Linear Model Approaches to NBA Game Outcomes: Optimizing Predictive Accuracy and Extracting Actionable Insights through Inference

Tim Tantivilaisin
Department of Statistics
University of California, Berkeley
Berkeley, CA 94720
timt@berkeleyedu

Jeffrey Kuo
Department of Statistics
University of California, Berkeley
Berkeley, CA 94720
jrkuo2015@berkeley.edu

Abstract

This paper explores the utilization of linear models to predict National Basketball Association (NBA) game outcomes and extract actionable insights for enhancing team performance. In our work, we present a description of our logistic regression model with LASSO penalty, which achieved a prediction accuracy of 71.22%. We further discuss the conclusions drawn from the model, considering its strengths and potential limitations. The paper culminates in a conclusion summarizing how different NBA teams could use our model to produce actionable insights to better improve their game outcomes.

1 Introduction

Our paper aims to predict the win/loss outcomes of the most recent NBA regular season (2022-2023), leveraging data on historical team performance, home field advantage, and more. For this purpose, we prefer to use linear models over more complex machine learning algorithms due to their greater interpretability. The data we used starts from the 2012 season for every team and fuels our chosen linear models. This type of analysis holds critical insights for NBA teams, impacting decisions such as player acquisition and coaching strategies. These decisions can significantly influence team revenues, with potential variations reaching hundreds of millions of dollars annually.

Our data comes from the nbastatR package that includes data for every NBA game played in a particular season with corresponding player performance (such as total points, minutes played, for each row), home field advantage, and many other columns, with the of just the 2022 season's number of observations being 26039 rows. For our analysis, utilized the last 10 seasons worth of data.

Our logistic regression model pursues dual objectives: maximizing prediction accuracy and extracting actionable insights through inference for NBA teams. To fulfill these goals, we present two versions of our model - one optimized for prediction accuracy using a logistic model with a LASSO penalty, and the other standard logistic regression model tailored for inference. These models demonstrate remarkable prediction accuracies of 71.22% and 68.94% respectively.

We conclude the insights our models can bring to NBA team, and what actions can be done as a result. We also discuss the limitations of our models, and what can be improved with future iterations.

2 Model

We are interested in modeling this probability:

$$\mathbb{P}(\text{Home Team wins vs. Away Team}|\text{data}) = \hat{p} \text{ where } 0 \leq \hat{p} \leq 1. \tag{1}$$

Preprint. Under review.

Where our predicted result of the game will be a binary function of:

$$\hat{y} = \begin{cases} \text{Home Team wins} & \hat{p}_i \geq 0.5 \\ \text{Away Team wins} & \hat{p}_i < 0.5. \end{cases} \tag{2}$$

To gain intuition on what type of data we would use to calculate this, suppose we are given the scenario of predicting the winner of Home vs Away. Naively, we would like to quantify how "good" the Home Team is and compare that to the Away Team recently. So the higher our quality measure of the Home Team is compared to that of the Away Team, the greater the value of $\hat{p}$. We can now observe that each row of our design matrix, will be a comparison (arithmetic difference) of how much better the Home Team was compared to the Away Team as our independent variable, and the win/loss result of the game as our dependent variable. Each column in the design matrix will represent a different quality measure.

Our initial aim was to devise a single model that excelled in both prediction accuracy and inferential capability. However, during the course of our research, we discovered a distinct trade-off between these two objectives. We found that a model which yielded excellent prediction accuracy inherently suffered from multi-collinearity among our variables, significantly impairing its inferential abilities. The complexities of this issue, including the underlying reasons and implications, will be further elaborated in Section 3.

In light of this observation, we opted for a two-model approach. For maximizing prediction accuracy, we implemented a logistic regression model with a LASSO penalty. On the other hand, to extract meaningful inferences, we utilized a regular logistic regression model. Consequently, this dual-model approach enabled us to fulfill our original objectives, albeit through separate, specialized models.

We use data starting from the 2012 season up through the day of prediction. For example, if we're predicting the winners of the teams playing on 12/12/2022, our training data would be all the games from 2012 - 12/11/2022. Thus we fit a new model for every game day in the 2022 season and use that model to predict on the outcomes on that game day. We construct our binary outcome variable to be 1 if the home team won and 0 if the away team won. To capture the relative advantage one team has over another, we construct our covariates to be the home team's average stats over their last 10 games minus the away team's average stats over their last 10 games. The stats we used in our logistic model with the LASSO penalty are: total rebounds, steals, blocks, turnovers, offensive rating, defensive rating, true shooting, and win rate. We adjust all stats except for true shooting and win rate to be on a per 100 possessions basis to control for pace of a game. We chose to average these stats over the last 10 games in order to capture each team's recent performance which should be representative of how they play in their next game. In addition, we include a covariate to indicate the relative difference between each team in whether the current game being played is a back to back (consecutive game). This covariate takes a value of 1 if the current game is a back to back for only the home team (meaning they played just the day before) but not the away team, a value of 0 if the current game is a back to back for both teams or for neither team, and a value of -1 if the current game is a back to back for only the away team. In our final logistic regression model that we use to extract insights, we removed the win rate covariate, keeping everything else the same. This will be further elaborated upon in Section 3.

3 Results and Limitations

3.1 Predictive Model Results

Our implementation of the logistic regression with a LASSO penalty selected two predictors: the coefficient associated with "Back to Back Game" games and the average win rate over the last 10 games. This resulted in an accuracy of 71.22% over the 2022-2023 season, demonstrating a precision of 0.7428 and recall of 0.7689. Exact numbers are displayed in Figure 1.

The dominance of these two variables can be attributed to the high degree of collinearity present among the predictors. The majority of the variables in our data set, such as avg assists, avg rebounds, etc., were found to be highly correlated with the 'average win percentage over the last 10 games.' As such, the inclusion of these additional variables did not contribute significantly to the predictive power of the model, as their effects were largely accounted for by the "average win percentage."

We also observed a non-intuitive trend when analyzing prediction accuracy over the course of the season. One might expect the prediction accuracy to be lower at the beginning of the season due to a 'start-up' effect, where new player acquisitions and team changes take time to in our model, since we are averaging over the last 10 games of the previous season. However, this expectation was not supported by our analysis. According to Figure 2 in the appendix, where the x-axis represents game days and the y-axis represents cumulative error count over the season, there was no noticeable difference in slope in prediction error count at the season's start. This could be attributed to the fact that only a few teams made significant trades during this particular year, resulting in minimal disruption to the continuity of team performance statistics from the previous season overall.

3.2 Inferential Model Results

As previously discussed, we remove the predictor of average win rate over the last 10 games because we do not believe this variable is helpful to coaches and management making player and game based decisions as it showed that teams are more likely to win their next game if they won more games than their opponent did. Doing so yielded us with the models shown in Table 1 and Table 2 in the Appendix. The model in Table 1 is the model used to predict the games of the second day of the 2022 regular season while the model in Table 2 is the model obtained after the last day of the 2022 regular season. We can see that our coefficients stayed mostly the same between the two models. This makes sense intuitively as the same data from 2012 up to the start of the 2022 season was used to train both models while the model in Table 2 also includes all the games in the 2022 season, which is a small proportion of the overall data set.

Since we designed our logistic regression model to output a higher probability if the home team won, we should expect the stats that a team wants to be higher, which signal that the team is playing well in regard to those aspects, to have positive coefficients and vice versa for stats that a team wants to be lower. This still holds true in our case where we are taking the difference between each home and away team stat, as we are subtracting away from home.

Let's examine the coefficients in Table 2 and see how one should interpret them. The difference in average steals has a coefficient of 0.061. Exponentiating by this value yields a value of 1.063 which means that holding all other variables in the model constant, the odds that the home team wins increases by a factor of 1.063 for each additional 1 steal per 100 possessions that they average over the last 10 games more than the away team. Intuitively, this makes sense as a steal means taking the ball away from your opponent and thus their opportunity to score. Similar logic can be applied to the average turnovers coefficient of -0.049 which translates to the home team's odds of winning decreasing by a factor of 0.952 for each additional 1 turnover per 100 possessions they average over their last 10 games holding all other variables constant. The covariate "Back to Back Game" has a relatively large magnitude coefficient of -0.254 and also has a different interpretation as it can only take on values of 1, 0, and -1 as previously mentioned. This means that going from either both or neither teams playing back-to-back games to just the home team playing back-to-back games would decrease the odds that the home team wins by a factor of 0.776. This makes sense as back-to-back games are exhausting because the players are getting only a day of rest as opposed to 2 or 3 days. Lastly, the intercept ("Coefficient" in the Tables) provides insight to home court advantage. An intercept of 0.330 translates to a baseline probability of 58.3% of the home team winning without considering any of the predictors in the model.

3.3 Limitations

Our models, while yielding promising results, is not without its limitations, i.e. the general nature of the coefficients derived from our models. These coefficients are indicative of league-wide trends and do not cater specifically to individual teams. While this broad-strokes approach aids in understanding overall game outcomes, it may not fully capture the unique strategies and dynamics at play within specific teams. Further, our models focus on team-level analysis and do not delve into the contribution of individual players. This approach, while simplifying the overall modeling process, might mask the impact of individual player performance on their respective teams.

Another aspect is, while the "start-up" effect may not be present overall, it may affect some teams much more than others. Furthermore, our model is not robust to sudden changes in player roster

during the season. This can be a problem where a sudden, season-ending injury to a key player might result in a drop in the model's predictive accuracy.

Finally, we must consider the evolutionary nature of game strategies. Our models, which utilize data starting from 2012, might not fully account for strategic shifts over this period. For instance, the current trend towards more frequent three-point shots significantly differs from the center-focused strategies of the 1990s and early 2000s.

4 Concluding Insights

We see our models being the most useful for NBA teams seeking to make strategic decisions in the key areas of player acquisition and coaching strategies.

When it comes to player acquisitions, teams can use our model to empirically assess the potential impact of prospective players on team performance. By considering a player's historical performance data, teams can observe how this might change the team's performance metrics, and their effect of their on future game outcomes. These insights can help teams make more informed decisions when scouting for talent or considering trades, maximizing the potential value they gain from new player acquisitions.

Additionally, the inferential capabilities of our model can inform coaching strategies. By examining the coefficients of our model, teams can identify key variables that significantly negatively contribute to their game outcomes. Armed with this information, coaches can tailor their training programs, game plans, or exercises to focus on these critical areas, enhancing their team's overall performance.

5 Appendix

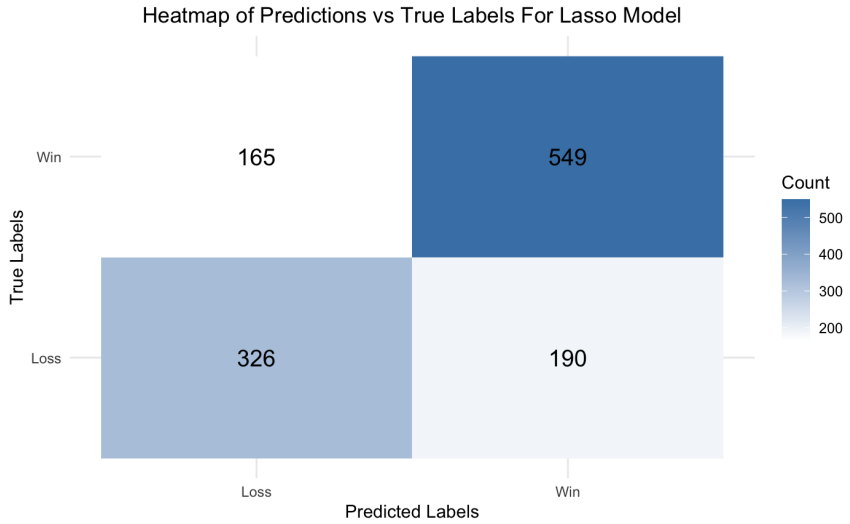


Figure 1: Confusion Matrix of Lasso Model

Table 1

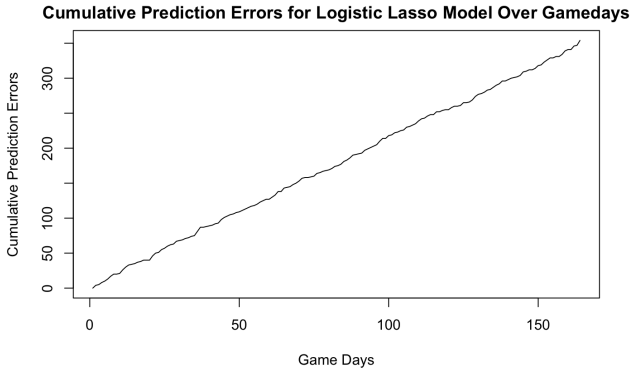
	Dependent variable:
	Home Team wins
Back to Back Game	-0.249^{***} (0.042)
Avg. Total Rebounds	0.039^{***} (0.011)
Avg. Steals	0.058^{***} (0.016)
Avg. Blocks	0.006 (0.015)
Avg Turnovers	-0.046^{***} (0.016)
Avg. Offensive Rating	0.081^{***} (0.012)
Avg. Defensive Rating	-0.110^{***} (0.006)
Avg. True Shooting	10.356^{***} (2.137)
Coefficient	0.325^{***} (0.022)
Observations	11,796
Log Likelihood	-6,402.078

Note: Logistic model after the first game of the 2022-2023 season. *p<0.1; **p<0.05; ***p<0.01

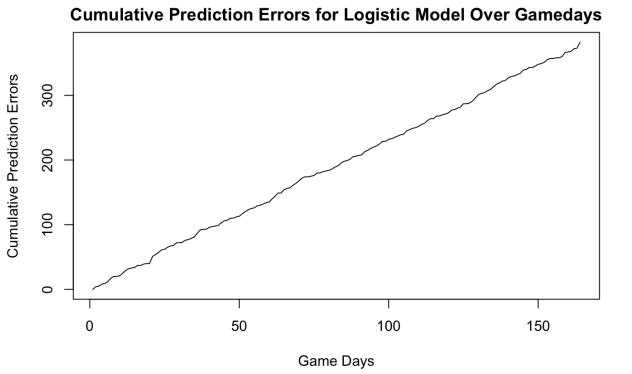
Table 2

	Dependent variable:
	Home Team wins
Back to Back Game	-0.254^{***} (0.040)
Avg. Total Rebounds	0.040^{***} (0.010)
Avg. Steals	0.061^{***} (0.016)
Avg. Blocks	0.004 (0.014)
Avg Turnovers	-0.049^{***} (0.015)
Avg. Offensive Rating	0.080^{***} (0.011)
Avg. Defensive Rating	-0.107^{***} (0.005)
Avg. True Shooting	10.539^{***} (2.007)
Coefficient	0.330^{***} (0.021)
Observations	13,011
Log Likelihood	-7,099.162

Note: Logistic model after the last game of the 2022-2023 season. *p<0.1; **p<0.05; ***p<0.01



(a) Cumulative errors for our model focused on inference.



(b) Cumulative errors for our model focusing on prediction accuracy.

Figure 2

6 References

[1] R: *nbastatR*, Alex Bresler. R package.

[2] *NBA Oracle* (2009), Matthew Beckler, Hongfei Wang, Michael Papamichael. Carnegie Mellon University.

[3] *NBA-Predict* (2019), Jake Kandell, *GitHub repository*, https://github.com/JakeKandell/NBA-Predict

Stat 230A Final RMD

Tim Tantivilaisin

2023-04-10

RUN THIS CODE BLOCK TO CREATE THE DATASET

```
# Install and Load the nbastatR package
library('nbastatR')
library('dplyr')
library('ggplot2')
Sys.setenv("VROOM_CONNECTION_SIZE" = 131072 * 2)

teams <- all_nba_teams(return_message = TRUE)

# what seasons you want to analyze
seasons <- seq(2013, 2023)

# Access the game tables for a particular season
gamedata <- game_logs(seasons = seasons)

# UNCOMMNET OUT THIS BLOCK TO SAVE TO CSV
# wd <- paste0(getwd(), "/data")
# write.csv(x = gamedata, file = paste0(wd,"/2012_2023_Data.csv"), row.names = FALSE)

############## This was for the proposal ##############

# let's get box scores with manipulating data
# group by game_id, then group by team, pts by team
box_scores <- gamedata %>%
  select(idGame, nameTeam, pts) %>%
  group_by(idGame, nameTeam) %>%
  summarize(sum_pts = sum(pts))

dim(gamedata)
head(gamedata)

dim(box_scores)
head(box_scores)

ggplot(box_scores, aes(x = box_scores$sum_pts, fill = box_scores$sum_pts)) +
  geom_histogram(color = "black", fill = "white", binwidth = 1) +
  xlab("Points Scored by Team per Game") +
  ylab("Frequency") +
  ggtitle("Histogram of Points Scored by Team per Game 2022 Season")
```

```r
library('dplyr')
library('ggplot2')

# load the data we got from the api in the previous code chunk
gamelogs <- read.csv("./data/2012_2023_Data.csv")

# let's select for columns to make the data more manageable
team_summary <- gamelogs %>%
  select(slugSeason, dateGame, idGame, nameTeam, isB2BSecond,
         locationGame, slugMatchup, outcomeGame, blk, tov, pf, pts, stl, treb,
         oreb, dreb, fga, fgm, ftm, fta) %>%
  arrange(idGame, nameTeam, locationGame)

# prepping data before we aggregate on a by game basis.
# need to calculate team stats first.
team_summary <- team_summary %>%
  group_by(idGame, nameTeam) %>%
  summarize(slugSeason = first(slugSeason), dateGame = first(dateGame),
            idGame = first(idGame), nameTeam = first(nameTeam),
            isB2BSecond = first(isB2BSecond), locationGame = first(locationGame),
            slugMatchup = first(slugMatchup),outcomeGame = first(outcomeGame),
            sum_blk = sum(blk), sum_tov = sum(tov), sum_pf = sum(pf), sum_stl = sum(stl),
            sum_pts = sum(pts), sum_treb = sum(treb),
            sum_oreb = sum(oreb), sum_dreb = sum(dreb) ,sum_fga = sum(fga),
            sum_fgm = sum(fgm), sum_ftm = sum(ftm), sum_fta = sum(fta)) %>%
  arrange(idGame, desc(locationGame))

# verified with https://www.statmuse.com/nba/ask?q=trailblazers+04%2F9%2F23+true+shooting+percentage
team_summary <- team_summary %>%
  mutate(trueshooting = sum_pts/(2*(sum_fga + 0.44*sum_fta)))

# now need to calculate number of possessions. This is an estimate.
# https://www.basketball-reference.com/about/glossary.html#team
# this is called poss in the previous
team_summary <- team_summary %>%
  mutate(poss = ifelse(locationGame == "H", 0.5*((sum_fga + 0.4*sum_fta - 1.07*(sum_oreb/(sum_oreb + lea

team_summary <- team_summary %>%
  mutate(poss = ifelse(locationGame == "A", lag(poss),poss))

# calculate team offensive rating
team_summary <- team_summary %>%
  mutate(orate = (sum_pts/poss)*100)

# calculate team defensive rating
team_summary <- team_summary %>%
  mutate(drate = ifelse(locationGame == "H", (lead(sum_pts)/poss)*100,
                        (lag(sum_pts)/poss)*100))

# let's make 0 1 for b2b
team_summary <- team_summary %>%
  mutate(isB2BSecond = ifelse(isB2BSecond == "FALSE", 0, 1))
```

```
# let's make 0 1 for W/L
team_summary <- team_summary %>%
  mutate(outcomeGame = ifelse(outcomeGame == "W", 1, 0))

# now scale blk, tov, stl, pts, treb per 100 possessions
varsToScale <- c("sum_blk", "sum_tov", "sum_stl", "sum_pts", "sum_treb")

team_summary <- team_summary %>%
  mutate(sum_blk = (sum_blk/poss)*100) %>%
  mutate(sum_tov = (sum_tov/poss)*100) %>%
  mutate(sum_stl = (sum_stl/poss)*100) %>%
  mutate(sum_pts = (sum_pts/poss)*100) %>%
  mutate(sum_treb = (sum_treb/poss)*100)

# now calculate + -, which should also be by 100 possessions
team_summary <- team_summary %>%
  mutate(plu_min = ifelse(locationGame == "H", sum_pts-lead(sum_pts),
         sum_pts - lag(sum_pts)))
```

At this point, we have all the data to start wrangling our design matrix.

```
design_matrix <- team_summary

design_matrix <- design_matrix %>%
  select(-slugMatchup, -sum_pf, -sum_oreb, -sum_dreb, -sum_fga, -sum_fgm, -sum_ftm,
         -sum_fta)

print(design_matrix)
```

```
## # A tibble: 26,416 x 17
## # Groups:   idGame [13,208]
##      idGame nameTeam    slugSeason dateGame isB2BSecond locationGame outcomeGame
##       <int> <chr>       <chr>      <chr>          <dbl> <chr>              <dbl>
##  1 21200001 Cleveland ~ 2012-13    2012-10~           0 H                      1
##  2 21200001 Washington~ 2012-13    2012-10~           0 A                      0
##  3 21200002 Miami Heat  2012-13    2012-10~           0 H                      1
##  4 21200002 Boston Cel~ 2012-13    2012-10~           0 A                      0
##  5 21200003 Los Angele~ 2012-13    2012-10~           0 H                      0
##  6 21200003 Dallas Mav~ 2012-13    2012-10~           0 A                      1
##  7 21200004 Toronto Ra~ 2012-13    2012-10~           0 H                      0
##  8 21200004 Indiana Pa~ 2012-13    2012-10~           0 A                      1
##  9 21200005 Philadelph~ 2012-13    2012-10~           0 H                      1
## 10 21200005 Denver Nug~ 2012-13    2012-10~           0 A                      0
## # i 26,406 more rows
## # i 10 more variables: sum_blk <dbl>, sum_tov <dbl>, sum_stl <dbl>,
## #   sum_pts <dbl>, sum_treb <dbl>, trueshooting <dbl>, poss <dbl>, orate <dbl>,
## #   drate <dbl>, plu_min <dbl>
```

Before we calculate the rolling average, let's see at which point every NBA team has played n games in the 2012 regular season.

```r
n <- 10
nba_teams <- design_matrix$nameTeam %>%
  unique() %>% sort()

# get the first 331 games of the season
test_matrix <- design_matrix[1:331, ]
game_count <- test_matrix %>% count(nameTeam) %>%
  group_by(nameTeam) %>%
  summarise(sum(n))

print(paste("The minimum number of games played by any team:",
            min(game_count[,2])))
```

```
## [1] "The minimum number of games played by any team: 10"
```

```r
# install.packages("slider")
library(slider)
# the strategy to get the last n games is to basically just do a lag(n = 10)
design_matrix <- design_matrix %>%
  group_by(nameTeam) %>%
  arrange(dateGame) %>%
  mutate(avg_pts = slide_dbl(sum_pts, mean, .before = 9, .complete = TRUE)) %>%
  mutate(avg_treb = slide_dbl(sum_treb, mean, .before = 9, .complete = TRUE)) %>%
  mutate(avg_stl = slide_dbl(sum_stl, mean, .before = 9, .complete = TRUE)) %>%
  mutate(avg_blk = slide_dbl(sum_blk, mean, .before = 9, .complete = TRUE)) %>%
  mutate(avg_tov = slide_dbl(sum_tov, mean, .before = 9, .complete = TRUE)) %>%
  mutate(avg_orate = slide_dbl(orate, mean, .before = 9, .complete = TRUE)) %>%
  mutate(avg_drate = slide_dbl(drate, mean, .before = 9, .complete = TRUE)) %>%
  mutate(avg_pl_min = slide_dbl(plu_min, mean, .before = 9, .complete = TRUE)) %>%
  mutate(avg_true_s = slide_dbl(trueshooting, mean, .before = 9, .complete = TRUE)) %>%
  mutate(avg_win_perc = slide_dbl(outcomeGame, sum, .before = 9, .complete = TRUE)) %>%
  mutate(avg_win_perc = avg_win_perc/10) %>%
  ungroup()

# get rid of N/A rows
design_matrix <- design_matrix[311:nrow(design_matrix),]

# then now select only the columns that we take the difference of
design_matrix <- design_matrix %>%
  select(-avg_pts,-sum_blk, -sum_tov, -sum_stl, -sum_pts, -sum_treb,
         -trueshooting, -poss, -orate, -drate, -plu_min)

# let's finally take the difference of the columns
diff_vec <- c("avg_treb", "isB2BSecond", "avg_stl", "avg_blk", "avg_tov", "avg_orate",
              "avg_drate", "avg_pl_min", "avg_true_s", "avg_win_perc")

design_matrix <- design_matrix %>%
  mutate(across(diff_vec, ~(ifelse(locationGame == "H", . - lead(.), lead(.)))))
```

```
## Warning: There was 1 warning in `mutate()`.
## i In argument: `across(diff_vec, ~(ifelse(locationGame == "H", . - lead(.),
##   lead(.))))`.
## Caused by warning:
```

```
## ! Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use ‘all_of()‘ or ‘any_of()‘ instead.
##   # Was:
##   data %>% select(diff_vec)
##
##   # Now:
##   data %>% select(all_of(diff_vec))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
```

```
# now that we've taken the difference, we only need the H rows
design_matrix <- design_matrix %>%
  filter(locationGame == "H")

# Find rows with any missing values (N/A)
rows_with_na <- apply(is.na(design_matrix), 1, any)

# Display the rows with N/A values
# design_matrix[rows_with_na, ]


##### think through this logic later, if the model turns out being bad.
# the cause of this are NBA teams changing names.
# let's remove N/A columns.
design_matrix <- design_matrix %>% na.omit(na.omit)

####################

# only selecting for columns we now need
design_matrix <- design_matrix %>%
  select(-idGame, -nameTeam, -locationGame)

# let's make outcome game a factor
design_matrix$outcomeGame <- as.factor(design_matrix$outcomeGame)
```

We can now finally split into train and test split. Where train is everything but the 2022-2023 season.

```
season <- "2022-23"

# Find the first instance of the value in the '2022-2023' column
test_start <- which(design_matrix$slugSeason == season)[1]

train_design_matrix <- design_matrix[1:test_start-1,]
test_design_matrix <- design_matrix[test_start:nrow(design_matrix),]

logit_model <- glm(outcomeGame ~ isB2BSecond + avg_treb + avg_stl + avg_blk + avg_tov +
                     avg_orate + avg_drate + avg_true_s + avg_win_perc + avg_pl_min,
                   data = train_design_matrix, family =
                     binomial(link="logit"))

summary(logit_model)
```

```
##
```

```
## Call:
## glm(formula = outcomeGame ~ isB2BSecond + avg_treb + avg_stl +
##     avg_blk + avg_tov + avg_orate + avg_drate + avg_true_s +
##     avg_win_perc + avg_pl_min, family = binomial(link = "logit"),
##     data = train_design_matrix)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6519  -0.8362   0.3535   0.8194   2.6039
##
## Coefficients: (1 not defined because of singularities)
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)   0.3408878  0.0231788  14.707  < 2e-16 ***
## isB2BSecond  -0.2787598  0.0436929  -6.380 1.77e-10 ***
## avg_treb      0.0153859  0.0116050   1.326    0.185
## avg_stl       0.0243830  0.0171048   1.426    0.154
## avg_blk      -0.0088431  0.0153534  -0.576    0.565
## avg_tov      -0.0001577  0.0167087  -0.009    0.992
## avg_orate    -0.0088280  0.0126264  -0.699    0.484
## avg_drate     0.0080190  0.0071072   1.128    0.259
## avg_true_s    2.9052751  2.2469788   1.293    0.196
## avg_win_perc  4.8273900  0.1694479  28.489  < 2e-16 ***
## avg_pl_min           NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 16085  on 11795  degrees of freedom
## Residual deviance: 11891  on 11786  degrees of freedom
## AIC: 11911
##
## Number of Fisher Scoring iterations: 4
```

```
x_test <- test_design_matrix %>%
  select(-slugSeason, -dateGame, -outcomeGame)

y_test <- test_design_matrix %>%
  select(outcomeGame)

# get predictions
predicted_probs <- predict(logit_model, newdata = x_test, type = "response")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
# Convert probabilities to binary outcomes using a threshold of 0.5
predicted_outcomes <- ifelse(predicted_probs > 0.5, 1, 0)

# getting accuracy
correct_predictions <- predicted_outcomes == y_test

# Calculate accuracy
```

```r
accuracy_log <- mean(correct_predictions)

print(accuracy_log)
```

```
## [1] 0.7097561
```

```r
# diagnosing collinearity with avg_pl_min
# design_matrix$avg_orate - design_matrix$avg_drate - design_matrix$avg_pl_min

# avg_pl_min is avg_orate - avg_drate so we drop that variable and re-run regression
design_matrix <- design_matrix %>% subset(select = -avg_pl_min)

test_start <- which(design_matrix$slugSeason == season)[1]

train_design_matrix <- design_matrix[1:test_start-1,]
test_design_matrix <- design_matrix[test_start:nrow(design_matrix),]

logit_model <- glm(outcomeGame ~ isB2BSecond + avg_treb + avg_stl + avg_blk + avg_tov +
                     avg_orate + avg_drate + avg_true_s,
                   data = train_design_matrix, family =
                     binomial(link="logit"))

summary(logit_model)
```

```
##
## Call:
## glm(formula = outcomeGame ~ isB2BSecond + avg_treb + avg_stl +
##     avg_blk + avg_tov + avg_orate + avg_drate + avg_true_s, family = binomial(link = "logit"),
##     data = train_design_matrix)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.9321  -0.9362   0.4389   0.8715   2.6159
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.324701   0.022131  14.671  < 2e-16 ***
## isB2BSecond -0.248951   0.041797  -5.956 2.58e-09 ***
## avg_treb     0.039044   0.011058   3.531 0.000414 ***
## avg_stl      0.057752   0.016353   3.532 0.000413 ***
## avg_blk      0.006292   0.014668   0.429 0.667951
## avg_tov     -0.045624   0.015903  -2.869 0.004120 **
## avg_orate    0.080766   0.011745   6.876 6.14e-12 ***
## avg_drate   -0.109725   0.005776 -18.997  < 2e-16 ***
## avg_true_s  10.355665   2.137270   4.845 1.26e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 16085  on 11795  degrees of freedom
## Residual deviance: 12804  on 11787  degrees of freedom
```

```
## AIC: 12822
##
## Number of Fisher Scoring iterations: 4
```

```
x_test <- test_design_matrix %>%
  select(-slugSeason, -dateGame, -outcomeGame)

y_test <- test_design_matrix %>%
  select(outcomeGame)

# get predictions
predicted_probs <- predict(logit_model, newdata = x_test, type = "response")

# Convert probabilities to binary outcomes using a threshold of 0.5
predicted_outcomes <- ifelse(predicted_probs > 0.5, 1, 0)

# getting accuracy
correct_predictions <- predicted_outcomes == y_test

# Calculate accuracy
accuracy_log <- mean(correct_predictions)

print(accuracy_log)
```

```
## [1] 0.6902439
```

```
# get all the game days for 2022-2023 season
design_matrix_2022 <- subset(design_matrix, slugSeason == season)
all_game_days <- unique(design_matrix_2022$dateGame)
num_errors_log_vec <- vector(mode = "numeric", length = length(all_game_days))

# function to return a confusion matrix for all predictions on date with model refit to each new game d
make_day_predictions <- function(data, date) {

  test_start <- which(data$dateGame == date)[1]
  test_end <- tail(which(data$dateGame == date), n=1)
  train_end <- test_start - 1

  train_design_matrix <- data[1:train_end,]
  test_design_matrix <- data[test_start:test_end,]

  logistic_model <- glm(outcomeGame ~ isB2BSecond + avg_treb + avg_stl + avg_blk + avg_tov +
                        avg_orate + avg_drate + avg_true_s + avg_win_perc,
                      data = train_design_matrix, family =
                        binomial(link="logit"))

  x_test <- test_design_matrix %>%
    select(-slugSeason, -dateGame, -outcomeGame)

  y_test <- test_design_matrix %>%
    select(outcomeGame)

  predicted_probs <- predict(logistic_model, newdata = x_test, type = "response")
```

```
  predicted_outcomes <- ifelse(predicted_probs > 0.5, 1, 0)

  y_test_vec <- y_test$outcomeGame

  # number errors
  num_errors_log <- sum(predicted_outcomes != y_test_vec)

  confusion_matrix <- table(factor(y_test_vec, levels = c(0, 1)),
                            factor(predicted_outcomes, levels = c(0, 1)),
                            dnn = c("Actual", "Predicted"))

  rownames(confusion_matrix) <- c("0", "1")
  colnames(confusion_matrix) <- c("0", "1")

  confusion_matrix_matrix <- as.matrix(confusion_matrix)

  return_list <- list(confusion_matrix_matrix, num_errors_log)

  return(return_list)
}

# initialize list of matrices
confusion_matrix_list <- vector("list", length = length(unique(design_matrix_2022$dateGame)))

# loop through all game days using our design matrix, creating list of confusion matrices for 2022-2023
for (i in 1:length(all_game_days)) {
  temp <- make_day_predictions(design_matrix, all_game_days[i])
  confusion_matrix_list[[i]] <- temp[[1]]
  num_errors_log_vec[i] <- temp[[2]]
}

confusion_mat_sum <- Reduce('+', confusion_matrix_list)

confusion_mat_sum
```

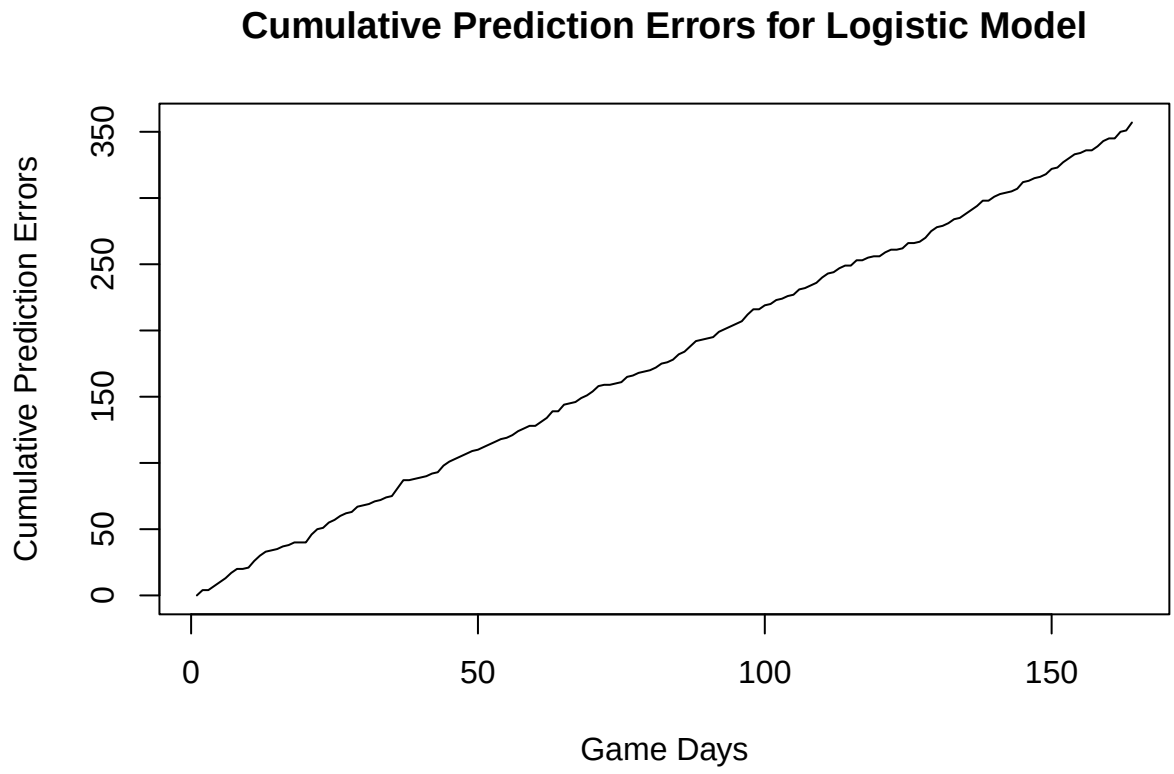
```
##       Predicted
## Actual   0   1
##      0 324 192
##      1 165 549
```

```
TN <- confusion_mat_sum[1,1]
FP <- confusion_mat_sum[1,2]
TP <- confusion_mat_sum[2,2]
FN <- confusion_mat_sum[2,1]

accuracy <- (TN + TP)/(TN + FP + TP + FN)
accuracy
```

```
## [1] 0.7097561
```

plotting cumulative errors in prediction logistic

```
game_days <- seq(length(all_game_days))
cumu_num_errors_log_vec <- cumsum(num_errors_log_vec)
# Plot the line graph
plot(game_days, cumu_num_errors_log_vec, type = "l", xlab = "Game Days", ylab = "Cumulative Prediction E
     main = "Cumulative Prediction Errors for Logistic Model", mar = c(5, 5, 4, 2))
```



Trying a ridge logistic

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-6
```

```
x_train <- train_design_matrix %>%
  select(-slugSeason, -dateGame, -outcomeGame)

y_train <- train_design_matrix %>%
  select(outcomeGame)

# Convert every column to numeric
```

```r
x_train <- as.matrix(data.frame(lapply(x_train, function(x) as.numeric(as.character(x)))))
y_train <- as.numeric(y_train$outcomeGame)

# Perform cross-validation to find the best lambda value for ridge
cv_fit_ridge <- cv.glmnet(x_train, y_train, family = "binomial", alpha = 0, nfolds = 10)

# Best lambda value
best_lambda_ridge <- cv_fit_ridge$lambda.min

# Fit ridge logistic regression model using the best lambda
ridge_logistic_regression <- glmnet(x_train, y_train, family = "binomial",
                                    alpha = 0, lambda = best_lambda_ridge)

# Coefficients
ridge_coefficients <- coef(ridge_logistic_regression)

# Predict probabilities
predicted_probabilities <- predict(ridge_logistic_regression, as.matrix(x_test), type = "response")

# Predict class labels
predicted_outcomes_ridge <- ifelse(predicted_probabilities > 0.5, 1, 0)

# getting accuracy
correct_pred_ridge <- predicted_outcomes_ridge == y_test

# Calculate accuracy
accuracy_log_ridge <- mean(correct_pred_ridge)

accuracy_log_ridge
```

```
## [1] 0.698374
```

```r
# function to return a confusion matrix for all predictions on date with model refit to each new game d
make_day_predictions_ridge <- function(data, date) {

  test_start <- which(data$dateGame == date)[1]
  test_end <- tail(which(data$dateGame == date), n=1)
  train_end <- test_start - 1

  train_design_matrix <- data[1:train_end,]
  test_design_matrix <- data[test_start:test_end,]

  x_train <- train_design_matrix %>%
  select(-slugSeason, -dateGame, -outcomeGame)

  y_train <- train_design_matrix %>%
    select(outcomeGame)

  # Convert every column to numeric
  x_train <- as.matrix(data.frame(lapply(x_train, function(x) as.numeric(as.character(x)))))
  y_train <- as.numeric(y_train$outcomeGame)

  # Perform cross-validation to find the best lambda value for ridge
```

```r
  cv_fit_ridge <- cv.glmnet(x_train, y_train, family = "binomial", alpha = 0, nfolds = 10)

  # Best lambda value
  best_lambda_ridge <- cv_fit_ridge$lambda.min

  # Fit ridge logistic regression model using the best lambda
  ridge_logistic_regression <- glmnet(x_train, y_train, family = "binomial",
                                      alpha = 0, lambda = best_lambda_ridge)

  x_test <- test_design_matrix %>%
    select(-slugSeason, -dateGame, -outcomeGame)

  y_test <- test_design_matrix %>%
    select(outcomeGame)

  predicted_probs <- predict(ridge_logistic_regression, newx = as.matrix(x_test), type = "response")
  predicted_outcomes <- ifelse(predicted_probs > 0.5, 1, 0)

  y_test_vec <- y_test$outcomeGame

  confusion_matrix <- table(factor(y_test_vec, levels = c(0, 1)),
                            factor(predicted_outcomes, levels = c(0, 1)),
                            dnn = c("Actual", "Predicted"))

  rownames(confusion_matrix) <- c("0", "1")
  colnames(confusion_matrix) <- c("0", "1")

  confusion_matrix_matrix <- as.matrix(confusion_matrix)

  return(confusion_matrix_matrix)
}

# initialize list of matrices
ridge_confusion_matrix_list <- vector("list", length = length(unique(design_matrix_2022$dateGame)))

# loop through all game days using our design matrix, creating list of confusion matrices for 2022-2023
for (i in 1:length(all_game_days)) {

  ridge_confusion_matrix_list[[i]] <- make_day_predictions_ridge(design_matrix, all_game_days[i])

}

ridge_confusion_mat_sum <- Reduce('+', ridge_confusion_matrix_list)

ridge_confusion_mat_sum
```

```
##       Predicted
## Actual   0   1
##      0 306 210
##      1 161 553
```

```r
TN_ridge <- ridge_confusion_mat_sum[1,1]
FP_ridge <- ridge_confusion_mat_sum[1,2]
TP_ridge <- ridge_confusion_mat_sum[2,2]
FN_ridge <- ridge_confusion_mat_sum[2,1]

ridge_accuracy <- (TN_ridge + TP_ridge)/(TN_ridge + FP_ridge + TP_ridge + FN_ridge)
ridge_accuracy
```

```
## [1] 0.698374
```

trying lasso logistic

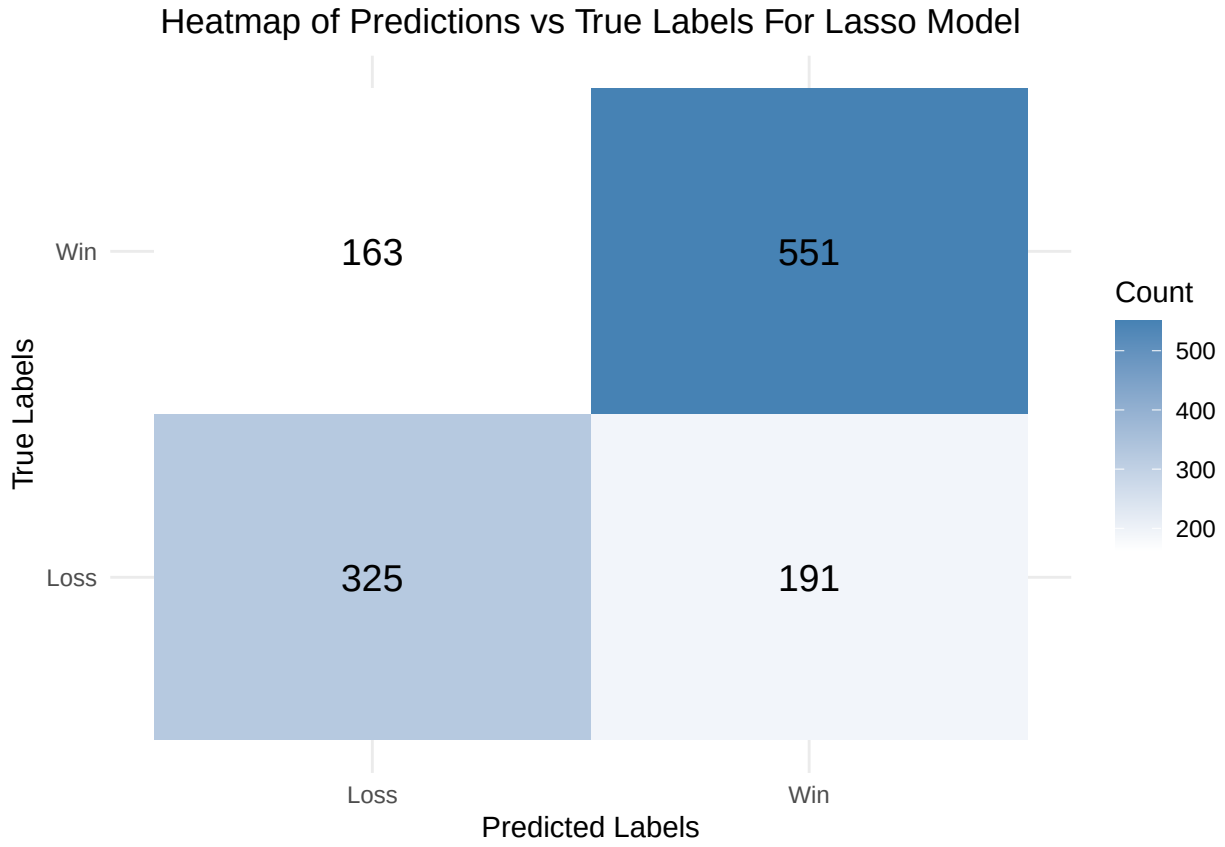
```r
# Perform cross-validation to find the best lambda value for lasso
cv_fit_lasso <- cv.glmnet(x_train, y_train, family = "binomial", alpha = 1, nfolds = 10)

# Best lambda value
best_lambda_lasso <- cv_fit_lasso$lambda.min

# Fit ridge logistic regression model using the best lambda
lasso_logistic_regression <- glmnet(x_train, y_train, family = "binomial",
                                    alpha = 1, lambda = best_lambda_lasso)

# Coefficients
lasso_coefficients <- coef(lasso_logistic_regression)

# Predict probabilities
predicted_prob_las <- predict(lasso_logistic_regression, as.matrix(x_test), type = "response")

# Predict class labels
predicted_outcomes_lasso <- ifelse(predicted_prob_las > 0.5, 1, 0)

# getting accuracy
correct_pred_lasso <- predicted_outcomes_lasso == y_test

# Calculate accuracy
accuracy_log_lasso <- mean(correct_pred_lasso)

accuracy_log_lasso
```

```
## [1] 0.7097561
```

```r
num_errors_lasso_vec <- vector(mode = "numeric", length = length(all_game_days))

make_day_predictions_lasso <- function(data, date) {

  test_start <- which(data$dateGame == date)[1]
  test_end <- tail(which(data$dateGame == date), n=1)
  train_end <- test_start - 1

  train_design_matrix <- data[1:train_end,]
```

```
  test_design_matrix <- data[test_start:test_end,]

  x_train <- train_design_matrix %>%
  select(-slugSeason, -dateGame, -outcomeGame)

  y_train <- train_design_matrix %>%
    select(outcomeGame)

  # Convert every column to numeric
  x_train <- as.matrix(data.frame(lapply(x_train, function(x) as.numeric(as.character(x)))))
  y_train <- as.numeric(y_train$outcomeGame)

  # Perform cross-validation to find the best lambda value for lasso
  cv_fit_lasso <- cv.glmnet(x_train, y_train, family = "binomial", alpha = 1, nfolds = 10)

  # Best lambda value
  best_lambda_lasso <- cv_fit_lasso$lambda.min

  # Fit lasso logistic regression model using the best lambda
  lasso_logistic_regression <- glmnet(x_train, y_train, family = "binomial",
                                      alpha = 0, lambda = best_lambda_lasso)

  x_test <- test_design_matrix %>%
    select(-slugSeason, -dateGame, -outcomeGame)

  y_test <- test_design_matrix %>%
    select(outcomeGame)

  predicted_probs <- predict(lasso_logistic_regression, newx = as.matrix(x_test), type = "response")
  predicted_outcomes <- ifelse(predicted_probs > 0.5, 1, 0)

  y_test_vec <- y_test$outcomeGame

  # number errors
  num_errors_lasso <- sum(predicted_outcomes != y_test_vec)

  confusion_matrix <- table(factor(y_test_vec, levels = c(0, 1)),
                            factor(predicted_outcomes, levels = c(0, 1)),
                            dnn = c("Actual", "Predicted"))

  rownames(confusion_matrix) <- c("0", "1")
  colnames(confusion_matrix) <- c("0", "1")

  confusion_matrix_matrix <- as.matrix(confusion_matrix)

  return_list <- list(confusion_matrix_matrix, num_errors_lasso)

  return(return_list)
}

# initialize list of matrices
lasso_confusion_matrix_list <- vector("list", length = length(unique(design_matrix_2022$dateGame)))
```

```r
# loop through all game days using our design matrix, creating list of confusion matrices for 2022-2023
for (i in 1:length(all_game_days)) {
  temp <- make_day_predictions_lasso(design_matrix, all_game_days[i])
  lasso_confusion_matrix_list[[i]] <- temp[[1]]
  num_errors_lasso_vec[i] <- temp[[2]]
}

lasso_confusion_mat_sum <- Reduce('+', lasso_confusion_matrix_list)

lasso_confusion_mat <- data.frame(
  "Predicted_Labels" = c("Loss", "Win"),
  "Loss" = c(lasso_confusion_mat_sum[1], lasso_confusion_mat_sum[3]),
  "Win" = c(lasso_confusion_mat_sum[2], lasso_confusion_mat_sum[4])
)

library(reshape2)

# Reshape the data
melted_data <- melt(lasso_confusion_mat, id.vars = "Predicted_Labels")
# Rename the columns
names(melted_data) <- c("Predicted Labels", "True Labels", "Count")

# Create the heatmap
ggplot(data = melted_data, aes(x = `Predicted Labels`, y = `True Labels`, fill = Count)) +
  geom_tile() +
  geom_text(aes(label = Count), color = "black", size = 5) +
  scale_fill_gradient(low = "white", high = "steelblue") +
  theme_minimal() +
  labs(title = "Heatmap of Predictions vs True Labels For Lasso Model", x = "Predicted Labels", y = "Tru
  theme(plot.title = element_text(hjust = 0.5))
```

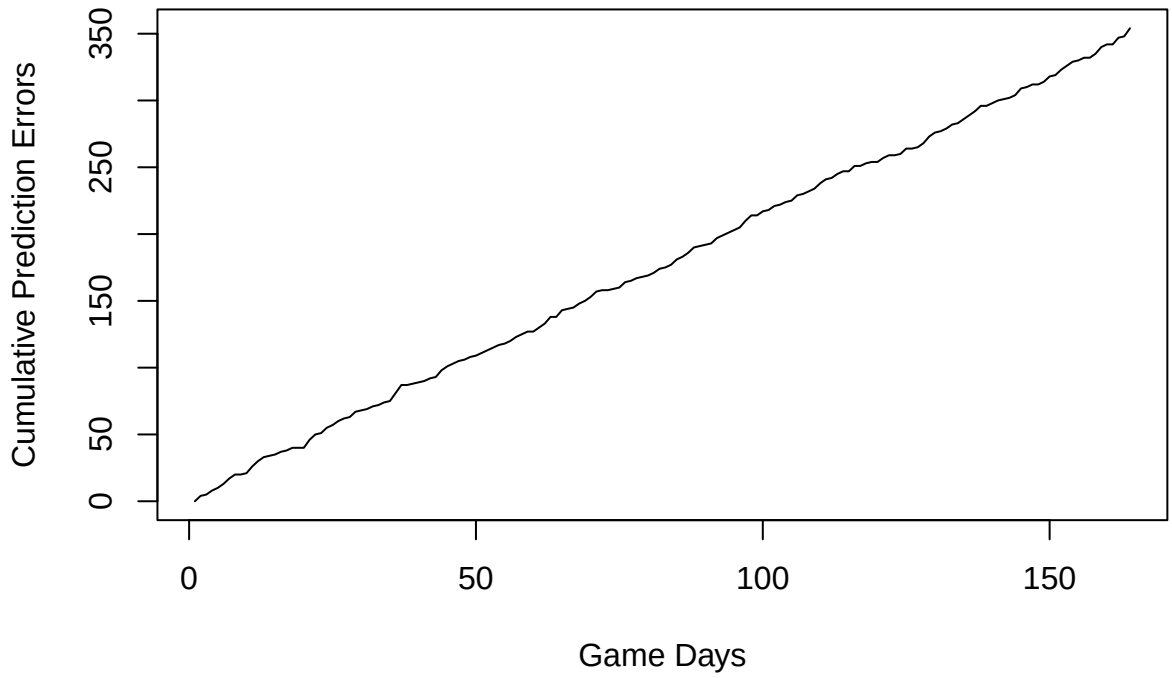



```
TN_lasso <- lasso_confusion_mat_sum[1,1]
FP_lasso <- lasso_confusion_mat_sum[1,2]
TP_lasso <- lasso_confusion_mat_sum[2,2]
FN_lasso <- lasso_confusion_mat_sum[2,1]

lasso_accuracy <- (TN_lasso + TP_lasso)/(TN_lasso + FP_lasso + TP_lasso + FN_lasso)
lasso_accuracy
```

```
## [1] 0.7121951
```

plotting cumulative errors in prediction

```
game_days <- seq(length(all_game_days))
cumu_num_errors_lasso_vec <- cumsum(num_errors_lasso_vec)
# Plot the line graph
plot(game_days, cumu_num_errors_lasso_vec, type = "l", xlab = "Game Days", ylab = "Cumulative Predictio
     main = "Cumulative Prediction Errors for Logistic Lasso Model Over Gamedays", mar = c(5, 5, 4, 2))
```



Model for inference

```
season <- "2022-23"

# Find the first instance of the value in the '2022-2023' column
test_start <- which(design_matrix$slugSeason == season)[1]

train_design_matrix <- design_matrix[1:test_start-1,]
test_design_matrix <- design_matrix[test_start:nrow(design_matrix),]

logit_model_inference <- glm(outcomeGame ~ isB2BSecond + avg_treb + avg_stl + avg_blk + avg_tov +
                      avg_orate + avg_drate + avg_true_s , data = train_design_matrix, family =
                      binomial(link="logit"))

summary(logit_model_inference)
```

```
##
## Call:
## glm(formula = outcomeGame ~ isB2BSecond + avg_treb + avg_stl +
##     avg_blk + avg_tov + avg_orate + avg_drate + avg_true_s, family = binomial(link = "logit"),
##     data = train_design_matrix)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.9321  -0.9362   0.4389   0.8715   2.6159
##
## Coefficients:
```

```
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.324701   0.022131  14.671  < 2e-16 ***
## isB2BSecond -0.248951   0.041797  -5.956 2.58e-09 ***
## avg_treb     0.039044   0.011058   3.531 0.000414 ***
## avg_stl      0.057752   0.016353   3.532 0.000413 ***
## avg_blk      0.006292   0.014668   0.429 0.667951
## avg_tov     -0.045624   0.015903  -2.869 0.004120 **
## avg_orate    0.080766   0.011745   6.876 6.14e-12 ***
## avg_drate   -0.109725   0.005776 -18.997  < 2e-16 ***
## avg_true_s  10.355665   2.137270   4.845 1.26e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 16085  on 11795  degrees of freedom
## Residual deviance: 12804  on 11787  degrees of freedom
## AIC: 12822
##
## Number of Fisher Scoring iterations: 4
```

```
x_test <- test_design_matrix %>%
  select(-slugSeason, -dateGame, -outcomeGame)

y_test <- test_design_matrix %>%
  select(outcomeGame)

# get predictions
predicted_probs_infer <- predict(logit_model_inference, newdata = x_test, type = "response")

# Convert probabilities to binary outcomes using a threshold of 0.5
predicted_outcomes_infer <- ifelse(predicted_probs_infer > 0.5, 1, 0)

# getting accuracy
correct_predictions_infer <- predicted_outcomes_infer == y_test

# Calculate accuracy
accuracy_log_infer <- mean(correct_predictions_infer)

print(accuracy_log_infer)
```

```
## [1] 0.6902439
```

```
num_errors_loginf_vec <- vector(mode = "numeric", length = length(all_game_days))

library(stargazer)
library(lmtest)
# function to return a confusion matrix for all predictions on date with model refit to each new game d
make_day_predictions_inf <- function(data, date, i) {

  test_start <- which(data$dateGame == date)[1]
  test_end <- tail(which(data$dateGame == date), n=1)
  train_end <- test_start - 1
```

```
  train_design_matrix <- data[1:train_end,]
  test_design_matrix <- data[test_start:test_end,]

  logistic_model <- glm(outcomeGame ~ isB2BSecond + avg_treb + avg_stl + avg_blk + avg_tov +
                      avg_orate + avg_drate + avg_true_s,
                    data = train_design_matrix, family =
                      binomial(link="logit"))

  if (i == 1){
    stargazer(logistic_model, align = T, single.row = T, type="latex",
              out="log_first_results.tex")
  }else if(i == 164) {
    stargazer(logistic_model, align = T, single.row = T, type="latex",
              out="log_last_results.tex")
  }

  x_test <- test_design_matrix %>%
    select(-slugSeason, -dateGame, -outcomeGame)

  y_test <- test_design_matrix %>%
    select(outcomeGame)

  predicted_probs <- predict(logistic_model, newdata = x_test, type = "response")
  predicted_outcomes <- ifelse(predicted_probs > 0.5, 1, 0)

  y_test_vec <- y_test$outcomeGame

  # number errors
  num_errors_log_inf <- sum(predicted_outcomes != y_test_vec)

  confusion_matrix <- table(factor(y_test_vec, levels = c(0, 1)),
                            factor(predicted_outcomes, levels = c(0, 1)),
                            dnn = c("Actual", "Predicted"))

  rownames(confusion_matrix) <- c("0", "1")
  colnames(confusion_matrix) <- c("0", "1")

  confusion_matrix_matrix <- as.matrix(confusion_matrix)
  return_list <- list(confusion_matrix_matrix, num_errors_log_inf)

  return(return_list)
}

# get all the game days for 2022-2023 season
design_matrix_2022 <- subset(design_matrix, slugSeason == season)

all_game_days <- unique(design_matrix_2022$dateGame)

# initialize list of matrices
inf_confusion_matrix_list <- vector("list", length = length(unique(design_matrix_2022$dateGame)))

# loop through all game days using our design matrix, creating list of confusion matrices for 2022-2023
for (i in 1:length(all_game_days)) {
```

```
  temp <- make_day_predictions_inf(design_matrix, all_game_days[i], i)
  inf_confusion_matrix_list[[i]] <- temp[[1]]
  num_errors_loginf_vec[i] <- temp[[2]]
}
```

```
##
## % Table created by stargazer v.5.2.3 by Marek Hlavac, Social Policy Institute. E-mail: marek.hlavac a
## % Date and time: Thu, May 11, 2023 - 16:25:15
## % Requires LaTeX packages: dcolumn
## \begin{table}[!htbp] \centering
##   \caption{}
##   \label{}
## \begin{tabular}{@{\extracolsep{5pt}}lD{.}{.}{-3} }
## \\[-1.8ex]\hline
## \hline \\[-1.8ex]
##  & \multicolumn{1}{c}{\textit{Dependent variable:}} \\
## \cline{2-2}
## \\[-1.8ex] & \multicolumn{1}{c}{outcomeGame} \\
## \hline \\[-1.8ex]
##  isB2BSecond & -0.249^{***}$ $(0.042) \\
##   avg\_treb & 0.039^{***}$ $(0.011) \\
##   avg\_stl & 0.058^{***}$ $(0.016) \\
##   avg\_blk & 0.006$ $(0.015) \\
##   avg\_tov & -0.046^{***}$ $(0.016) \\
##   avg\_orate & 0.081^{***}$ $(0.012) \\
##   avg\_drate & -0.110^{***}$ $(0.006) \\
##   avg\_true\_s & 10.356^{***}$ $(2.137) \\
##   Constant & 0.325^{***}$ $(0.022) \\
##  \hline \\[-1.8ex]
## Observations & \multicolumn{1}{c}{11,796} \\
## Log Likelihood & \multicolumn{1}{c}{-6,402.078} \\
## Akaike Inf. Crit. & \multicolumn{1}{c}{12,822.160} \\
## \hline
## \hline \\[-1.8ex]
## \textit{Note:}  & \multicolumn{1}{r}{$^{*}$p$<$0.1; $^{**}$p$<$0.05; $^{***}$p$<$0.01} \\
## \end{tabular}
## \end{table}
##
## % Table created by stargazer v.5.2.3 by Marek Hlavac, Social Policy Institute. E-mail: marek.hlavac a
## % Date and time: Thu, May 11, 2023 - 16:25:18
## % Requires LaTeX packages: dcolumn
## \begin{table}[!htbp] \centering
##   \caption{}
##   \label{}
## \begin{tabular}{@{\extracolsep{5pt}}lD{.}{.}{-3} }
## \\[-1.8ex]\hline
## \hline \\[-1.8ex]
##  & \multicolumn{1}{c}{\textit{Dependent variable:}} \\
## \cline{2-2}
## \\[-1.8ex] & \multicolumn{1}{c}{outcomeGame} \\
## \hline \\[-1.8ex]
##  isB2BSecond & -0.254^{***}$ $(0.040) \\
##   avg\_treb & 0.040^{***}$ $(0.010) \\
```

```
##   avg\_stl & 0.061^{***}$ $(0.016) \\
##   avg\_blk & 0.004$ $(0.014) \\
##   avg\_tov & -0.049^{***}$ $(0.015) \\
##   avg\_orate & 0.080^{***}$ $(0.011) \\
##   avg\_drate & -0.107^{***}$ $(0.005) \\
##   avg\_true\_s & 10.539^{***}$ $(2.007) \\
##   Constant & 0.330^{***}$ $(0.021) \\
##  \hline \\[-1.8ex]
## Observations & \multicolumn{1}{c}{13,011} \\
## Log Likelihood & \multicolumn{1}{c}{-7,099.162} \\
## Akaike Inf. Crit. & \multicolumn{1}{c}{14,216.320} \\
## \hline
## \hline \\[-1.8ex]
## \textit{Note:}  & \multicolumn{1}{r}{$^{*}$p$<$0.1; $^{**}$p$<$0.05; $^{***}$p$<$0.01} \\
## \end{tabular}
## \end{table}
```

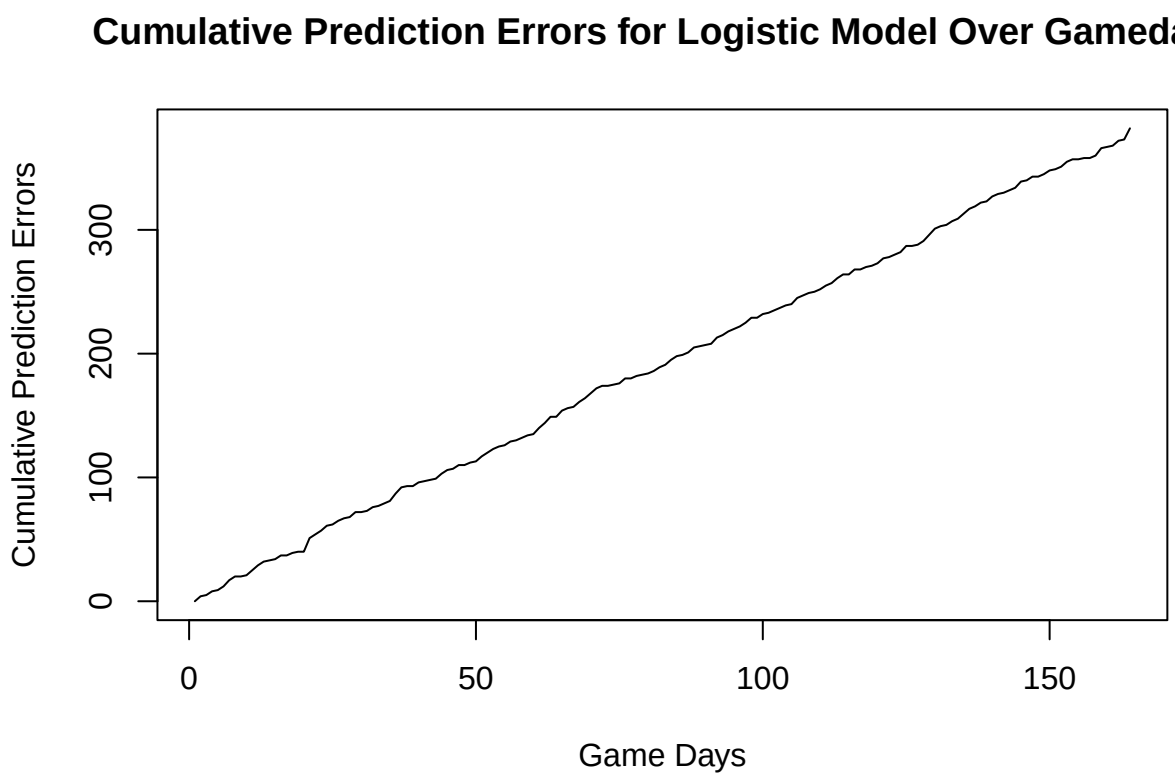
```
inf_confusion_mat_sum <- Reduce('+', inf_confusion_matrix_list)

inf_confusion_mat_sum
```

```
##       Predicted
## Actual   0   1
##      0 295 221
##      1 161 553
```

```
inf_TN <- inf_confusion_mat_sum[1,1]
inf_FP <- inf_confusion_mat_sum[1,2]
inf_TP <- inf_confusion_mat_sum[2,2]
inf_FN <- inf_confusion_mat_sum[2,1]

inf_accuracy <- (inf_TN + inf_TP)/(inf_TN + inf_FP + inf_TP + inf_FN)
inf_accuracy
```

```
## [1] 0.6894309
```

```
cumu_num_errors_loginf_vec <- cumsum(num_errors_loginf_vec)
# Plot the line graph
plot(game_days, cumu_num_errors_loginf_vec, type = "l", xlab = "Game Days", ylab = "Cumulative Predicti
     main = "Cumulative Prediction Errors for Logistic Model Over Gamedays", mar = c(5, 5, 4, 2))
```



fitting some other models for fun.

```
# random forest model
set.seed(254)
library(randomForest)
rf_model <- randomForest(outcomeGame ~ isB2BSecond + avg_treb + avg_stl + avg_blk + avg_tov +
                     avg_orate + avg_drate + avg_true_s + avg_win_perc,
                   data = train_design_matrix)

rf_predictions <- predict(rf_model, x_test)

confusion_matrix <- table(Predicted = rf_predictions, Actual = t(y_test[,1]))
accuracy_rf <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
cat("Random Forest Accuracy: ", accuracy_rf)
```

```
## Random Forest Accuracy:  0.6943089
```

0.704065 accuracy

let's try random forest with only the variables that LASSO chose.

```
# random forest model
set.seed(254)
library(randomForest)
rf_model <- randomForest(outcomeGame ~ isB2BSecond + avg_win_perc,
                  data = train_design_matrix)

rf_predictions <- predict(rf_model, x_test)

confusion_matrix <- table(Predicted = rf_predictions, Actual = t(y_test[,1]))
accuracy_rf <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
cat("Random Forest Accuracy: ", accuracy_rf)
```

```
## Random Forest Accuracy:  0.7097561
```

```
# XGboost model
library(xgboost)

train_design_matrix$outcomeGame <- as.integer(train_design_matrix$outcomeGame) - 1

test_design_matrix$outcomeGame <- as.integer(test_design_matrix$outcomeGame) - 1

train_data <- train_design_matrix %>%
  select(-slugSeason, -dateGame)


test_data <- test_design_matrix %>%
  select(-slugSeason, -dateGame)

# Convert the data to a suitable format for xgboost
train_matrix <- xgb.DMatrix(data = as.matrix(train_data[, -which(names(train_data) == "outcomeGame")]),
                            label = train_data$outcomeGame)
test_matrix <- xgb.DMatrix(data = as.matrix(test_data[, -which(names(test_data) == "outcomeGame")]),
                           label = test_data$outcomeGame)

# set params for xgboost model
params <- list(
  objective = "binary:logistic",
  eval_metric = "error",
  max_depth = 6,
  eta = 0.3,
  min_child_weight = 1,
  subsample = 1,
  colsample_bytree = 1
)

# training the model
xgb_model <- xgb.train(
  params = params,
  data = train_matrix,
  nrounds = 100, # number of boosting rounds
```

```
  watchlist = list(train = train_matrix, test = test_matrix),
  early_stopping_rounds = 10, # stop if no improvement in test set performance after 10 rounds
  print_every_n = 10 # print evaluation metric every 10 rounds
)
```

```
## [1]  train-error:0.242286    test-error:0.300000
## Multiple eval metrics are present. Will use test_error for early stopping.
## Will train until test_error hasn't improved in 10 rounds.
##
## [11] train-error:0.227874    test-error:0.291057
## [21] train-error:0.214564    test-error:0.295935
## Stopping. Best iteration:
## [12] train-error:0.226772    test-error:0.290244
```

```
xg_predictions <- predict(xgb_model, as.matrix(test_data[, -which(names(test_data) == "outcomeGame")]))

# getting accuracy
predicted_labels <- ifelse(xg_predictions > 0.5, 1, 0)
accuracy_xg <- mean(predicted_labels == test_data$outcomeGame)
print(accuracy_xg)
```

```
## [1] 0.7097561
```

0.7081301 accuracy

Naive Bayes

```
# Naive Bayes
library(e1071)
library(caret)
library(lattice)
set.seed(254)

train_data$outcomeGame <- as.factor(train_data$outcomeGame)
test_data$outcomeGame <- as.factor(test_data$outcomeGame)

train_control <- trainControl(method="repeatedcv", number=10, repeats=3)

nb_model <- train(outcomeGame~., data=train_data, trControl=train_control,
                  method="naive_bayes")

nb_predictions <- nb_model %>% predict(test_data[,-2])
accuracy_nb <- mean(nb_predictions == test_data$outcomeGame)
print(accuracy_nb)
```

```
## [1] 0.704878
```

0.7056911 accuracy

SVM

```
library(doParallel)
library(foreach)

cl <- makeCluster(detectCores() - 2)
registerDoParallel(cl)

# Define the parameter grid for the SVM model
parameter_grid <- expand.grid(gamma = 10^seq(-5,5, length.out = 20),
                              cost = 10^seq(-5,5, length.out = 20))

# run tune.svm in parallel
svm_results_radial <- foreach(i = 1:nrow(parameter_grid), .packages = c("e1071")) %dopar% {
  tune.svm(outcomeGame ~ ., data = train_data, kernel = "radial", gamma = parameter_grid[i,"gamma"],
           cost = parameter_grid[i,"cost"], metric = 'accuracy',
           tunecontrol = tune.control(cross = 10))
}

# stop parallel processing
stopCluster(cl)

# Select the best SVM model based on accuracy
error_radial <- sapply(svm_results_radial, function(x) min(x$performances$error))
best_model_radial <- svm_results_radial[[which.min(error_radial)]]

# Retrieve the best parameters from the best SVM model
best_parameters_radial <- best_model_radial$best.parameters
best_parameters_radial

# gamma     cost
# 3.359818e-05 8858.668

# Train the SVM model using the best parameters
svm_radial <- svm(outcomeGame ~ ., data = train_data, kernel = "radial", gamma =
                    best_parameters_radial$gamma, cost = best_parameters_radial$cost)

unregister_dopar <- function() {
  env <- foreach:::.foreachGlobals
  rm(list=ls(name=env), pos=env)
}

unregister_dopar()

# do predictions
radial_predictions <- predict(svm_radial, newdata = test_data[,-2])

# Calculate the accuracy of the model
radial_accuracy <- mean(as.numeric(radial_predictions)-1 == test_data[,2])
print(paste("Radial Accuracy:", radial_accuracy))
```

Radial Accuracy: 0.709756097560976”