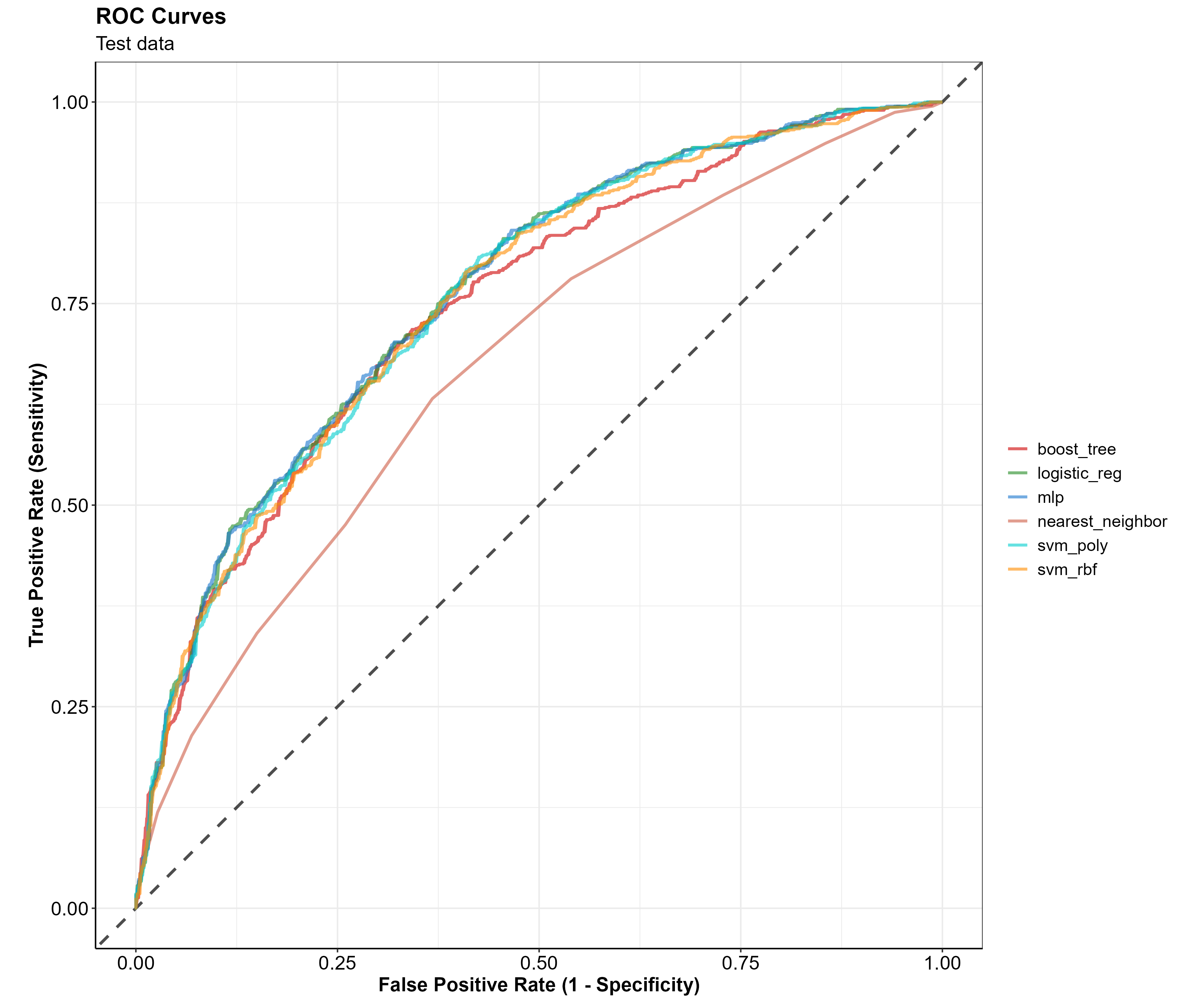
In our scenario, with models that are trying to predict if an NFL team will win, a **True Positive event would be that the model predicts a win and the team actually wins, while in the False Positive event the model predicts a win, and the team actually loses**.

The True Positive Rate (TPR) is on the Y axis and the False Positive Rate (FPR) is on the X axis.

The TPR, or **sensitivity**, along the Y axis indicates the probability that the model predicted a win and the team actually won. While FPR or, **1 - specificity**, along the X axis indicates the probability that the model predicted a win and the team actually lost.

The closer the ROC curve is to the top left corner of the plot the better the model is doing at correctly classifying the data. A perfect ROC Curve would be a 90 degree angle along the top left corner of the plot and would indicate that the model is able to perfectly classify the data correctly.

roc\_curve <-  
 roc\_df %>%  
 ggplot2::ggplot() +  
 ggplot2::geom\_abline(  
 lty = 2,   
 alpha = 0.7,  
 color = "black",   
 size = 1) +  
 ggplot2::geom\_line(  
 ggplot2::aes(  
 x = 1 - specificity,   
 y = sensitivity,  
 color = model),   
 alpha = 0.6,   
 size = 1  
 ) +  
 # ggplot2::scale\_color\_manual(values = RColorBrewer::brewer.pal(6, "Dark2")) +  
 ggplot2::scale\_color\_manual(values = c( "red3", "forestgreen", "dodgerblue3", "coral3", "cyan3", "darkorange")) +  
 ggplot2::coord\_equal() +  
 ggplot2::labs(  
 title = "ROC Curves",  
 subtitle = "Test data",  
 x = "False Positive Rate (1 - Specificity)",  
 y = "True Positive Rate (Sensitivity)",  
 col = "Models"  
 ) +  
 apatheme   
  
# ggsave(  
# here::here("img", "roc\_curve.png"),  
# roc\_curve,  
# width = 12,  
# height = 10  
# )



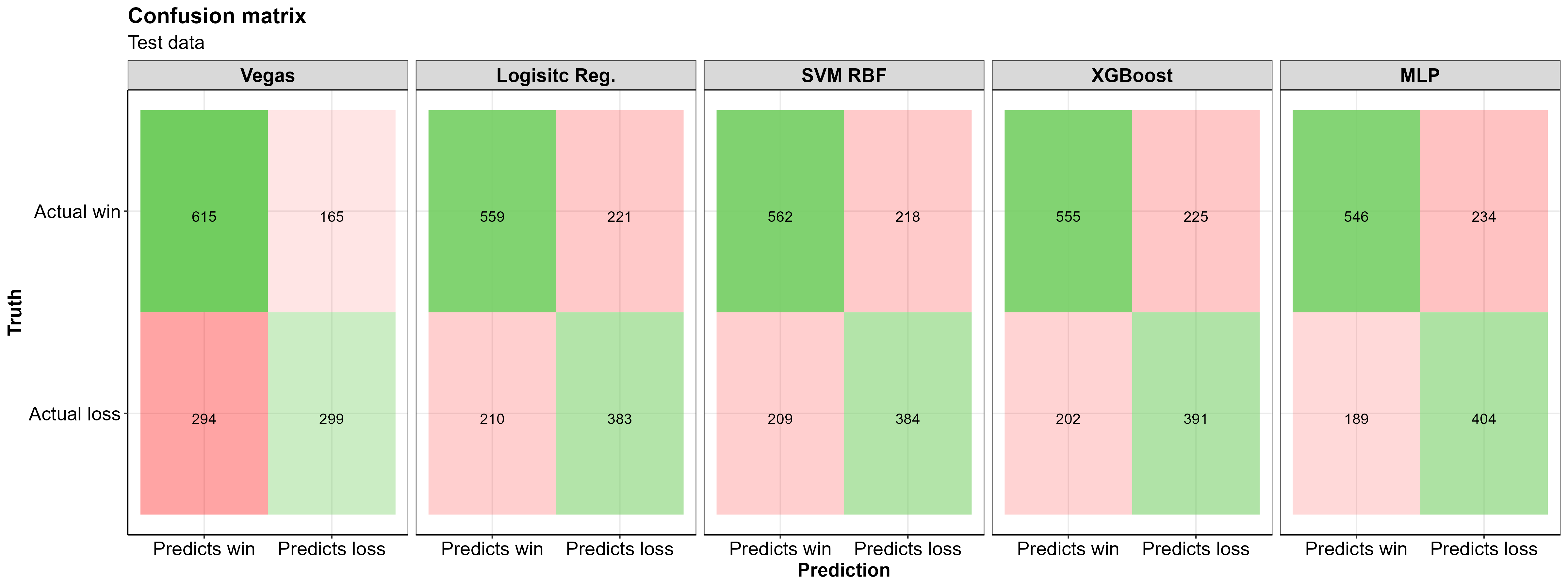
Looks like all of the models are performing similarly on the test data, except for the K-nearest neighbors model, which is much worse at classifying the data then the rest of the models. This ROC curve just visually confirms what we saw in the model metric summary table. Based on these metrics, I am going to drop the K nearest neighbors model for now and focus on the rest of the models. I will also just use the better performing of the two SVM models (SVM RBF, Radial basis function Support Vector Machines)

## Confusion Matrices

A confusion matrix is a helpful way of understanding where a classification model is correct or incorrect in it’s predictions. At first glance, I thought the name confusion matrix came from the fact that *I was confused*. Turns out the name has more meaning, its purpose is to highlight *where a classification model is confused in its predictions*…

So, if this is your first time seeing a confusion matrix this illustration might help.

**Insert hot dog confusion matrix image**



**A few notes from looking at this confusion matrix:** - All three of my models do a better job of correctly predicting losses in the test data than Las Vegas did in the same games

* The Support Vector Machine has more total correct predictions than both other models
* The SVM RBF Model is the highest sensitivity model, it correctly predict wins better than the all other models as well as Las Vegas’s predictions
* The Logistic Reg. Model predicts losses better than the SVM and XGBoost models
* ~35% of correct predictions are losses, the remaining ~65% are correctly predicted wins (~43% of the test data were actual losses so this isn’t ***too*** far off)
* For this data/domain, having the model predict more wins than losses makes logical sense, **the home team tends to win more often than they lose**
* In general, the models have the most trouble classifying FP

And here are the summary statistics for these confusion matrices

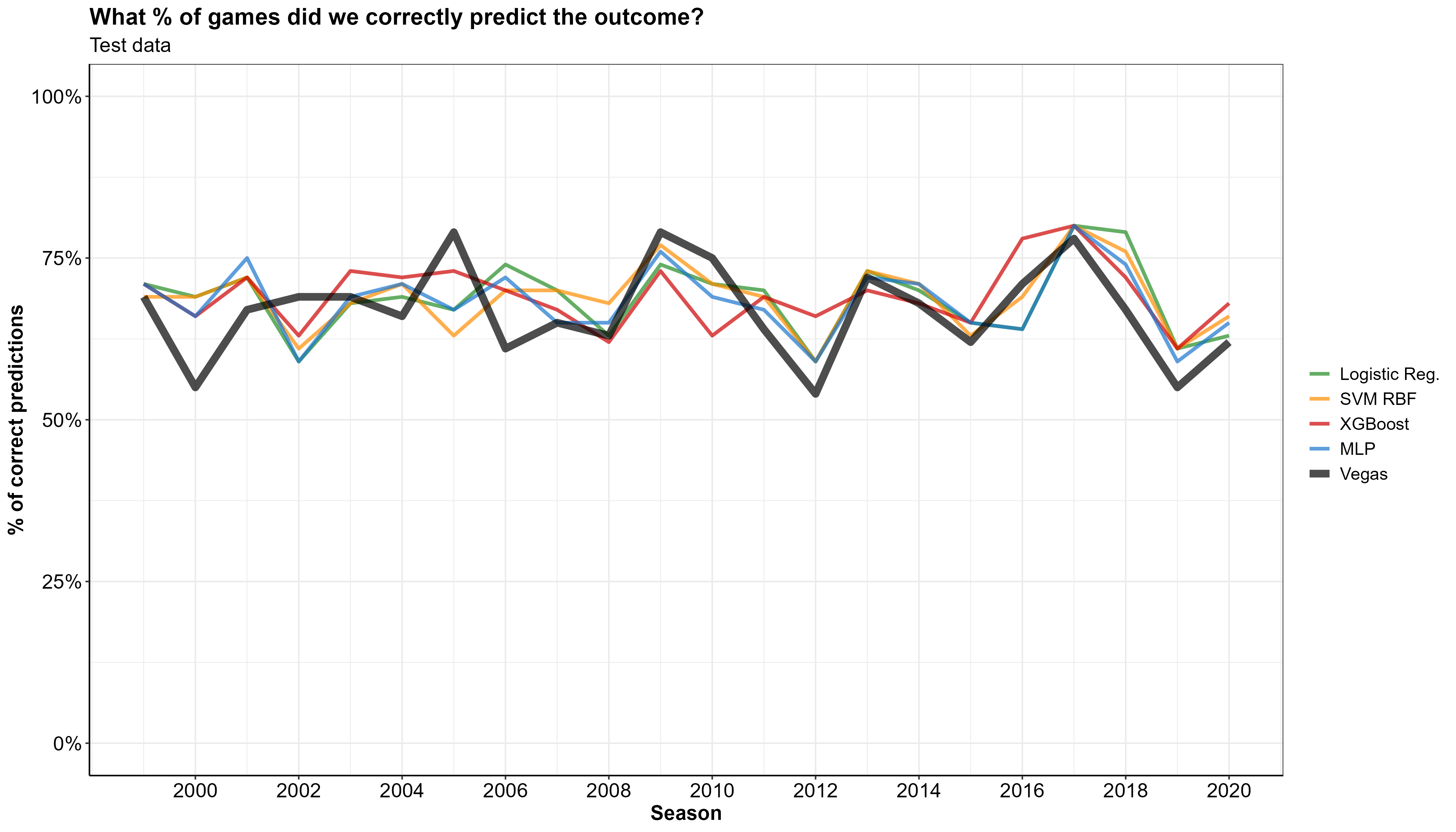
confmat\_tbl %>%  
 flextable::flextable() %>%   
 flextable::add\_header\_row(  
 values = c("", "Metrics"),  
 colwidths = c(1, 5)  
 ) %>%  
 flextable::theme\_box() %>%   
 flextable::align(align = "center", part = "all")

|  | **Metrics** | | | | |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Accuracy** | **Misclassification** | **Precision** | **Sensitivity** | **Specificity** |
| Vegas | 0.67 | 0.33 | 0.79 | 0.68 | 0.64 |
| Logisitc Reg. | 0.69 | 0.31 | 0.72 | 0.73 | 0.63 |
| SVM RBF | 0.69 | 0.31 | 0.72 | 0.73 | 0.64 |
| XGBoost | 0.69 | 0.31 | 0.71 | 0.73 | 0.63 |
| MLP | 0.69 | 0.31 | 0.70 | 0.74 | 0.63 |

Looking at the SVM model in the table above, we see a **sensitivity** of ~0.72, which means the SVM model ***correctly predicted the home team to win 72% of the time.*** The **specificity** of 0.73 indicates that ***73% of home team losses were correctly predicted***. Among all the models, the SVM model looks to do the best at correctly identifying wins and losses for the home team.

## Prediction from the test data

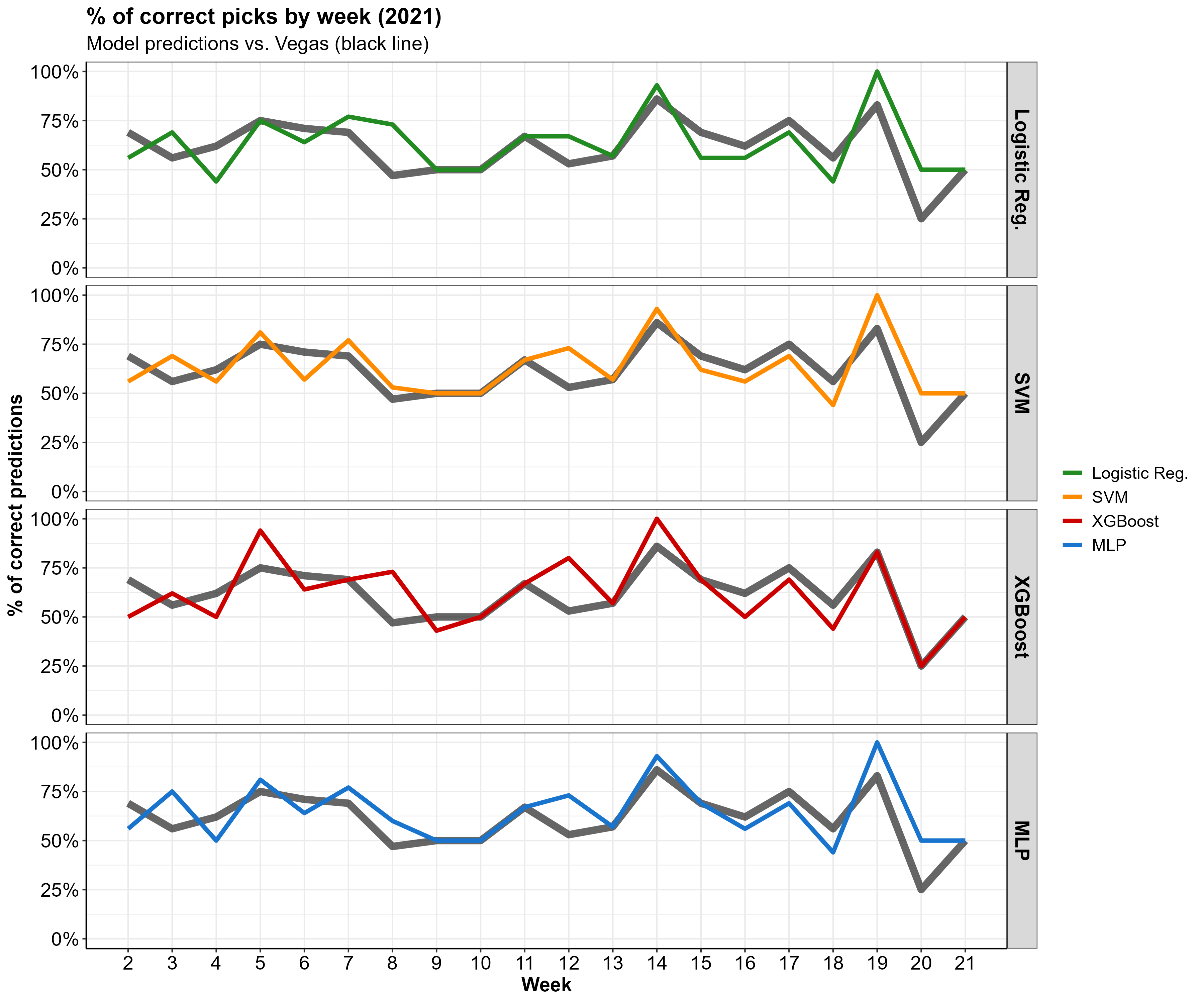
If we look at the 25% of the data we split off as a testing dataset, we are looking at 1373 games. We can generate predictions for those games, and calculate the percent of games each year that our models predicted the correct outcome for.



## Holdout dataset (2021 Predictions)

Now let’s apply the fitted models to the 2021 season holdout data and see how those predictions fare against Vegas’ predicted winners.

We can visualize percent of games that Las Vegas got correct in 2021 and compare it to the ML models % of correct predictions. The X axis has the percent of correct predictions and the Y axis is the week of the NFL season.



Looks like our predictions are right in line with Vegas, in some weeks we do better than Vegas are predicting the correct game outcomes and some weeks we do worse.

The table below shows the actual outcomes of the 2021 season, the favored team according to Las Vegas, and the prediction’s from my fitted models. Green highlighted cells indicate that the prediction lined up with the actual game outcome, and red highlighted cells mean that the prediction was incorrect. The table is from the home team’s persepctive (i.e. a win in the actual\_outcome column means the home team won).

First we’ll can check out one of the weeks that our models outperformed Vegas.

## Good week of predictions

All of our models did better than Vegas in week 12 (i.e. we correctly predicted the outcome more that Vegas did)

# A week with more correct preditions than Las Vegas  
good\_week <-   
 pred\_table %>%   
 dplyr::filter(week == 12)   
 # dplyr::select(-mlp\_pred)  
  
# Color code TP and TN as green and FP and FN as red  
good\_week\_color <- ifelse(good\_week[, c(5, 6, 7, 8, 9)] == good\_week[, c(4, 4, 4, 4, 4)], "#85E088", "indianred2")  
# good\_week\_color <- ifelse(good\_week[, c(5, 6, 7, 8)] == good\_week[, c(4, 4, 4, 4)], "#85E088", "indianred2")  
# Flextable with color coding  
  
good\_week %>%   
 flextable::flextable() %>%   
 flextable::bg(j = c(5:9), bg = good\_week\_color) %>%  
 # flextable::bg(j = c(5:8), bg = good\_week\_color) %>%  
 flextable::add\_header\_row(  
 values = c(" ", "Vegas", "Models"), # values = c(" ","Vegas Predictions", "Model Predictions"),  
 colwidths = c(4, 1, 4)  
 # colwidths = c(4, 1, 3)  
 # values = c(" ", "Vegas / Model Predictions"), colwidths = c(4, 4)  
 ) %>%   
 flextable::theme\_box() %>%   
 flextable::align(align = "center", part = "all") %>%   
 flextable::fit\_to\_width(max\_width = 6)

|  | | | | **Vegas** | **Models** | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **week** | **home\_team** | **away\_team** | **actual\_outcome** | **vegas\_pred** | **log\_reg\_pred** | **svm\_pred** | **xgb\_pred** | **mlp\_pred** |
| 12 | BAL | CLE | win | win | loss | loss | win | loss |
| 12 | CIN | PIT | win | win | win | win | win | win |
| 12 | DAL | LV | loss | win | win | win | win | win |
| 12 | DEN | LAC | win | loss | win | win | loss | win |
| 12 | DET | CHI | loss | loss | loss | loss | loss | loss |
| 12 | GB | LA | win | loss | loss | loss | win | loss |
| 12 | HOU | NYJ | loss | win | win | loss | loss | loss |
| 12 | IND | TB | loss | loss | loss | loss | loss | loss |
| 12 | JAX | ATL | loss | loss | loss | loss | loss | loss |
| 12 | MIA | CAR | win | loss | win | win | win | win |
| 12 | NE | TEN | win | win | win | win | win | win |
| 12 | NO | BUF | loss | loss | loss | loss | loss | loss |
| 12 | NYG | PHI | win | loss | loss | loss | loss | loss |
| 12 | SF | MIN | win | win | win | win | win | win |
| 12 | WAS | SEA | win | loss | win | win | win | win |

## Bad week of predictions

And now a bad week…

# A week with fewer correct preditions than Las Vegas  
bad\_week <-   
 pred\_table %>%   
 dplyr::filter(week == 15)  
  
# Color code TP and TN as green and FP and FN as red  
bad\_week\_color <- ifelse(bad\_week[, c(5, 6, 7, 8, 9)] == bad\_week[, c(4, 4, 4, 4, 4)], "#85E088", "indianred2")  
  
# Flextable with color coding  
bad\_week %>%   
 flextable::flextable() %>%   
 flextable::bg(j = c(5:9), bg = bad\_week\_color) %>%   
 flextable::add\_header\_row(  
 values = c(" ", "Vegas", "Models"),   
 colwidths = c(4, 1, 4)  
 ) %>%   
 flextable::theme\_box() %>%   
 flextable::align(align = "center", part = "all") %>%   
 flextable::fit\_to\_width(max\_width = 6)

|  | | | | **Vegas** | **Models** | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **week** | **home\_team** | **away\_team** | **actual\_outcome** | **vegas\_pred** | **log\_reg\_pred** | **svm\_pred** | **xgb\_pred** | **mlp\_pred** |
| 15 | BAL | GB | loss | loss | loss | loss | loss | loss |
| 15 | BUF | CAR | win | win | win | win | win | win |
| 15 | CHI | MIN | loss | loss | loss | loss | loss | loss |
| 15 | CLE | LV | loss | loss | win | loss | win | win |
| 15 | DEN | CIN | loss | win | win | win | win | win |
| 15 | DET | ARI | win | loss | loss | loss | loss | loss |
| 15 | IND | NE | win | loss | win | win | win | win |
| 15 | JAX | HOU | loss | win | win | win | loss | loss |
| 15 | LA | SEA | win | win | loss | loss | win | win |
| 15 | LAC | KC | loss | loss | loss | loss | loss | loss |
| 15 | MIA | NYJ | win | win | win | win | win | win |
| 15 | NYG | DAL | loss | loss | loss | loss | loss | loss |
| 15 | PHI | WAS | win | win | win | win | win | win |
| 15 | PIT | TEN | win | win | loss | loss | loss | loss |
| 15 | SF | ATL | win | win | win | win | win | win |
| 15 | TB | NO | loss | win | win | win | win | win |

# **Conclusion**

We did it, we implemented NFL data into a Machine learning workflow and generated some reasonably accurate predictions!

To recap what I did: - Started with raw NFL play-by-play data - Cleaned and processed the data into a model-ready format - Selected a handful of ML models, - trained and tested the models with our prepped data, - Split our data into training and testing datasets - Trained our models using the training data and evaulated them using 10 fold cross validation. - Determined optimal hyperparameters and used these to refit our models on the entire dataset - Generated predictions and compared our predictions to the predicted favorites according to Las Vegas

I’m not a huge sports bettor, but I really just wanted to make a model that could potentially make me money if I was.