

# Stats 191

## Final Project:

### Analysis of Athlete Performance Statistics

A collaboration by:

Peter Kiernan

Nick Panyanouvong

Jodie Meng

Roger Xia

Christopher Noll

# Table of Contents

<b>Dataframe Introduction and Data Cleaning</b>	<b>2</b>
GOAL 1: West-Coast Cup and East-Coast Cup	4
SUMMARY	5
STEP 1: Morning vs. Evening	5
STEP 2: Verifying Model Assumptions	6
STEP 3: Predicting Player Performance	9
GOAL 2: Regular Soccer Season	11
STEP 1: Determining probability of winning against a competitor given team scores	11
Model determination to relate score difference and probability of winning	11
Using MLE to find the cutoff and probability distribution	12
Checking for bias	13
STEP 2: Using probability distribution to find the best teams	14
GOAL 3: Future Coaching Directions	17
<b>Appendix</b>	<b>18</b>

## To Do

Nick: exploratory data analysis, check reference categories

Peter: least square bias, add normal curve in goal 2 first graph

- Export all code as pdf
- Merge Final submission with code pdfs and report

Chris: check for largest betas in normalized model for most impactful covariates (goal 3) ✓

Jodie: Step 3 residuals

Roger: adding t test results

Include table with betas using pairwise t test results

## Plan for submission

1. Submit 3.19 sometime around midnight (text when finish)
2. Submit 3.20 evening, by 9:30pm

# Dataframe Introduction and Data Cleaning

We used the R coding language to perform our statistical analyses and used the following packages in particular: tidyverse, MASS, haven, rstudioapi, and tidyverse.

To begin our analysis, we first compiled three main dataframes:

## 1. All\_games

The All\_games dataframe consists of data for following covariates for 1000 players across 10 try-out matches:

- “game score”: A score between 0 and 100 that evaluates a player’s performance in a particular try-out game.
- “coast”: Describes the coast on which the try-out game took place. Our university is on the west coast, but a few players needed to try-out on the east coast due to travel.
  - Possible values:
    - “east coast”
    - “west coast”
- “game scheduled”: Describes the try-out game’s scheduled time slot.
  - Possible values:
    - “morning”
    - “noon”
    - “evening”
- “percent training sessions attended”: The percentage of university-offered soccer training sessions a player attended. These sessions are mandatory, but attendance was not enforced.
- “overall fitness score”: A score summarising a player’s athletic state, which is measured the morning before each game.
- “# extra strategy sessions attended”: The number of optional strategy sessions a candidate attended each day before the game. Players were highly encouraged to attend these meetings.
- “hours of sleep the night before game”: The number of hours of sleep (rounded to the nearest integer) a player got the night before each try-out game.
- “early bird/night owl”: Describes whether a player is an early bird (sleeps and wakes up early) or a night owl (sleeps and wakes up late).
- “previous competitive sports”: Describes what sports a player has engaged in prior to trying out.
- “# meals on day prior to game”: The number of meals a player consumed before each try-out game.
- “university year”: Describes whether the player is their freshman, sophomore, junior, or senior year of college.

We also added a game-number covariate that ranges from 1 to 10. We used this All\_games dataframe in our R analyses to achieve [Goal 1](#), [Goal 2](#), and [Goal 3](#).

To convert our qualitative covariates into a quantitative format for regression analysis, we wrote indicator functions for the “game scheduled,” “early bird/night owl,” “previous sports,” and “university year” covariates. For the initial three covariates, we used one-hot encoding to denote the appearance of a unique category relative to a reference category, as denoted below.

Covariate	Reference Category
Game scheduled	Noon
Early bird/night owl	Early bird
Previous sports	Football

For the “university year” covariate, we used an indicator function to assign each player a value of 1-4 corresponding to freshman to senior status.

## 2. Match\_ups

The match\_ups dataframe consists of the expected team scores of competitors in the regular college soccer season. We assumed team scores were derived from the average of players’ scores, and we mainly applied this dataframe for [Goal 2](#).

## 3. Previous\_results

The previous\_results dataframe consists of data extracted from previous\_season\_results.csv, which describes the team score of competing teams from the prior season and the corresponding outcome. We mainly applied this dataframe for [Goal 2](#) in order to find a game’s win chance as a function of score.

We were interested in determining the effect of score difference on game outcome, so we added an additional column indicating the difference between scores.

## GOAL 1: West-Coast Cup and East-Coast Cup

Two teams of 10 students each are to be assembled to enter the West-Coast-Cup and East-Coast-Cup, respectively (goalkeepers have already been picked in previous try-outs, and selecting appropriate candidates for substitutes is moved to a separate decision process).

### SUMMARY

In assembling our east and west-coast rosters, we keep one primary consideration in mind: players placed in the west coast league will exclusively play morning games, while those in the east coast league will exclusively play evening games. This motivates an investigation into the differential effects of morning vs. evening games on player scores.

After identifying these differential effects and adjusting our regression model accordingly, we then proceed to probe our data for unusual behavior. Upon verifying that none of our model assumptions are (too seriously) violated, we use it to, for each student, predict their scores under evening vs. morning conditions. The top ten players who score highest in the evening and morning are placed on the east coast and west coast rosters, respectively.

### STEP 1: Morning vs. Evening

We begin by considering an initial regression model, **overallmodel**, that simply regresses game score against the other covariates in **all\_games**. Notably, in selecting the best players, we chose to ignore two covariates deemed as “mutable” - that is, “**percent training sessions attended**” and “**# extra strategy sessions attended**” - as player attendance at either of these can be modified by further instruction from their coach. Therefore, we equalized the values of these two covariates in our model by setting their values (for each player) to the same number.

We then wanted to consider how differences in game scheduling time may influence other covariates in our model. Notably, we hypothesized that player scores in night games may be positively correlated with “**night owl**” habits, while player scores in morning games may be negatively correlated with “**night owl**”.

To verify this, we subsetting our data into two groups: one consisting exclusively of morning scores, and the other of evening scores. We used these two groups to train different regression models for morning scores and evening scores, then examined the covariate coefficients across the two in order to assess if any slopes were significantly different. We conducted pairwise Z-tests for coefficient differences between the two models (using the Bonferroni correction to adjust for multiple testing), and our analysis revealed that the only coefficient significantly different between the two models was that corresponding to **night owl**. The results of those tests are shown below.

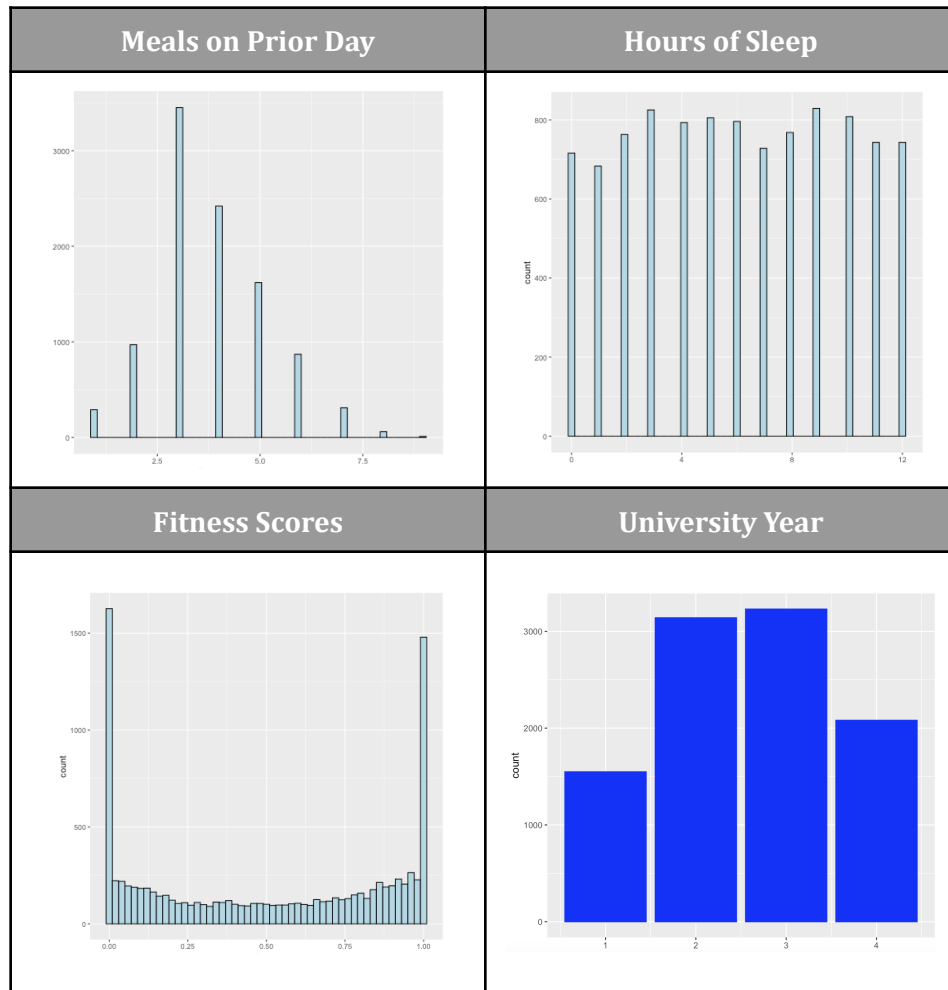
	z statistic	p-value	significant?
(Intercept)	3.3902912	6.981841e-04	Yes
`percent training sessions attended`	0.1964102	8.442891e-01	No
`overall fitness score`	-0.5311043	5.953465e-01	No
`# extra strategy sessions attended`	-2.0621411	3.919431e-02	No
`hours of sleep the night before game`	-1.2012703	2.296464e-01	No
`# meals on day prior to game`	-0.1491282	8.814525e-01	No
`university year`	-1.2314218	2.181651e-01	No
curling	2.6734191	7.508238e-03	No
gymnastics	1.0019243	3.163801e-01	No
baseball	2.8829042	3.940273e-03	No
martial_arts	1.6422567	1.005368e-01	No
frisbee	1.9428710	5.203175e-02	No
table_tennis	3.3925747	6.923905e-04	Yes
basketball	1.4153805	1.569570e-01	No
night_owl	-14.2528688	4.302371e-46	Yes

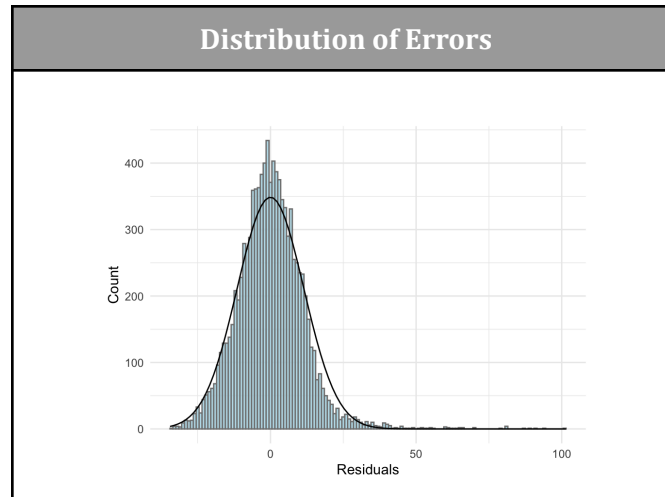
Only two covariates exhibited significant coefficient differences between models: **night\_owl** and **table\_tennis**. However, because the effect size of **night\_owl** was much larger than that of **table\_tennis**, we decided only to make adjustments based on the former.

We concluded that “night\_owl” interacts differently with “morning” and “evening” covariates, and accordingly incorporated additional slope interaction terms **morning \* night\_owl** and **evening \* night\_owl** in **overallmodel** to account for these nuances.

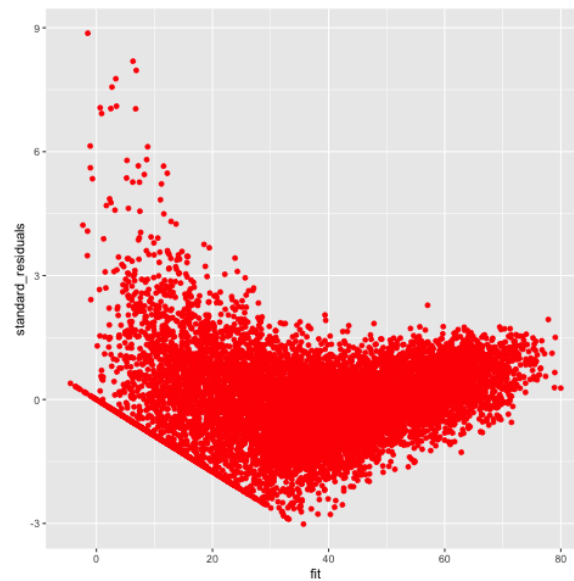
## STEP 2: Verifying Model Assumptions

Before applying this model, our next course of action was to identify any irregular behavior. We started by analyzing plots of each covariate against player score in order to identify nonlinear trends or outliers. We started by modeling the frequency distributions of our covariates, as well as our residuals. These are shown below:



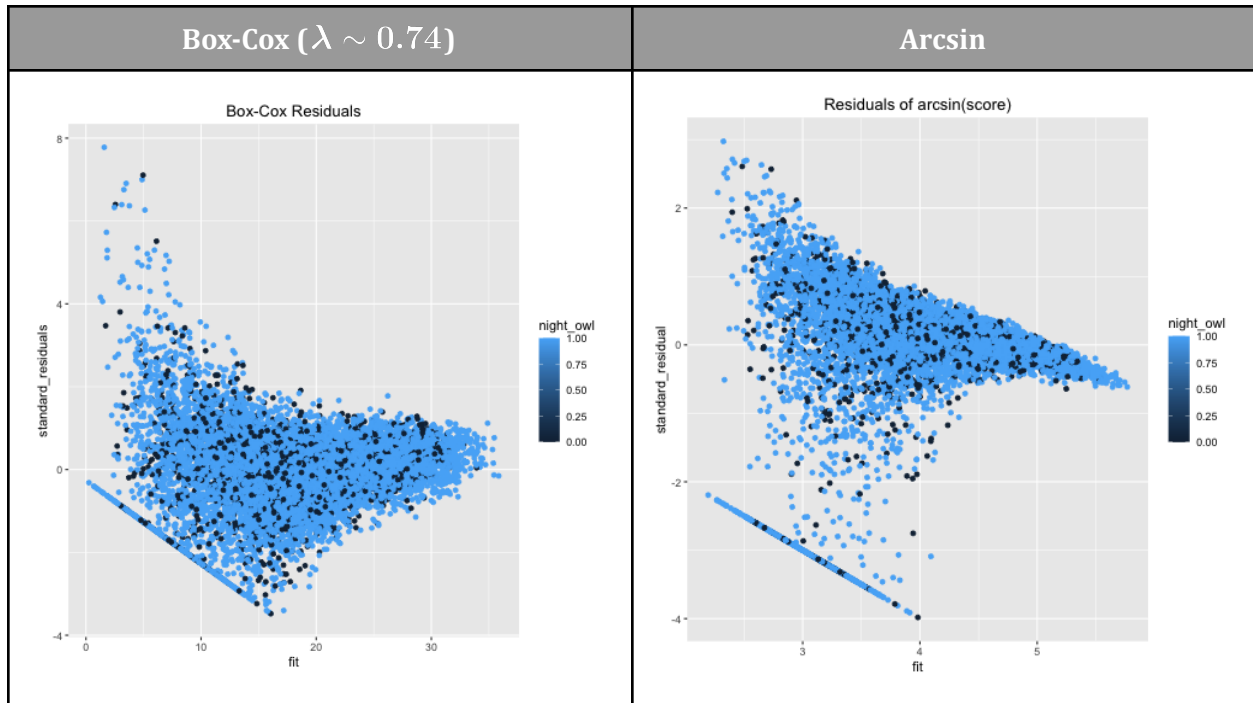


There are two notable takeaways. The first is that the distribution of our errors roughly follows a Normal curve, so the Gaussian error assumption is valid. The second observation is that the frequency distribution for fitness score (which one might expect to play a significant role in predicting player performance), is noticeably bimodal. This bimodal distribution manifests itself clearly in a general residual plot of **overallmodel**, which displays marked heteroskedasticity and nonlinearity.



The residual plot indicates high residual variance for lower predicted scores. In order to address this heteroskedasticity, we attempted several transformations of the data, including Box-Cox and arcsin:





Unfortunately, none of our candidate transformations were able to address the observed heteroskedasticity, nor were we able to devise a covariate transformation that could address the bimodal distribution of **fitness score**.

Notably, because the heteroskedasticity of our plot is most pronounced for lower predicted scores, it should not have a significant impact on our determination of the most qualified players. We also checked for collinearity, and found no variance inflation factors greater than 5. Thus, we decided to proceed with our analysis.

### STEP 3: Predicting Player Performance

To predict player performance for our East-Coast or West-Coast cups, we generated two datasets that predicted players' game scores across all 10 try-out games in morning and evening conditions.

To do this, we made two hypothetical versions of **all\_games** where game was played in the morning or the evening. Thus, in our `morningized_games` dataset, all observations had a morning indicator value of 1 and an evening value of 0, while the converse was true for our `eveningized_games` dataset. We then applied **overall model** to the morningized game data and eveningized game data, such that we generated predicted scores across 10 try-out games.

Afterward, we generalized player performance by calculating their average predicted score across the 10 try-out games. We then selected the top 20 highest-scoring players across

morningized\_games and eveningized\_games for consideration into our 10-member West-Coast and East-Coast teams.

For each high-scoring player, we compared their predicted average morning and evening game scores. Based on whichever game score was higher, we added them to appropriate roster (West-Coast: morning, East-Coast: evening). As our final step, we filtered teams and selected the top 10-highest performing players for our final roster.

Here are our two rosters:

The 2023 West-Coast Team:

1. Player 794
2. Player 461
3. Player 253
4. Player 359
5. Player 372
6. Player 320
7. Player 201
8. Player 761
9. Player 652
10. Player 791

The 2023 East-Coast Team:

1. Player 785
2. Player 660
3. Player 824
4. Player 235
5. Player 778
6. Player 180
7. Player 89
8. Player 760
9. Player 631
10. Player 129

## GOAL 2: Regular Soccer Season

The regular college soccer season consists of twenty games, and the head-coach has obtained information on the expected player-score for each of these twenty opposing teams. They'd like you to suggest two candidate rosters: one that minimises the chances of yielding a single game, and one that maximizes the number of expected wins.

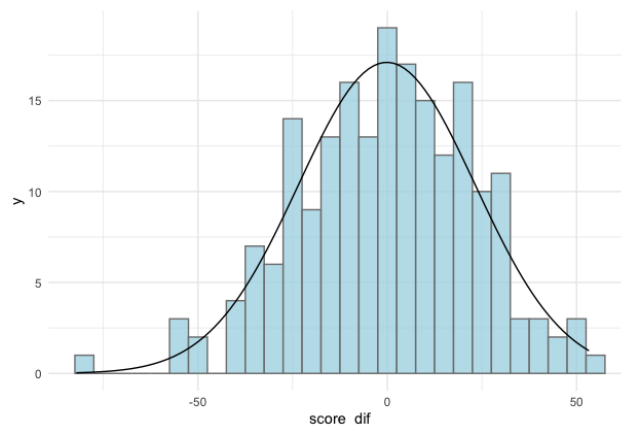
To create the two candidate rosters for the regular soccer season, we must employ two different objective functions. The first objective is equivalent to maximizing the probability of winning all games. Assuming  $P_n$  represents the probability of a particular team winning a single game,  $n$ , then

first task is equivalent to maximizing  $\prod_{n=1}^{20} P_n$ . Further, the second task is equivalent to maximizing

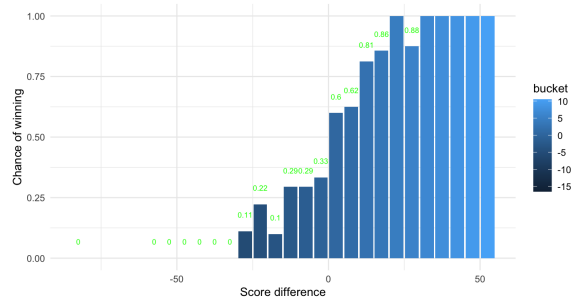
$\sum_{n=1}^{20} P_n$ , which represents the number of expected wins given each  $P_n$ .

### STEP 1: Mapping Team Score Differences to Probability of Winning

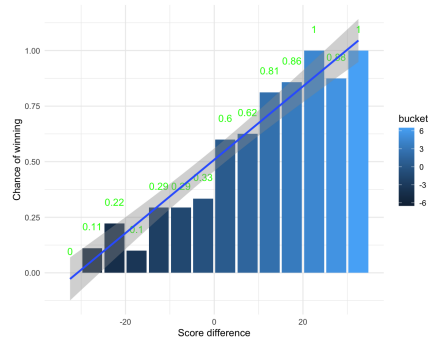
In order to maximize either objective function, we must determine  $P_n$  given the score difference between the team of interest and a competing team. We must first start by determining the appropriate model to relate score difference and probability of winning. This can be achieved by visualizing the distribution of score differences and outcomes within the previous\_results dataframe. Plotting the score difference against their frequencies in the prev\_games dataframe, we can see that the distribution is approximately normal.



From further computation of summary statistics, we know that the mean is 0.116 with standard error of 0.116 points. We can also observe that the majority of score differences falls between -35 and +35 points. Now, we are interested in plotting the score differences between a given team and competitor against the probability of winning.



We can see from the following that for a score difference less than -35 points, a loss is almost always observed, while for score difference greater than 35 points, a win is almost always guaranteed. Thus, we can model the relationship between score difference and outcome as piecewise, such that below a certain cutoff, a loss is always observed, while above another cutoff, a win is always observed. Between the cutoffs, the relationship can be modeled linearly.

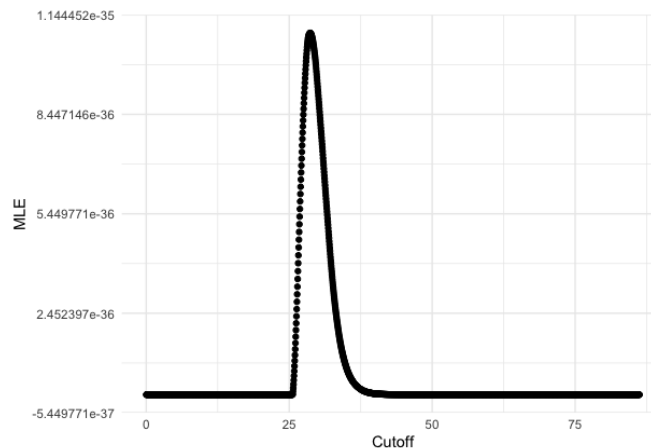


## Using MLE to find the cutoff and probability distribution

However, we want to find the best cutoff more objectively than estimation. To do this, we will use the maximum likelihood estimation as our objective function, as least squares ended up being both biased and not a good fit for discrete data like this. Then, our MLE is the product of the chance of losing for all the games we lost with the chance of winning for all the games we won—equivalently:

$$P_{\beta} = \prod_{i=1}^n \left( P(x_i)^{y_i} (1 - P(x_i))^{1-y_i} \right)$$

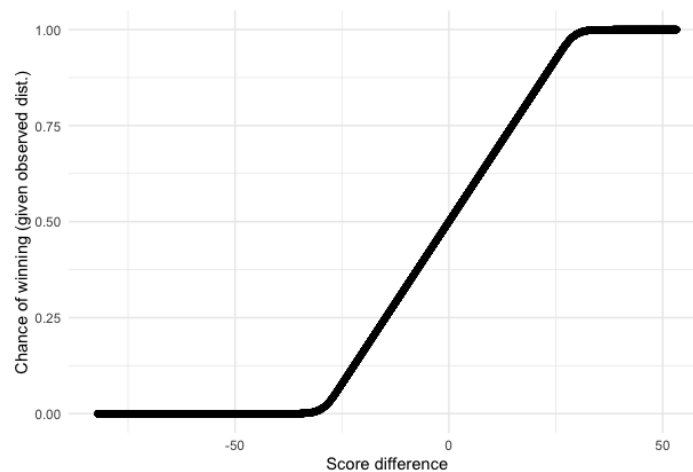
Then, if we plot this probability for each prospective cutoff:



Then, we can clearly see one maximum in a slightly skewed right distribution. This maximum ends up being 28.634; but we're not interested in estimating the true probability distribution using just one cutoff. We need to accommodate for the uncertainty in the cutoff—we don't have a 100% chance of winning at a score difference of 28.635, because there's a reasonable chance the true score cutoff is 30. So, we can do the weighted average of the predicted chance of winning for each score difference at each cutoff  $C_i$  against the likelihood of that cutoff. We want to give each probability a weight equal to its likelihood, but normalize by the total likelihood:

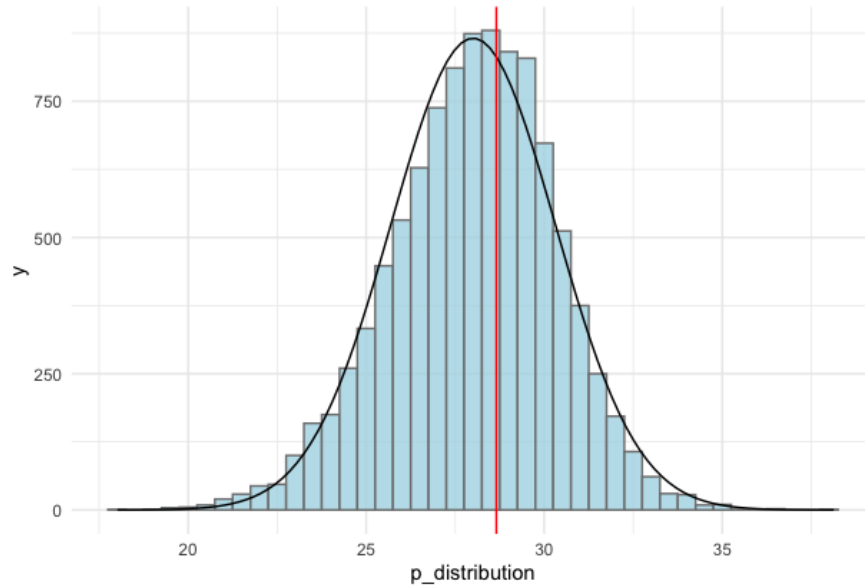
$$\hat{P}(x) = \frac{\sum_{i=1}^N MLE(C_i) P_{C_i}(x)}{\sum_{i=1}^N MLE(C_i)}$$

This gives us the probability of winning at each score difference under this model given our observed distribution:



## Checking for bias

From here, we simulated data akin to ours: 200 datapoints, at normally distributed score differences with a mean of 0 and standard deviation of 23.3 (like the real data, which had a standard deviation of 23.3 and a mean that was not significantly different from the expected 0). We simulated the result of each game using our measured cutoff of 28.635. We then ran this same MLE to find an estimated cutoff for each, and made a histogram of the results for 10,000 such regressions and datasets.



This is a histogram of the frequency at which each cutoff bucket occurred, with the true cutoff marked in red and a normal distribution with the mean and standard deviation of the data added. As we can see, the data is approximately normal, with a slight skew on the mean but no skew on the mode.

## STEP 2: Identifying the Best Teams

From here, we want to see how variable and how skilled each player is. To find these means and standard deviations, we used the same technique as before to create datasets of all the games adjusted as if they were morning games, and similarly for noon and evening. We also maximized the “percent training sessions attended” and “# extra strategy sessions attended” variables, as previously discussed, because these can be controlled easily by the coach. We then took the mean and standard deviation of the estimated score for each of these 30 games for each player. Under the Gaussian error assumption, we can then model each player’s score distribution as a normal distribution about that mean and with that standard deviation.

Then, the overall game performance of a team is the average of the performance of all the players, so it is also a Gaussian distribution with variance equal to the sum of the variance of each of the players. To see the probability of beating a score  $S$ , we need to evaluate the probability of team A beating that team. That is, for each possible score  $x$ , we want to multiply the chance of them getting that score (or rather, an infinitesimal distance about that score) by the chance of them winning if they get that score (using the probability distribution from the last step), and add up all these infinitesimal probabilities. That evaluates to:

$$P_A(C) = \int_0^{100} N(x)P(x - C)dx \approx \Delta x \sum_{i=0}^{\frac{100}{\Delta x}} N(x)P(x - C)$$

Where  $N(x)$  is the normal PDF describing the chance of the team scoring score  $x$ , and  $P$  is the score distribution from before. Because we don't have the tools to compute this analytically, we need to approximate this with relatively small  $\Delta x$ . We can repeat this to find the probabilities against each of the 20 teams, and add or multiply them depending on which measure we are using.

From here, what we really want to do is set up the 1000 choose 10 teams ( $\sim 2.6 \cdot 10^{23}$  teams) that we can possibly make out of the participating players, and see which one performed best in each measure. However, that is impractical, and even reduced versions of this task, like taking the top  $n$ th percentile, severely limit how many players can be considered due to computation time.

However, we can cut down on computation time through one simple principle. We should never pick a player for a potential team if there is a player with a similar standard deviation in their scores but a higher mean. This is because we know higher means are always better, but we don't know how the tradeoff between mean and standard deviation behaves. If your mean is higher than the other team's, you want a lower standard deviation, so you will win more consistently; if your mean is lower than the other teams, you want a higher standard deviation, so that you have a better chance of getting lucky and beating them. But, of course, the opposing team's score is different from game to game, so we need to try all sorts of standard deviations.

To implement this, we divided the prospective players into standard deviation buckets, like a histogram. We used buckets of width 2.5 starting at the lowest standard deviation, both because that was both computationally feasible and narrow enough to capture relatively good resolution on the standard deviation. These buckets will define what is a 'similar standard deviation'—we will never use a player on a potential team if there is an unused player in their bucket with a higher mean.

Then, we kept only the top 10 players in each bucket, because we'll never use the 11th player. The number of teams we can possibly make by taking top players from each bucket such that we have a total of 10 comes out to be only  $\sim 370,000$ . To keep the processing time on this manageable, we did an initial run with  $\Delta x = 1$ , over which it took 13.5 hours to process all teams of this type. We then took the top 7,000 teams that performed best in each measure, and recalculated their performance to as high of precision as possible. From there, the two teams that performed best are the two teams that were highest performing in each measure at the end of this—these two rosters are:

The Candidate Roster that *Minimizes* the Chances of Yielding a Single Game: (has a  $7.77 \cdot 10^{-6}$  chance of going undefeated, or about 1 in 128,000)

1. Player 526
2. Player 941
3. Player 345
4. Player 580
5. Player 329
6. Player 441
7. Player 264
8. Player 632
9. Player 971
10. Player 535

The Candidate Roster that *Maximizes* the number of expected wins (expected to win 10.84 of the 20 games given)

1. Player 526
2. Player 941
3. Player 345
4. Player 580
5. Player 329
6. Player 441
7. Player 264
8. Player 367
9. Player 582
10. Player 535



### GOAL 3: Future Coaching Directions

How to best coach a player is a complicated question, with strategies so far being based on intuition more than systematic exploration. The head-coach would like you to use available data to inform them on which player-covariates their coaching program should pay most attention to, once a student has been recruited to the team.

Given our statistical analyses, the coaching staff must focus on optimizing the physical prowess of their players to optimize their players' performance. Namely, during our analysis, we found that **`percent training sessions attended`** and **`overall fitness score`** had the two largest significant positive effects on game performance. Two other variables that had a smaller positive impact on player performance were **`# extra strategy sessions attended`** and **`hours of sleep the night before game.`** The precise values of the coefficients, as well as the results of marginal hypotheses tests with  $\mathbb{H}_0: \hat{\beta}_k = 0$ , are shown in the table below.

Coefficients:

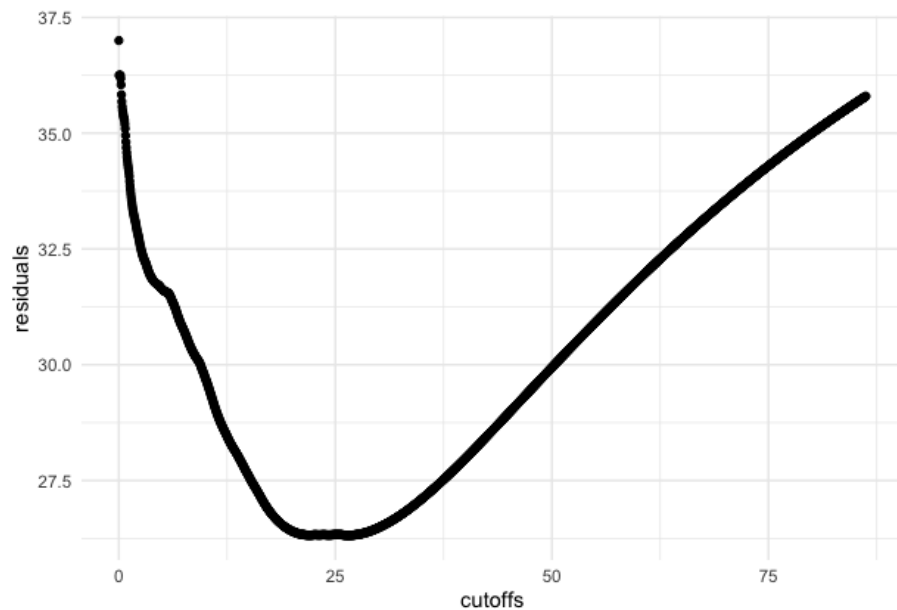
	Estimate	Std. Error	t value	Pr(> t )	
(Intercept)	6.32779	0.67586	9.363	< 2e-16	***
`percent training sessions attended`	27.20260	0.32881	82.730	< 2e-16	***
`overall fitness score`	24.71562	0.32079	77.045	< 2e-16	***
`# extra strategy sessions attended`	1.41472	0.04184	33.814	< 2e-16	***
`hours of sleep the night before game`	0.32412	0.03382	9.585	< 2e-16	***
`# meals on day prior to game`	-1.34705	0.09274	-14.525	< 2e-16	***
`university year`	-0.45318	0.13421	-3.377	0.000736	***
evening	-0.29225	0.70325	-0.416	0.677732	
morning	7.55333	0.94946	7.955	1.98e-15	***
night_owl	1.02835	0.37876	2.715	0.006638	**
morning:night_owl	-7.98685	1.04697	-7.629	2.59e-14	***
evening:night_owl	7.25411	0.77177	9.399	< 2e-16	***

As a result, the coaching team should do everything in their power to ensure all their players attend training sessions and do everything possible to elevate their players' overall fitness score (e.g., crack down on no-shows and pushing their players during training sessions). To truly optimize their players' performance, they should also mandate strategy sessions and enforce curfews to ensure their players gain what they can from better strategy and more sleep.

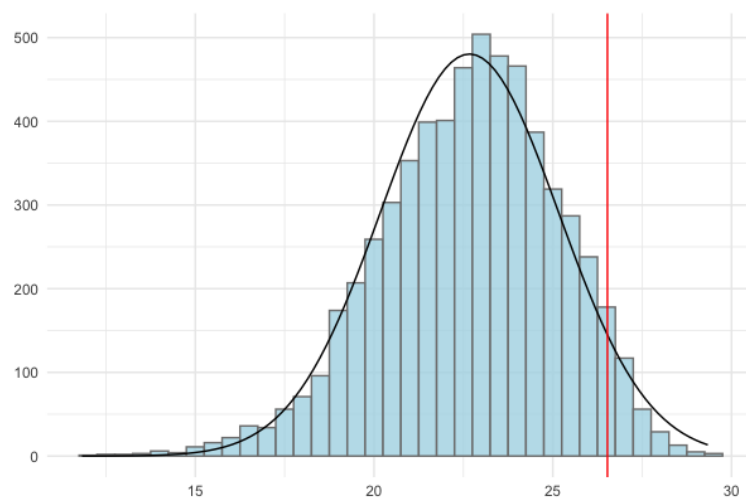
# Appendix

## Bias of Least Squares

Initially, instead of using the MLE as our objective function, we used vertical distance squared. Its objective function looked like this:



Which is a little strange in that it has a wide, flat trough, but not that weird. However, when we ran simulations, as we did with the MLE, to simulate new data, we observed quite significant bias:



The red line is again the true value used in the simulations, and y axis is again the frequency. Thus, likelihood was a significantly better measure.

This report is automatically generated with the R package [knitr](#) (version 1.41) .

```
#### Functions

### Graphing

# Generates and saves multiple histograms for the specified COLS of the dataframe
multi_histogram <- function(data, cols){

  for(column in cols){

    png(paste("../OUTPUTS/", column, ".png"))

    bin_width <- (max(data[[column]]) - min(data[[column]])) / 50 # Adjust bin width
    histo <- ggplot(data = data, aes(x = !!sym(column))) +
      geom_histogram(binwidth = bin_width, color = 'black', fill = 'lightblue') +
      ggtitle(paste(column, " distribution")) +
      theme(plot.title = element_text(hjust = 0.5, vjust = 0.5))

    print(histo)

    dev.off()
  }
}

### Utility stuff

# Just returns a list of z-scores from a list
normalize <- function(statistic){
  return ((statistic - mean(statistic))/var(statistic))
}

# Returns the mode because that doesn't exist by default for some reason
Mode <- function(statistic) {
  uniques <- unique(statistic)
  uniques[which.max(tabulate(match(statistic, uniques)))]
}

# Determine if an element matches the specified sport
match_sport <- function(strings, sport){
  results <- c()
  for(string in strings){
    if (string == sport){
      results <- c(results, 1)
    } else {
      results <- c(results, 0)
    }
  }

  return(results)
}

### Bucketing to build teams efficiently

# Partitions a dataframe into buckets by one variable, while
# each bucket is organized in descending order by another.

# Takes column names in strings.

partition <- function(df, bucketvar, sortvar, bucketsize, maxsize) {
  buckets <- list()
  lower <- min(df[[bucketvar]])
  while(lower <= max(df[[bucketvar]])) {
    temp <- filter(df,!!as.symbol(bucketvar) >= lower,
      !!as.symbol(bucketvar) < lower + bucketsize)
    temp <- temp[order(temp[[sortvar]], decreasing = TRUE),]
    temp <- filter(temp[1:maxsize,],!is.na(!!as.symbol(bucketvar)))
    buckets[[length(buckets) + 1]] <- temp[order(temp[[sortvar]], decreasing = TRUE),]
    lower <- lower + bucketsize
  }
  return(buckets)
}
```

```

# Finds the last index of a vector that is nonzero, excluding the
# index at the end of the vector. (Very specific but useful here.)
last_nonfinal_nonzero <- function(vec) {
  i <- length(vec) - 1
  while(i > 0){
    if(vec[i] != 0) {
      return(i)
    }
    i <- i - 1
  }
  return(0)
}

# Grabs given numbers of players from the appropriate buckets
make_team <- function(amounts, buckets) {
  if(length(amounts) != length(buckets)) {
    print(amounts)
    print(length(buckets))
    stop("Amounts not as long as buckets!")
  }
  output <- NULL
  for(i in 1:length(buckets)) {
    if(amounts[i] > 0) {
      temp <- buckets[[i]]
      output <- rbind(output,temp[1:amounts[i],])
    }
  }
  return(output)
}

# Takes in a list of amounts and the buckets they belong to, and
# increments it by one, moving the rightmost entry right by one,
# or if it's at the last spot, moving the next rightmost right by
# one and resetting everything at the last spot to that spot.

# Example: 2 0 0 0 -> 1 1 0 0 -> 1 0 1 0 -> 1 0 0 1 -> 0 2 0 0

increment_amounts <- function(buckets,bucket_amounts) {
  last <- last_nonfinal_nonzero(bucket_amounts)
  # Reset all the ones at the end to the new beginning, after
  # incrementing the appropriate 'turtle' counter
  add <- bucket_amounts[length(bucket_amounts)]
  bucket_amounts[length(bucket_amounts)] <- 0
  # If everything is in the last cell, return null to signal
  # that we're done
  if(last == 0) {
    print("All at the end!")
    return(NULL)
  }
  bucket_amounts[last] <- bucket_amounts[last] - 1
  bucket_amounts[last + 1] <- 1 + add
  # Not all the buckets are the same length, need to make them fit
  cur <- last + 1
  while(bucket_amounts[cur] > nrow(buckets[[cur]])) {
    # If we're at the last one, we need to roll over by incrementing
    # again
    if(cur == length(bucket_amounts)){
      # Normal increment stuff
      add <- bucket_amounts[length(bucket_amounts)]
      bucket_amounts[length(bucket_amounts)] <- 0
      cur <- last_nonfinal_nonzero(bucket_amounts)
      # If everything is in the last cell, return as usual
      if(cur == 0) {
        return(NULL)
      }
      bucket_amounts[cur] <- bucket_amounts[cur] - 1
      bucket_amounts[cur + 1] <- 1 + add
    } else {
      # Just slide the excess over to the next bucket
      bucket_amounts[cur + 1] <- bucket_amounts[cur + 1] + bucket_amounts[cur] - nrow(buckets[[cur]])
      bucket_amounts[cur] <- nrow(buckets[[cur]])
    }
  }
  cur <- cur + 1
  # Repeat to make sure there's enough space in the
  # bucket we slid the excess into

```

```

}
return(bucket_amounts)
}

# Tries all combinations of these players, but you never use
# a player unless all higher-mean players from their sd bucket
# have already been used. Takes a list of dataframes, like above,
# along with auxiliary inputs.

# num_sd is the number of standard deviation away from the mean to
# calculate over

bucket_combo_teams <- function(bucketlist, num_sd, odds, score_difs,
                                team_scores, rounding_places, totalruns) {
  # This keeps track of how many people we are pulling from each bucket
  bucket_amounts <- numeric(length(bucketlist))
  bucket_amounts[1] <- 10
  player1 <- c()
  player2 <- c()
  player3 <- c()
  player4 <- c()
  player5 <- c()
  player6 <- c()
  player7 <- c()
  player8 <- c()
  player9 <- c()
  player10 <- c()
  avg_wins <- c()
  undefeated_chance <- c()
  i <- 0
  while(!is.null(bucket_amounts)) {
    players <- make_team(bucket_amounts, bucketlist)
    player1[length(player1)+1] <- players$`student label`[1]
    player2[length(player2)+1] <- players$`student label`[2]
    player3[length(player3)+1] <- players$`student label`[3]
    player4[length(player4)+1] <- players$`student label`[4]
    player5[length(player5)+1] <- players$`student label`[5]
    player6[length(player6)+1] <- players$`student label`[6]
    player7[length(player7)+1] <- players$`student label`[7]
    player8[length(player8)+1] <- players$`student label`[8]
    player9[length(player9)+1] <- players$`student label`[9]
    player10[length(player10)+1] <- players$`student label`[10]

    # We can model the distribution of their average easily,
    # because each of their score distributions are normal (see graph)
    team_pdf <- score_pdf_from_players(players$mean,players$sd)
    team_cdf <- score_cdf_from_players(players$mean,players$sd)
    results <- success_measures(odds, score_difs, team_pdf, team_cdf,
                                team_scores, rounding_places,
                                mean(players$mean) - num_sd * sqrt(sum(players$sd^2)),
                                mean(players$mean) + num_sd * sqrt(sum(players$sd^2)))
    avg_wins[length(avg_wins)+1] <- results[1]
    undefeated_chance[length(undefeated_chance)+1] <- results[2]
    bucket_amounts <- increment_amounts(bucketlist,bucket_amounts)
    i <- i + 1
    if(i %% round(totalruns/100,0) == 0) {
      print(paste(round(i/totalruns * 100,1), "%", "done"))
    }
  }
  print(paste("Ran", i, "times"))
  return(data.frame(player1,player2,player3,player4,player5,player6,
                    player7,player8,player9,player10,avg_wins,
                    undefeated_chance))
}

### Piecewise regression stuff

## Piecewise utility functions

# Just takes in a cutoff to determine the model and an input
# to evaluate it at and returns appropriates
piecewise_predict <- function(input, cutoff) {
  if(input <= -1*cutoff) {
    return(0)
  }

```

```

    if(input >= cutoff) {
      return(1)
    }
    return(0.5 + input/(2*cutoff))
  }
}

# Same as above but optimized for vectors - significant
# time reduction
piecewise_multipredict <- function(input_vec, cutoff) {
  output = vector(mode(input_vec),length(input_vec))
  output[input_vec <= -1*cutoff] <- 0
  output[input_vec >= cutoff] <- 1
  parsed <- input_vec[input_vec < cutoff]
  output[input_vec < cutoff][parsed
    >-1 * cutoff] <- 0.5 + input_vec[input_vec < cutoff][parsed
    >-1 * cutoff]/(2*cutoff)
  return(output)
}

# These three just define the lower bound of where we should
# bother searching for cutoffs
lower_limit <- function(dependent, independent) {
  return(min(abs(last_loss(dependent,independent)),abs(first_win(dependent,independent))))
}

last_loss <- function(dependent, independent) {
  return(min(independent[dependent == 1]))
}

first_win <- function(dependent, independent) {
  return(max(independent[dependent == 0]))
}

### Objective functions

# Notice how these have the same arguments, and both want to be
# minimized, so we can treat them the same in our functions

# This one caused bad skew but is included for defending why we
# didn't use it - it's vertical distance squared
piecewise_residual_square <- function(dependent, independent, cutoff) {
  return(sum((dependent - piecewise_multipredict(independent,cutoff))^2))
}

# This one is significantly better, it's the chance of the observed
# game results happening in a particular model
piecewise_probability <- function(dependent, independent, cutoff) {
  return(-1 * abs(prod(1 - dependent - piecewise_multipredict(independent,cutoff))))
}

# Finding residuals for each cutoff

multiple_piecewise_residual <- function(dependent, independent, steps, objective_func) {
  return(targeted_piecewise_residual(dependent, independent, steps, 0, max(max(independent),max(-1*independent))))
}

targeted_piecewise_residual <- function(dependent, independent, steps, lower, upper, objective_func) {
  step <- (upper-lower)/steps
  cutoffs <- c()
  residuals <- c()
  for(i in 0:steps) {
    cutoffs <- append(cutoffs,lower + step * i)
    residuals <- append(residuals,objective_func(dependent, independent, lower + step * i))
  }
  return(data.frame(cutoffs,residuals))
}

# Using these to find the best cutoff

optimize_cutoff <- function(dependent, independent, objective_func) {
  cutoff <- find_cutoff(dependent, independent, 1000,
    lower_limit(dependent,independent) * 0.95,
    max(abs(independent)) * 1.05, objective_func)
  cutoff <- find_cutoff(dependent, independent, 1000,
    cutoff + 5, cutoff - 5, objective_func)
}

```

```

cutoff <- find_cutoff(dependent, independent, 400,
                      cutoff + 0.1, cutoff - 0.1, objective_func)
return(round(cutoff,3))
}

find_cutoff <- function(dependent, independent, steps, lower, upper, objective_func) {
  residual_by_cutoff <- targeted_pieewise_residual(dependent, independent, steps, lower, upper, objective_
  return(mean(residual_by_cutoff$cutoffs[which(residual_by_cutoff$residuals == min(residual_by_cutoff$resid
})

## Simulating samples to test for bias

# Represents a single game, randomly determines the result
# based on the model
simulate_pieewise_sample <- function(input, cutoff) {
  return(sample(c(0,1), size = 1,prob = c(1-pieewise_predict(input, cutoff),pieewise_predict(input, cuto:
})

# Based on a normal distribution of score differences, this
# uses the above to generate many games in a given model
simulate_pieewise_distribution <- function(cutoff, n, score_sd) {
  score_difs = rnorm(n, sd=score_sd)
  winners = c()
  for(item in score_difs) {
    winners <- append(winners, simulate_pieewise_sample(item, cutoff))
  }
  return(data.frame(score_difs, winners))
}

# Can run ~1000 sims of n=200 per minute on residual sqaured,
# and the same in 30 seconds on probability
cutoff_distribution <- function(true_cutoff, numsims, numsamples, score_sd, objective_func) {
  measured_cutoffs <- c()
  for(i in 1:numsims) {
    score_by_winners <- simulate_pieewise_distribution(true_cutoff,
                                                       numsamples, score_sd)
    measured_cutoffs <- append(measured_cutoffs,optimize_cutoff(score_by_winners$winners,
                                                                score_by_winners$score_difs,
                                                                objective_func))

    if(i %% (numsims/100) == 0) {
      print(paste(i/numsims*100,"%"))
    }
  }
  return(measured_cutoffs)
}

### Finding the best team for normal season

# Takes in lists of win chances and score differences, an input, and
# the number of digits the entries in the dataframe are rounded to,
# and outputs the associated win chance

confidence_predict <- function(odds, score_difs, input, rounding_places) {
  if(round(input,rounding_places) %in% score_difs) {
    return(odds[score_difs
              == round(input, digits = rounding_places)])
  }
  if(round(input,rounding_places) < min(score_difs)) {
    return(odds[1])
  }
  if(round(input,rounding_places) > max(score_difs)) {
    return(odds[length(odds)])
  }
  print("confidence_predict error! Score_difs have gaps")
}

# Takes in a normalized probability density function for score_dif
# and outputs the chances of winning. Also needs auxiliary inputs
# like rounding places and the chances of winning at score_difs,
# along with the bounds to search over

# score_dif_density_func should output probability density as a function
# of score_dif

```

```

distribution_win_chance <- function(odds, score_difs, score_dif_density_func,
                                   score_dif_cdf, rounding_places, low, high) {
  if(low > 0) {
    print("Distribution win chance low end must be below 0!")
    low <- -1
  }

  cur_score_dif <- 0
  step_size <- 10^{-1*rounding_places}
  result <- 0
  for(i in 0:ceiling((high)/step_size)) {
    if(confidence_predict(odds,score_difs,cur_score_dif,rounding_places)
       > 0.999) {
      result <- result + score_dif_cdf(cur_score_dif)
      break
    }
    result <- result + step_size * score_dif_density_func(cur_score_dif) *
      confidence_predict(odds,score_difs,cur_score_dif,rounding_places)
    cur_score_dif <- round(cur_score_dif + step_size,rounding_places)
  }
  cur_score_dif <- 0
  for(i in 0:ceiling((low)/step_size)) {
    if(confidence_predict(odds,score_difs,cur_score_dif,rounding_places)
       < 0.001) {
      break
    }
    result <- result + step_size * score_dif_density_func(cur_score_dif) *
      confidence_predict(odds,score_difs,cur_score_dif,rounding_places)
    cur_score_dif <- round(cur_score_dif - step_size,rounding_places)
  }
  return(result)
}

# Uses above with a list of scores of teams we're playing against
# combined with a score_func for our team to calculate the odds
# of never losing a game and average won games - first entry
# is average, second is undefeated odds

success_measures <- function(odds, score_difs, score_func, score_cdf,
                             team_scores, rounding_places, low, high) {
  result <- c(0,1)
  win_chances <- c()
  for(score in team_scores) {
    win_chances <- append(win_chances,distribution_win_chance(
      odds, score_difs, function(x){score_func(x + score)},
      function(x){score_cdf(x + score)}, rounding_places,
      low - score, high - score
    ))
  }
  for(chance in win_chances) {
    result[1] <- result[1] + chance
    result[2] <- result[2] * chance
  }
  return(result)
}

# Assuming a normal distribution, and that the overall game performance
# is the average of the players' game performance, we can make a
# helper function to calculate the score distribution for the team

# The normal assumption is reasonable - see the analysis sheet,
# there's a helpful plot on it

score_pdf_from_players <- function(player_means, player_sds) {
  output_func <- function(x){dnorm(x,mean = mean(player_means),
                                   sd = sqrt(sum(player_sds^2)))}
  return(output_func)
}

score_cdf_from_players <- function(player_means, player_sds) {
  output_func <- function(x){pnorm(x,mean = mean(player_means),
                                   sd = sqrt(sum(player_sds^2)),
                                   lower.tail = FALSE)}
  return(output_func)
}

```



The R session information (including the OS info, R version and all packages used):

```
sessionInfo()
```

```
## R version 4.2.2 (2022-10-31)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Monterey 12.6
##
## Matrix products: default
## LAPACK: /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats      graphics  grDevices  utils      datasets  methods   base
##
## other attached packages:
## [1] tinytex_0.43    MASS_7.3-58.1  rstudioapi_0.14 haven_2.5.1    lubridate_1.9.2
## [6] forcats_1.0.0   stringr_1.5.0  dplyr_1.1.0   purrr_1.0.1    readr_2.1.4
## [11] tidyr_1.3.0     tibble_3.1.8   ggplot2_3.4.1  tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
## [1] highr_0.10      pillar_1.8.1    compiler_4.2.2  tools_4.2.2     bit_4.0.5
## [6] evaluate_0.19   lattice_0.20-45 nlme_3.1-160    lifecycle_1.0.3 gtable_0.3.1
## [11] timechange_0.2.0 mgcv_1.8-41     pkgconfig_2.0.3 rlang_1.0.6     Matrix_1.5-1
## [16] cli_3.6.0       parallel_4.2.2  xfun_0.36       knitr_1.41      withr_2.5.0
## [21] generics_0.1.3  vctrs_0.5.2     hms_1.1.2       bit64_4.0.5     grid_4.2.2
## [26] tidyselect_1.2.0 glue_1.6.2      R6_2.5.1        fansi_1.0.3     vroom_1.6.0
## [31] farver_2.1.1    tzdb_0.3.0      magrittr_2.0.3  splines_4.2.2   scales_1.2.1
## [36] ellipsis_0.3.2  colorspace_2.0-3 labeling_0.4.2   utf8_1.2.2      stringi_1.7.12
## [41] munsell_0.5.0   crayon_1.5.2
```

```
Sys.time()
```

```
## [1] "2023-03-20 00:08:00 PDT"
```

```
##### RUN THIS CODE BEFORE DOING ANYTHING ELSE #####

# MAKE SURE YOU HAVE ALL CSVs DOWNLOADED AND IN YOUR WORKING DIRECTORY

# Throughout this, there are varying difficulties of tasks marked TODO.
# They represent places where my work has been left at a place where there
# is a natural continuation of my work. Feel free to continue or start anew
# however you see fit, just the TODO activities give leads for stuck moments.

## Dependencies
library(tidyverse)
library(haven)

# Adjust wd, verify
file_path <- rstudioapi::getSourceEditorContext()$path
setwd(dirname(file_path))

source('../SCRIPTS/helper_functions.R') # Import helpers

## Open data and make their storage variables
games <- list(read_csv("../RAW_DATA/game_1_data.csv"),
              read_csv("../RAW_DATA/game_2_data.csv"),
              read_csv("../RAW_DATA/game_3_data.csv"),
              read_csv("../RAW_DATA/game_4_data.csv"),
              read_csv("../RAW_DATA/game_5_data.csv"),
              read_csv("../RAW_DATA/game_6_data.csv"),
              read_csv("../RAW_DATA/game_7_data.csv"),
              read_csv("../RAW_DATA/game_8_data.csv"),
              read_csv("../RAW_DATA/game_9_data.csv"),
              read_csv("../RAW_DATA/game_10_data.csv"))

## Rows: 1000 Columns: 12
## — Column specification —————
## Delimiter: ",",
## chr (5): coast, game scheduled, early bird/night owl, previous competitive sports, uni...
## dbl (7): student label, game score, percent training sessions attended, overall fitness...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 1000 Columns: 12
## — Column specification —————
## Delimiter: ",",
## chr (5): coast, game scheduled, early bird/night owl, previous competitive sports, uni...
## dbl (7): student label, game score, percent training sessions attended, overall fitness...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 1000 Columns: 12
## — Column specification —————
## Delimiter: ",",
## chr (5): coast, game scheduled, early bird/night owl, previous competitive sports, uni...
## dbl (7): student label, game score, percent training sessions attended, overall fitness...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 1000 Columns: 12
## — Column specification —————
## Delimiter: ",",
## chr (5): coast, game scheduled, early bird/night owl, previous competitive sports, uni...
## dbl (7): student label, game score, percent training sessions attended, overall fitness...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 1000 Columns: 12
## — Column specification —————
## Delimiter: ",",
## chr (5): coast, game scheduled, early bird/night owl, previous competitive sports, uni...
## dbl (7): student label, game score, percent training sessions attended, overall fitness...
##
```

```
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 1000 Columns: 12
## — Column specification —————
## Delimiter: ",",
## chr (5): coast, game scheduled, early bird/night owl, previous competitive sports, uni...
## dbl (7): student label, game score, percent training sessions attended, overall fitnes...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 1000 Columns: 12
## — Column specification —————
## Delimiter: ",",
## chr (5): coast, game scheduled, early bird/night owl, previous competitive sports, uni...
## dbl (7): student label, game score, percent training sessions attended, overall fitnes...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 1000 Columns: 12
## — Column specification —————
## Delimiter: ",",
## chr (5): coast, game scheduled, early bird/night owl, previous competitive sports, uni...
## dbl (7): student label, game score, percent training sessions attended, overall fitnes...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 1000 Columns: 12
## — Column specification —————
## Delimiter: ",",
## chr (5): coast, game scheduled, early bird/night owl, previous competitive sports, uni...
## dbl (7): student label, game score, percent training sessions attended, overall fitnes...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
## Rows: 1000 Columns: 12
## — Column specification —————
## Delimiter: ",",
## chr (5): coast, game scheduled, early bird/night owl, previous competitive sports, uni...
## dbl (7): student label, game score, percent training sessions attended, overall fitnes...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# Reads the games list into one big dataframe
# with an extra category indicating the game number
all_games <- games[[1]] %>%
  mutate(game = 1)
for (index in 2:10) {
  all_games = rbind(all_games, games[[index]] %>% mutate(game = index))
}

# Converts some of the categoricals in the just-made dataframe into
# indicator functions. The way I'm doing grades is clunky but I
# couldn't find a better way

# Identify all unique sports
previous_sports_col <- distinct(select(all_games, 'previous competitive sports'))
```

```
## Error in select(all_games, "previous competitive sports"): unused argument ("previous competitive sports")
```

```
unique_sports_list <- as.list(previous_sports_col)[[1]]
```

```
## Error in as.list(previous_sports_col): object 'previous_sports_col' not found
```

```
# Generate indicator columns
columns <- c()

for(i in seq(length(unique_sports_list))){
  sport <- unique_sports_list[[i]]
```

```

print(sport)
col <- match_sport(all_games$'previous competitive sports', sport)
columns[[i]] <- col
}

```

```
## Error in seq(length(unique_sports_list)): object 'unique_sports_list' not found
```

```

# Add all sports indicator columns
all_games <- all_games %>%
  mutate(curling = columns[[1]], gymnastics = columns[[2]], baseball = columns[[3]],
    martial_arts = columns[[4]], frisbee = columns[[5]], table_tennis = columns[[6]],
    basketball = columns[[7]], football = columns[[8]])

all_games <- all_games %>%
  mutate(evening = as.integer(`game scheduled` == "evening"),
    morning = as.integer(`game scheduled` == "morning"),
    `west coast` = as.integer(coast == "west coast"),
    `university year` = ifelse(`university year`=="frosh",1,
      ifelse(`university year`=="sophomore",2,
        ifelse(`university year`=="junior",3,
          ifelse(`university year`=="senior",4,NA)))),
    night_owl = as.integer(`early bird/night owl` == "night owl"))

#mutate(`winning team` = as_factor(`winning team`)) %>% # Previously used but breaks averages

# NICK CODE: Write to CSV
write.csv(all_games, "../PROCESSED_DATA/processed_all_games.csv", row.names = FALSE)

# Creates a dataframe that has information in one row for each player
# TODO: Fix evening score and morning score (they are currently broken)
# TODO: Add variances on each covariate? Could be good for confidence intervals
# TODO: Add indicator function for previous sports once added to all_games
player_means <- distinct(all_games %>%
  group_by(`student label`) %>%
  summarize(`game score` = mean(`game score`),
    `evening score` = sum(`game score` * evening)/sum(evening),
    `morning score` = sum(morning * `game score`)/sum(morning),
    `percent training sessions attended` = mean(`percent training sessions attended`),
    `overall fitness score` = mean(`overall fitness score`),
    `# extra strategy sessions attended` = mean(`# extra strategy sessions attended`),
    `hours of sleep the night before game` = mean(`hours of sleep the night before game`),
    `previous competitive sports` = Mode(`previous competitive sports`),
    `# meals on day prior to game` = mean(`# meals on day prior to game`),
    `university year` = Mode(`university year`),
    evening = mean(evening),
    morning = mean(morning),
    night_owl = Mode(night_owl)))

# Normalized version of all_games, except for indicator function variables
# Useful for the 'how best to coach a player' portion, we can compare
# beta values directly with this.
normalized_games <- all_games %>%
  mutate(`percent training sessions attended` = normalize(`percent training sessions attended`),
    `overall fitness score` = normalize(`overall fitness score`),
    `# extra strategy sessions attended` = normalize(`# extra strategy sessions attended`),
    `hours of sleep the night before game` = normalize(`hours of sleep the night before game`),
    `# meals on day prior to game` = normalize(`# meals on day prior to game`),
    `university year` = normalize(`university year`))

match_ups <- read_csv("../RAW_DATA/season_match_up.csv")

## Rows: 1 Columns: 21
## — Column specification —————
## Delimiter: ","
## chr (1): team label
## dbl (20): 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

```

```
previous_results <- read_csv("../RAW_DATA/previous_season_results.csv")

## Rows: 200 Columns: 4
## — Column specification —————
## Delimiter: ","
## dbl (4): game label, team 1 score, team 2 score, winning team
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

write_csv(all_games, "../PROCESSED_DATA/processed_normalized_all_games.csv", row.names = FALSE)

# Add a score difference variable
previous_results <- previous_results %>%
  mutate(score_dif = `team 1 score` - `team 2 score`)

write_csv(previous_results, "../PROCESSED_DATA/processed_previous_results.csv", row.names = FALSE)
```

The R session information (including the OS info, R version and all packages used):

```
sessionInfo()
```

```
## R version 4.2.2 (2022-10-31)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Monterey 12.6
##
## Matrix products: default
## LAPACK: /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats      graphics  grDevices utils      datasets  methods   base
##
## other attached packages:
## [1] tinytex_0.43      MASS_7.3-58.1    rstudioapi_0.14  haven_2.5.1      lubridate_1.9.2
## [6] forcats_1.0.0     stringr_1.5.0    dplyr_1.1.0      purrr_1.0.1      readr_2.1.4
## [11] tidyr_1.3.0       tibble_3.1.8     ggplot2_3.4.1    tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
## [1] highr_0.10        pillar_1.8.1     compiler_4.2.2   tools_4.2.2      bit_4.0.5
## [6] evaluate_0.19     lattice_0.20-45  nlme_3.1-160     lifecycle_1.0.3  gtable_0.3.1
## [11] timechange_0.2.0  mgcv_1.8-41      pkgconfig_2.0.3  rlang_1.0.6      Matrix_1.5-1
## [16] cli_3.6.0         parallel_4.2.2   xfun_0.36        knitr_1.41       withr_2.5.0
## [21] generics_0.1.3    vctrs_0.5.2      hms_1.1.2        bit64_4.0.5      grid_4.2.2
## [26] tidyselect_1.2.0  glue_1.6.2       R6_2.5.1         fansi_1.0.3      vroom_1.6.0
## [31] farver_2.1.1      tzdb_0.3.0       magrittr_2.0.3    splines_4.2.2    scales_1.2.1
## [36] ellipsis_0.3.2    colorspace_2.0-3 labeling_0.4.2    utf8_1.2.2       stringi_1.7.12
## [41] munsell_0.5.0     crayon_1.5.2
```

```
Sys.time()
```

```
## [1] "2023-03-20 00:09:42 PDT"
```

This report is automatically generated with the R package [knitr](#) (version 1.41) .

```
# Install packages (uncomment and run when needed)
# install.packages('tidyverse')
# install.packages('haven')

# Libraries and helpers
library(tidyverse)
library(haven)
library(rstudioapi)

# Adjust wd
file_path <- rstudioapi::getSourceEditorContext()$path
setwd(dirname(file_path))

# Import helpers
source("helper_functions.R")

# Import processed data
previous_results <- read_csv("../PROCESSED_DATA/processed_previous_results.csv")
```

```
## Rows: 200 Columns: 5
## — Column specification —————
## Delimiter: ",",
## dbl (5): game label, team 1 score, team 2 score, winning team, score_dif
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
match_ups <- read_csv("../RAW_DATA/season_match_up.csv")
```

```
## Rows: 1 Columns: 21
## — Column specification —————
## Delimiter: ",",
## chr (1): team label
## dbl (20): 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
match_ups <- match_ups[1,2:20]
```

```
##### PREVIOUS RESULTS ANALYSIS #####
```

```
## Understanding the distribution of previous results
length(previous_results$score_dif) # n = 200
```

```
## [1] 200
```

```
summary(previous_results$score_dif)
```

```
##      Min.   1st Qu.   Median     Mean   3rd Qu.     Max.
## -82.1400 -14.8375   1.1750  -0.1155  16.5375  53.2700
```

```
ggplot(previous_results, aes(x=score_dif)) +
  geom_histogram(binwidth=5, color="grey50", fill="lightblue", alpha=0.8) +
  theme_minimal() +
  stat_function(fun = function(x){dnorm(x,mean = mean(previous_results$score_dif)
                                         ,sd = sd(previous_results$score_dif))}*length(previous_results$score_dif))
```

 plot of chunk auto-report

```
# Looks approximately normally distributed, with a mean of:
```

```
mean(previous_results$score_dif)
```

```
## [1] -0.11555
```

```
#-0.116, and standard deviation of:  
score_sd = sd(previous_results$score_dif)  
# 23.328, for a standard error on the mean of:  
sd(previous_results$score_dif)/length(previous_results$score_dif)
```

```
## [1] 0.116638
```

```
# 0.116. So the mean is not significantly different from 0,  
# and it's approximately normally distributed with SD = 23.328  
# This will be useful later.
```

```
previous_results_means <- previous_results %>%  
  mutate(bucket = floor(score_dif/5) + 0.5) %>%  
  group_by(bucket) %>%  
  summarize(  
    n = sum(!is.na(`winning team`)),  
    mean = mean(`winning team`)  
  )
```

```
# From these, we can see that the bulk of the data is between  
# -35 and +35 on the score_dif axis. This will be useful later.
```

```
# This one shows the winrate in the buckets as a function of score difference:  
# (buckets of 5)  
# TODO: (fast) Rename the axis on these  
ggplot(previous_results_means, aes(x=bucket*5, y=mean, fill=bucket)) +  
  geom_bar(stat="identity") +  
  labs(y="Chance of winning", x="Score difference") +  
  geom_text(aes(label=round(mean, digits=3)), color="green", vjust=-2.5) +  
  theme_minimal()
```

plot of chunk auto-report

```
# We can see from this that for score_dif < -35, we lose (almost) every game -  
# there would be a confidence interval on that which makes for a small chance,  
# but anyway - and for score_dif > 35, we win (almost) any game
```

```
# Thus, we can model this function piecewise. Below some cutoff, we always lose,  
# and above another, we always win. Between, we can model it as linear.
```

```
# Notably, this function should be symmetric about score_dif = 0, mean = 0.5  
# due to the win chance at score_dif=x being equal to the lose chance at  
# score_dif = -x (in a perfect world)
```

```
# Also notably, this function should be continuous (or at least close) -  
# this means that we expect the function's confidence interval to contain  
# (-35,0) and (35,1), or whatever our cutoffs are.
```

```
# This one limits it to those bounds, and makes a pretty reasonable trendline  
# It makes an error message but the issue it says is happening isn't actually  
# happening
```

```
ggplot(filter(previous_results_means, abs(bucket*5) <= 35), aes(x=bucket*5, y=mean, fill=bucket)) +  
  geom_bar(stat="identity") +  
  geom_text(aes(label=round(mean, digits=3)), color="green", vjust=-2.5) +  
  stat_smooth(method="lm") +  
  theme_minimal()
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

```
## Warning: The following aesthetics were dropped during statistical transformation: fill  
## i This can happen when ggplot fails to infer the correct grouping structure in the data.  
## i Did you forget to specify a `group` aesthetic or to convert a numerical variable into a  
## factor?
```

plot of chunk auto-report



```
# But we can do better than that! Lets do some regressions

### Regression stuff with helper functions


# Initially, I did this with vertical distance squared. The code
# below is left as a record of that, but is not actually helpful for
# regression - it turns out vertical distance squared has a strong
# bias towards underestimating the cutoff

## With vertical distance squared
piecewise_residual_square(previous_results$`winning team`,
                          previous_results$score_dif, 35)
```

```
## [1] 27.10651
```

```
residual_by_cutoff <- multiple_piecewise_residual(previous_results$`winning team`,
                                                  previous_results$score_dif, 2000, piecewise_residual_square)

# We can minimize this
ggplot(residual_by_cutoff, aes(x=cutoffs, y=residuals)) +
  geom_point() +
  theme_minimal()
```

 plot of chunk auto-report


```
cutoff <- mean(residual_by_cutoff$cutoffs[which(residual_by_cutoff$residuals == min(residual_by_cutoff$residuals))])

# Now we know the region to search in
targeted_range <- targeted_piecewise_residual(previous_results$`winning team`,
                                              previous_results$score_dif, 50000, cutoff - 5, cutoff + 5,
                                              piecewise_residual_square)

# Accurate to three decimal places - because we searched
# a range of length 10 in 50,000 intervals
cutoff <- round(mean(targeted_range$cutoffs[which(targeted_range$residuals == min(targeted_range$residuals))]))

a <- read.csv("../PROCESSED_DATA/cutoff_distribution_demo.csv")$a
# Proof-of-concept simulation:
# WARNING: TAKES 1 MINUTE TO RUN
# a <- append(a,cutoff_distribution(cutoff, 1000, 200, score_sd, piecewise_residual_square))

# This is the plot that makes the bias clear. The true cutoff is shown in
# red, and as you can see, the data are significantly and systematically
# skewed from it. This means we can't trust the results of this regression
# without significant adjustments, so we might as well regress a different
# way
ggplot(data.frame(a), aes(x=a)) +
  geom_histogram(binwidth=0.5, color="grey50", fill="lightblue", alpha=0.8) +
  theme_minimal() +
  stat_function(fun = function(x){dnorm(x,mean = mean(a),sd = sd(a))}*length(a)*0.5) +
  geom_vline(xintercept = cutoff, color = "red")
```

 plot of chunk auto-report

```
# write.csv(data.frame(a), "../PROCESSED_DATA/cutoff_distribution_demo.csv")

# The new objective function is the likelihood of the observed data
# occurring in the theorized model. All of these chances will be small,
# but some will be much smaller than others.

## With -1 times probability of occurring
# The -1 is so that we can minimize this, instead of maximizing -
# it lets us use the same functions as vertical distance squared
piecewise_probability(previous_results$`winning team`,
                     previous_results$score_dif, 35)
```

```
## [1] -8.116819e-37
```



```
probs_by_cutoff <- multiple_piecewise_residual(previous_results$`winning team`,
                                              previous_results$score_dif, 2000, piecewise_probability)

ggplot(probs_by_cutoff, aes(x=cutoffs, y=-1*residuals)) +
  geom_point() +
  theme_minimal() +
  labs(y="MLE", x="Cutoff")
```

 plot of chunk auto-report

```
probs_by_cutoff$residuals <- probs_by_cutoff$residuals/sum(probs_by_cutoff$cutoffs[2] * probs_by_cutoff$residuals)

probs_by_cutoff <- probs_by_cutoff %>% mutate(
  cumprob = cumsum(residuals)*probs_by_cutoff$cutoffs[2]
)

percentile_2.5 <- probs_by_cutoff$cutoff[probs_by_cutoff$cumsum>0.025][1]
percentile_97.5 <- probs_by_cutoff$cutoff[probs_by_cutoff$cumsum>0.975][1]

# Plot with lines denoting the 95% confidence interval for the cutoff
# From this plot, you can already see the expected cutoff and how
# we expect the possible true cutoffs to be distributed given these
# data.
ggplot(probs_by_cutoff, aes(x=cutoffs, y=residuals)) +
  geom_point() +
  theme_minimal() +
  labs(y="Probability density (given observed results)", x="Cutoff") +
  geom_vline(xintercept = percentile_2.5, color = "red") +
  geom_vline(xintercept = percentile_97.5, color = "red")
```

**## Warning: Removed 1 rows containing missing values (`geom\_vline()`).**

**## Warning: Removed 1 rows containing missing values (`geom\_vline()`).**

 plot of chunk auto-report

```
cutoff <- mean(residual_by_cutoff$cutoffs[which(residual_by_cutoff$residuals == min(residual_by_cutoff$residuals))])

# Now we know the region to search in
targeted_range <- targeted_piecewise_residual(previous_results$`winning team`,
                                              previous_results$score_dif, 50000, cutoff - 5, cutoff + 5,
                                              piecewise_probability)

# Accurate to three decimal places - because we searched
# a range of length 10 in 50,000 intervals
cutoff <- round(mean(targeted_range$cutoffs[which(targeted_range$residuals == min(targeted_range$residuals))]), 3)

p_distribution <- read.csv("../PROCESSED_DATA/p_cutoff_distribution_demo.csv")
# Proof-of-concept simulation:
# WARNING: TAKES 5 MINUTES TO RUN (if you uncomment it)
# p_distribution <- append(p_distribution, cutoff_distribution(cutoff, 10000, 200, score_sd, piecewise_probability))

ggplot(data.frame(p_distribution), aes(x=p_distribution)) +
  geom_histogram(binwidth=0.5, color="grey50", fill="lightblue", alpha=0.8) +
  theme_minimal() +
  stat_function(fun = function(x){dnorm(x, mean = mean(p_distribution$p_distribution),
                                       sd = sd(p_distribution$p_distribution))*length(p_distribution$p_distribution)}, aes(x=p_distribution)) +
  geom_vline(xintercept = cutoff, color="red")
```

 plot of chunk auto-report

```
# Plot shows a little bias on the mean, but none on the mode -
# seems acceptable

# write.csv(data.frame(p_distribution), "../PROCESSED_DATA/p_cutoff_distribution_demo.csv")

# Now we want probabilities of winning at different score
# differences, given this distribution of cutoffs - the mean chance
# of winning, given that we observed this distribution.

# We'll round to 2 decimal places and evaluate integrals with
```

```
# subintervals of 0.01, because those are the gradations on the
# score data we have
rounding_places <- 2

## Takes ~3 minutes to run (if you uncomment it)

# This code just runs all the computations. We stored it so
# we don't have to recalculate it every time.
# It just goes through and for a given score difference, adds up
# the products of winning under a given model times our confidence
# of that model being true under these observations.

# chance_winning <- c()
# score_difs <- c()
# score_step_size <- 10^{-1*rounding_places}
# current_score_dif <- min(previous_results$score_dif)
# no_zeros <- filter(probs_by_cutoff,residuals>0)
# for(j in 1:ceiling((max(previous_results$score_dif)
# - min(previous_results$score_dif))/score_step_size)) {
#   score_difs <- append(score_difs,round(current_score_dif, rounding_places))
#   current_sum <- 0
#   for(i in 1:nrow(no_zeros)) {
#     current_sum <- current_sum + no_zeros[i,2] * piecewise_predict(
#       current_score_dif, no_zeros[i,1])
#   }
#   chance_winning <- append(chance_winning,current_sum * probs_by_cutoff$cutoffs[2])
#   current_score_dif <- current_score_dif + score_step_size
# }
# win_chances <- data.frame(score_difs,chance_winning)
# write_csv(win_chances,"../PROCESSED_DATA/confidence_win_rates.csv")

win_chances <- read_csv("../PROCESSED_DATA/confidence_win_rates.csv")
```

```
## Rows: 13541 Columns: 2
## — Column specification —————
## Delimiter: ","
## dbl (2): score_difs, chance_winning
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# This is what our probability curve looks like after accounting
# for our confidence distribution in true cutoffs
ggplot(win_chances, aes(x=score_difs, y=chance_winning)) +
  geom_point() +
  theme_minimal() +
  labs(y="Chance of winning (given observed dist.)", x="Score difference")
```

 plot of chunk auto-report

```
# Example use of success_measures
success_measures(win_chances$chance_winning,win_chances$score_difs,
  function(x){dnorm(x,mean=50,sd=15)},
  function(x){pnorm(x,mean=50,sd=15,lower.tail = FALSE)},
  match_ups,2,5,95)
```

```
## [1] 3.926551e+00 2.024547e-17
```

The R session information (including the OS info, R version and all packages used):

```
sessionInfo()
```

```
## R version 4.2.2 (2022-10-31)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Monterey 12.6
##
## Matrix products: default
## LAPACK: /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRlapack.dylib
##
```

```
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats      graphics  grDevices  utils      datasets  methods   base
##
## other attached packages:
## [1] tinytex_0.43    MASS_7.3-58.1  rstudioapi_0.14 haven_2.5.1    lubridate_1.9.2
## [6] forcats_1.0.0   stringr_1.5.0  dplyr_1.1.0   purrr_1.0.1    readr_2.1.4
## [11] tidyr_1.3.0     tibble_3.1.8   ggplot2_3.4.1 tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
## [1] highr_0.10      pillar_1.8.1    compiler_4.2.2  tools_4.2.2     bit_4.0.5
## [6] evaluate_0.19   lattice_0.20-45 nlme_3.1-160    lifecycle_1.0.3 gtable_0.3.1
## [11] timechange_0.2.0 mgcv_1.8-41     pkgconfig_2.0.3 rlang_1.0.6      Matrix_1.5-1
## [16] cli_3.6.0       parallel_4.2.2  xfun_0.36       knitr_1.41       withr_2.5.0
## [21] generics_0.1.3  vctrs_0.5.2     hms_1.1.2       bit64_4.0.5      grid_4.2.2
## [26] tidyselect_1.2.0 glue_1.6.2      R6_2.5.1        fansi_1.0.3      vroom_1.6.0
## [31] farver_2.1.1    tzdb_0.3.0      magrittr_2.0.3  splines_4.2.2    scales_1.2.1
## [36] ellipsis_0.3.2  colorspace_2.0-3 labeling_0.4.2   utf8_1.2.2       stringi_1.7.12
## [41] munsell_0.5.0   crayon_1.5.2
```

```
Sys.time()
```

```
## [1] "2023-03-20 00:09:07 PDT"
```

This report is automatically generated with the R package [knitr](#) (version 1.41) .

```
# Install packages (uncomment and run when needed)
# install.packages('tidyverse')
# install.packages('haven')

# Libraries and helpers
library(tidyverse)
library(haven)
library(rstudioapi)
library(MASS)

# Adjust wd
file_path <- rstudioapi::getSourceEditorContext()$path
setwd(dirname(file_path))

# Import helpers
source("helper_functions.R")

# Import processed data
normalized_games <- read_csv("../PROCESSED_DATA/processed_normalized_all_games.csv")
```

```
## Rows: 10000 Columns: 17
## — Column specification —————
## Delimiter: ",",
## chr (4): coast, game scheduled, early bird/night owl, previous competitive sports
## dbl (13): student label, game score, percent training sessions attended, overall fitne...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
all_games <- read_csv("../PROCESSED_DATA/processed_all_games.csv")
```

```
## Rows: 10000 Columns: 25
## — Column specification —————
## Delimiter: ",",
## chr (4): coast, game scheduled, early bird/night owl, previous competitive sports
## dbl (21): student label, game score, percent training sessions attended, overall fitne...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
previous_results <- read_csv("../PROCESSED_DATA/processed_previous_results.csv")
```

```
## Rows: 200 Columns: 5
## — Column specification —————
## Delimiter: ",",
## dbl (5): game label, team 1 score, team 2 score, winning team, score_dif
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
match_ups <- read_csv("../RAW_DATA/season_match_up.csv")
```

```
## Rows: 1 Columns: 21
## — Column specification —————
## Delimiter: ",",
## chr (1): team label
## dbl (20): 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
match_ups <- match_ups[1,2:20]
```

```
## Preliminary transformations and regressions

# Do regressions on all relevant covariates and add
# the predicted values and residuals into the dataframes

# Because night_owl interacts with the evening and morning variables,
# we need to add extra terms to account for the cross-variable
# interaction - this makes sense, because being a night owl affects
# your performance differently based on the time of day
overallmodel <- lm(`game score` ~ `percent training sessions attended`
  + `overall fitness score`
  + `# extra strategy sessions attended`
  + `hours of sleep the night before game`
  + `# meals on day prior to game`
  + `university year`
  + `curling` + `gymnastics` + `baseball` + `martial_arts`
  + `frisbee` + `table_tennis` + `basketball`
  + evening + morning + night_owl
  + night_owl * morning + night_owl * evening,
  data=all_games)
summary(overallmodel) # Run lines like this again to see the model
```

```
##
## Call:
## lm(formula = `game score` ~ `percent training sessions attended` +
##   `overall fitness score` + `# extra strategy sessions attended` +
##   `hours of sleep the night before game` + `# meals on day prior to game` +
##   `university year` + curling + gymnastics + baseball + martial_arts +
##   frisbee + table_tennis + basketball + evening + morning +
##   night_owl + night_owl * morning + night_owl * evening, data = all_games)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -34.493  -7.032  -0.371   6.625 101.483
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)      2.53884    0.75278   3.373 0.000747 ***
## `percent training sessions attended` 26.80310    0.32129  83.423 < 2e-16 ***
## `overall fitness score`      22.54530    0.32902  68.522 < 2e-16 ***
## `# extra strategy sessions attended`   1.54357    0.04119  37.470 < 2e-16 ***
## `hours of sleep the night before game` 0.33426    0.03299  10.132 < 2e-16 ***
## `# meals on day prior to game`    -0.84742    0.09338  -9.075 < 2e-16 ***
## `university year`           0.03513    0.13357   0.263 0.792527
## curling                -0.59852    0.42401  -1.412 0.158113
## gymnastics             -2.09892    0.55520  -3.780 0.000157 ***
## baseball              -1.24266    0.45112  -2.755 0.005886 **
## martial_arts           4.94071    0.44492  11.105 < 2e-16 ***
## frisbee                5.47228    0.45988  11.899 < 2e-16 ***
## table_tennis          -0.70959    0.43318  -1.638 0.101434
## basketball             5.06037    0.53900   9.388 < 2e-16 ***
## evening               -0.38195    0.68511  -0.558 0.577195
## morning                7.71937    0.92515   8.344 < 2e-16 ***
## night_owl              0.94990    0.36917   2.573 0.010095 *
## morning:night_owl     -8.20912    1.02030  -8.046 9.54e-16 ***
## evening:night_owl      7.28970    0.75183   9.696 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.45 on 9981 degrees of freedom
## Multiple R-squared:  0.6814, Adjusted R-squared:  0.6808
## F-statistic: 1186 on 18 and 9981 DF, p-value: < 2.2e-16
```

```
all_games <- subset(cbind(all_games, predict(overallmodel, se.fit = TRUE))) %>%
  mutate(residual = `game score` - `fit`), select = -c(`df`, `residual.scale`))

normalizedmodel <- lm(`game score` ~ `percent training sessions attended`
  + `overall fitness score`
  + `# extra strategy sessions attended`
  + `hours of sleep the night before game`
  + `# meals on day prior to game`
  + `university year`
  + evening + morning + night_owl,
```

```

data=normalized_games)

summary(normalizedmodel)

##
## Call:
## lm(formula = `game score` ~ `percent training sessions attended` +
##   `overall fitness score` + `# extra strategy sessions attended` +
##   `hours of sleep the night before game` + `# meals on day prior to game` +
##   `university year` + evening + morning + night_owl, data = normalized_games)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -36.951  -7.426  -0.168   7.060 100.400
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)      5.73175    0.65774   8.714 < 2e-16 ***
## `percent training sessions attended` 27.27642    0.33164  82.247 < 2e-16 ***
## `overall fitness score`      24.76341    0.32356  76.533 < 2e-16 ***
## `# extra strategy sessions attended`  1.41283    0.04220  33.483 < 2e-16 ***
## `hours of sleep the night before game` 0.31872    0.03411   9.344 < 2e-16 ***
## `# meals on day prior to game`      -1.36861    0.09353 -14.633 < 2e-16 ***
## `university year`           -0.43773    0.13536  -3.234  0.00123 **
## evening              5.72868    0.29238  19.593 < 2e-16 ***
## morning              0.99846    0.40341   2.475  0.01334 *
## night_owl            1.77506    0.31569   5.623 1.93e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.86 on 9990 degrees of freedom
## Multiple R-squared:  0.658, Adjusted R-squared:  0.6577
## F-statistic: 2136 on 9 and 9990 DF, p-value: < 2.2e-16

```

```

normalized_games <- subset(cbind(normalized_games, predict(normalizedmodel, se.fit = TRUE)) %>%
  mutate(residual = `game score` - fit), select = -c(df, residual.scale))

```

```

#####
# OUTLIER DETECTION

```

```

# Begin by generating frequency distributions for all covariates in order to identify
# high-leverage points.

```

```

multi_histogram(all_games, cols = c("hours of sleep the night before game",
  "# meals on day prior to game",
  "overall fitness score"))

```

```

#####
# TRANSFORMATIONS AND RESIDUALS

```

```

# We want to plot the residuals of our model's game score predictions against the
# predictions themselves. Let's extract this from all_games and create a scatterplot.

```

```

png("../OUTPUTS/initial_residuals.png")
all_games_residuals <- dplyr::select(all_games, `fit`, `se.fit`, `residual`)
all_games_residuals <- all_games_residuals %>% mutate(standard_residuals = rstandard(overallmodel))
ggplot(all_games_residuals, aes(x = fit, y = standard_residuals)) +
  geom_point(color = 'red')
dev.off()

```

```


## RStudioGD
##      2

```

```

##### ARCSIN TRANSFORM #####
arcsin_games <- all_games

```

 plot of chunk unnamed-chunk-1

```

arcsin_games <- arcsin_games %>% mutate(arcsin_scores = asinh(`game score`))

arcsinmodel <- lm(`arcsin_scores` ~ `percent training sessions attended`

```

```
+ `overall fitness score`
+ `# extra strategy sessions attended`
+ `hours of sleep the night before game`
+ `# meals on day prior to game`
+ `university year`
+ evening + morning + night_owl
+ night_owl * morning + night_owl * evening,
data=arcsin_games)
```

```
summary(arcsinmodel)
```

```
##
## Call:
## lm(formula = arcsin_scores ~ `percent training sessions attended` +
##     `overall fitness score` + `# extra strategy sessions attended` +
##     `hours of sleep the night before game` + `# meals on day prior to game` +
##     `university year` + evening + morning + night_owl + night_owl *
##     morning + night_owl * evening, data = arcsin_games)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -3.9822 -0.1591  0.0911  0.3622  2.9747
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)      2.796848   0.047768  58.551 < 2e-16 ***
## `percent training sessions attended`  1.137294   0.023099  49.235 < 2e-16 ***
## `overall fitness score`      0.897730   0.022520  39.863 < 2e-16 ***
## `# extra strategy sessions attended`  0.066627   0.002939  22.668 < 2e-16 ***
## `hours of sleep the night before game`  0.024717   0.002375  10.409 < 2e-16 ***
## `# meals on day prior to game`    -0.080817   0.006527  -12.382 < 2e-16 ***
## `university year`      -0.021636   0.009478   -2.283  0.0225 *
## evening      -0.001369   0.049401   -0.028  0.9779
## morning       0.360140   0.066692   5.400 6.82e-08 ***
## night_owl     0.006438   0.026607   0.242  0.8088
## morning:night_owl    -0.381085   0.073543  -5.182 2.24e-07 ***
## evening:night_owl    0.325032   0.054215   5.995 2.10e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8259 on 9988 degrees of freedom
## Multiple R-squared:  0.4077, Adjusted R-squared:  0.4071
## F-statistic: 625 on 11 and 9988 DF, p-value: < 2.2e-16
```

```
arcsin_games <- subset(arcsin_games, select = -c(`fit`, `residual`, `se.fit`))
arcsin_games <- subset(cbind(arcsin_games, predict(arcsinmodel, se.fit = TRUE))) %>%
  mutate(standard_residual = `arcsin_scores` - `fit`), select = -c(`df`, `residual.scale`))
```

```
arcsin_games$`residual` <- rstandard(arcsinmodel)
```

```
png("../OUTPUTS/arcsin_residuals.png")
```

```
ggplot(arcsin_games, aes(x = fit, y = standard_residual)) +
  geom_point(aes(color = night_owl)) +
  ggtitle("Residuals of arcsin(score)") +
  theme(plot.title = element_text(hjust = 0.5, vjust = 0.5))
```

```
dev.off()
```

```
## RStudioGD
##      2
```


```
##### BOXCOX #####
boxcox_all_games <- all_games
```

```
# Ensure that all responses (scores) are above zero
boxcox_all_games$`game score` <- boxcox_all_games$`game score` + 0.001
```

```
# boxcox() requires a linear model arg lm(), for some reason.
boxcox_overall_model <- lm(`game score` ~ `percent training sessions attended`
+ `overall fitness score`
```

```
+ `# extra strategy sessions attended`
+ `hours of sleep the night before game`
+ `# meals on day prior to game`
+ `university year`
+ `curling` + `gymnastics` + `baseball` + `martial_arts`
+ `frisbee` + `table_tennis` + `basketball`
+ evening + morning + night_owl
+ night_owl * morning + night_owl * evening,
data=boxcox_all_games)
```

```
# Find the optimal lambda
bc <- boxcox(boxcox_overall_model)
```

 plot of chunk unnamed-chunk-1

```
lambda <- bc[[1]][which.max(bc[[2]])] # This is the lambda we'll use for Box-Cox

# Transform the scores column
boxcox_all_games$`game score` <- ((boxcox_all_games$`game score`)^(lambda) - 1)/lambda

# Run regression with transformed scores
fin_boxcox_overall_model <- lm(`game score` ~ `percent training sessions attended`
+ `overall fitness score`
+ `# extra strategy sessions attended`
+ `hours of sleep the night before game`
+ `# meals on day prior to game`
+ `university year`
+ `curling` + `gymnastics` + `baseball` + `martial_arts`
+ `frisbee` + `table_tennis` + `basketball`
+ evening + morning + night_owl
+ night_owl * morning + night_owl * evening,
data=boxcox_all_games)

# Fit the model and plot residuals
boxcox_all_games <- subset(boxcox_all_games, select = -c(`fit`, `se.fit`, `residual`))
boxcox_all_games <- subset(cbind(boxcox_all_games,predict(fin_boxcox_overall_model, se.fit = TRUE)) %>%
mutate(residual = `game score` - `fit`), select = -c(`df`, `residual.scale`))

standard_residuals <- rstandard(fin_boxcox_overall_model)
png("../OUTPUTS/boxcox_residuals.png")

# Plot
ggplot(boxcox_all_games, aes(x = fit, y = standard_residuals)) +
geom_point(aes(color = night_owl)) +
ggtitle("Box-Cox Residuals") +
theme(plot.title = element_text(hjust = 0.5, vjust = 0.5))


dev.off()
```

```
## RStudioGD
## 2
```

```
#####

# ALL GAMES ANALYSIS

## Errors are approximately normal
ggplot(data.frame(overallmodel$residuals), aes(x=overallmodel$residuals)) +
geom_histogram(binwidth=1, color="grey50", fill="lightblue", alpha=0.8) +
theme_minimal() +
stat_function(fun = function(x){dnorm(x,mean = mean(overallmodel$residuals),
sd = sd(overallmodel$residuals))}*length(overallmodel$residuals)*1
```

 plot of chunk unnamed-chunk-1

```
# Number of meals is initially positively correlated
model <- lm(`game score` ~ `# meals on day prior to game`,
data=all_games)
summary(model)
```


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



```
## lm(formula = `game score` ~ `# meals on day prior to game`, data = all_games)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -38.390 -14.022  -1.228   13.923   62.334
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)      38.5345     0.6023   63.976 <2e-16 ***
## `# meals on day prior to game`  -0.1447     0.1480   -0.978    0.328
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 20.27 on 9998 degrees of freedom
## Multiple R-squared:  9.561e-05,    Adjusted R-squared:  -4.4e-06
## F-statistic: 0.956 on 1 and 9998 DF,  p-value: 0.3282
```

```
ggplot(all_games, aes(x=`# meals on day prior to game`,y=`game score`)) +
  stat_smooth(method = "lm") +
  geom_point() +
  theme_minimal()
```

```
## `geom_smooth()` using formula = 'y ~ x'
```


 plot of chunk unnamed-chunk-1

```
# But after accounting for training sessions attended, it's negatively correlated (and significantly so)
model <- lm(`game score` ~ `# meals on day prior to game` + `percent training sessions attended`,
  data=all_games)
summary(model)
```

```
##
## Call:
## lm(formula = `game score` ~ `# meals on day prior to game` +
##   `percent training sessions attended`, data = all_games)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -53.25 -11.38    0.33   11.60   82.91
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)      27.5884     0.4932   55.93 <2e-16 ***
## `# meals on day prior to game`  -1.7495     0.1181  -14.82 <2e-16 ***
## `percent training sessions attended`  33.0892     0.4202   78.74 <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.93 on 9997 degrees of freedom
## Multiple R-squared:  0.3828, Adjusted R-squared:  0.3827
## F-statistic: 3101 on 2 and 9997 DF,  p-value: < 2.2e-16
```

```
ggplot(all_games, aes(x=`# meals on day prior to game`,y=`percent training sessions attended`)) +
  stat_smooth(method = "lm") +
  geom_point() +
  theme_minimal()
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

 plot of chunk unnamed-chunk-1

```
# There's two ways to manage this: we can say that there might be an unknown
# causal relationship present and so it's simply impossible to vary the number
# of meals eaten without affecting the other covariates, which is what the
# beta value represents (think of the midterm). OR we can ignore that and say
# we should have them eat less meals for better coaching - I think the former
# is better.
```

```
## Check for correlation between covariates
used_covariates <- c("percent training sessions attended",
```

```

      "overall fitness score",
      "# extra strategy sessions attended",
      "hours of sleep the night before game",
      "# meals on day prior to game",
      "university year",
      "curling", "gymnastics", "baseball", "martial_arts",
      "frisbee", "table_tennis", "basketball",
      "evening", "morning", "night_owl")


correlations <- c()
for(var in used_covariates) {
  correlations <- append(correlations, summary(lm(as.formula(paste("`", var, "~.", sep = "")), data =
                                                    all_games[,names(all_games) %in% used_covariates]))$r.s
}

correlations_by_covar <- data.frame(used_covariates, correlations)

if(length(correlations_by_covar$correlations[correlations_by_covar$correlations >= 0.8]) > 0) {
  print("Some variables are very correlated (inflation factor > 5)!")
}

## Check for non-linearity
ggplot(all_games, aes(x = `percent training sessions attended`, y = residual)) +
  geom_point(color = 'black') +
  ggtitle("Percent training sessions attended residuals")


```

 plot of chunk unnamed-chunk-1

```

ggplot(all_games, aes(x = `overall fitness score`, y = residual)) +
  geom_point(color = 'black') +
  ggtitle("Overall fitness score residuals")


```

 plot of chunk unnamed-chunk-1

```

ggplot(all_games, aes(x = `# extra strategy sessions attended`, y = residual)) +
  geom_point(color = 'black') +
  ggtitle("# extra strategy sessions attended residuals")


```

 plot of chunk unnamed-chunk-1

```

ggplot(all_games, aes(x = `hours of sleep the night before game`, y = residual)) +
  geom_point(color = 'black') +
  ggtitle("Hours of sleep the night before game residuals")


```

 plot of chunk unnamed-chunk-1

```

ggplot(all_games, aes(x = `# meals on day prior to game`, y = residual)) +
  geom_point(color = 'black') +
  ggtitle("# meals on day prior to game the night before game residuals")

```

 plot of chunk unnamed-chunk-1

```

## Models for morning and evening games

# This is useful for checking whether our model is a good fit -
# specifically, whether the relationship between game score and
# the other covariates depends on whether it's a morning or evening game.

# Notably, the most significant difference (by far) between the two is
# for the night_owl indicator variable - which is what we expect to see,
# intuitively. None of the other ones are particularly significant,
# especially with the multiple testing fallacy. There is also a significant
# difference in the intercept - which makes sense too, because the intercept
# represents early birds. So early birds have an 11 point difference (after
# accounting for changes in the other variables) between morning and evening
# games, while night owls only have about a 3 point difference

eveningmodel <- lm(`game score` ~ `percent training sessions attended`
  + `overall fitness score`
  + `# extra strategy sessions attended`
  + `hours of sleep the night before game`
  + `# meals on day prior to game`
  + `university year` + `curling` + `gymnastics` + `baseball` + `martial_arts`
  + `frisbee` + `table_tennis` + `basketball`
  + night_owl,

```

```
data=filter(all_games, evening == 1))
summary(eveningmodel)
```

```
##
## Call:
## lm(formula = `game score` ~ `percent training sessions attended` +
##   `overall fitness score` + `# extra strategy sessions attended` +
##   `hours of sleep the night before game` + `# meals on day prior to game` +
##   `university year` + curling + gymnastics + baseball + martial_arts +
##   frisbee + table_tennis + basketball + night_owl, data = filter(all_games,
##     evening == 1))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -33.930  -5.810  -0.054   5.764  72.185
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -0.70575     1.32200   -0.534   0.5935
## `percent training sessions attended` 28.23703     0.62704  45.032 < 2e-16 ***
## `overall fitness score`      23.80548     0.63505  37.486 < 2e-16 ***
## `# extra strategy sessions attended`  1.74739     0.07604  22.980 < 2e-16 ***
## `hours of sleep the night before game` 0.35386     0.06060   5.839 6.05e-09 ***
## `# meals on day prior to game`    -0.98684     0.17316  -5.699 1.37e-08 ***
## `university year`              0.25961     0.24179   1.074   0.2831
## curling          -0.66593     0.74636  -0.892   0.3724
## gymnastics       -1.79243     0.97922  -1.830   0.0673 .
## baseball        -1.76261     0.82572  -2.135   0.0329 *
## martial_arts      6.48938     0.78423   8.275 2.23e-16 ***
## frisbee          6.23494     0.81790   7.623 3.69e-14 ***
## table_tennis     -1.12085     0.77856  -1.440   0.1501
## basketball       5.49054     0.95929   5.724 1.19e-08 ***
## night_owl        8.24885     0.54648  15.094 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.511 on 2155 degrees of freedom
## Multiple R-squared:  0.7782, Adjusted R-squared:  0.7768
## F-statistic: 540.1 on 14 and 2155 DF, p-value: < 2.2e-16
```

```
morningmodel <- lm(`game score` ~ `percent training sessions attended`
+ `overall fitness score`
+ `# extra strategy sessions attended`
+ `hours of sleep the night before game`
+ `# meals on day prior to game`
+ `university year` + `curling` + `gymnastics` + `baseball` + `martial_arts`
+ `frisbee` + `table_tennis` + `basketball`
+ night_owl,
data=filter(all_games, morning == 1))
summary(morningmodel)
```

```
##
## Call:
## lm(formula = `game score` ~ `percent training sessions attended` +
##   `overall fitness score` + `# extra strategy sessions attended` +
##   `hours of sleep the night before game` + `# meals on day prior to game` +
##   `university year` + curling + gymnastics + baseball + martial_arts +
##   frisbee + table_tennis + basketball + night_owl, data = filter(all_games,
##     morning == 1))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -28.857  -6.537   0.006   6.318  69.449
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)      8.4525     2.3557   3.588 0.000350 ***
## `percent training sessions attended` 28.4781     1.0550  26.993 < 2e-16 ***
## `overall fitness score`      23.1455     1.0682  21.668 < 2e-16 ***
## `# extra strategy sessions attended`  1.4360     0.1305  11.006 < 2e-16 ***
## `hours of sleep the night before game` 0.2130     0.1004   2.122 0.034076 *
## `# meals on day prior to game`    -1.0375     0.2923  -3.549 0.000405 ***
```

```
## `university year`          -0.3495      0.4315  -0.810  0.418179
## curling                    3.4971      1.3667   2.559  0.010652 *
## gymnastics                 0.3061      1.8514   0.165  0.868738
## baseball                  2.8432      1.3677   2.079  0.037895 *
## martial_arts              9.0980      1.3813   6.586  7.36e-11 ***
## frisbee                   9.4807      1.4567   6.508  1.21e-10 ***
## table_tennis              4.2273      1.3707   3.084  0.002100 **
## basketball                8.3717      1.7954   4.663  3.55e-06 ***
## night_owl                 -7.2238      0.9380  -7.701  3.30e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.08 on 975 degrees of freedom
## Multiple R-squared:  0.6969, Adjusted R-squared:  0.6925
## F-statistic: 160.1 on 14 and 975 DF,  p-value: < 2.2e-16
```

```
## MORNING AND EVENING RESIDUAL PLOTS
```

```
# Let's explore the residual plots of each of these models. Start w/ morning:
morning_residuals <- filter(all_games, morning == 1) %>%
  dplyr::select(`game score`, `early bird/night owl`)

morning_residuals <- cbind(morning_residuals, predict(morningmodel, se.fit = TRUE)) %>%
  mutate(residuals = `game score` - fit)

# General residual plot

png("../OUTPUTS/Morning_General_Residuals.png")

ggplot(morning_residuals, aes(x = fit, y = residuals)) +
  geom_point(color = 'green') +
  ggtitle("Morning Model Residuals")

dev.off()
```

```
## RStudioGD
##          2
```

```
png("../OUTPUTS/Morning_Grouped_Residuals.png")


# Grouping by night owl/early bird
ggplot(morning_residuals, aes(x = fit, y = residuals)) +
  geom_point(aes(color = `early bird/night owl`)) +
  ggtitle("Morning Model Residuals")

dev.off()
```

```
## RStudioGD
##          2
```

```
# Let's explore the evening model now!
evening_residuals <- filter(all_games, evening == 1) %>%
  select(`game score`, `early bird/night owl`)
```

```
## Error in select(., `game score`, `early bird/night owl`): unused arguments (`game score`, `early bird/n:
```

 plot of chunk unnamed-chunk-3

```
evening_residuals <- cbind(evening_residuals, predict(eveningmodel, se.fit = TRUE)) %>%
  mutate(residuals = `game score` - fit)
```

```
## Error in cbind(evening_residuals, predict(eveningmodel, se.fit = TRUE)): object 'evening_residuals' not
```

```
# General residual plot
png("../OUTPUTS/Evening_General_Residuals.png")

ggplot(evening_residuals, aes(x = fit, y = residuals)) +
```

```
geom_point(color = 'green') +  
ggtitle("Evening Model Residuals")
```

```
## Error in ggplot(evening_residuals, aes(x = fit, y = residuals)): object 'evening_residuals' not found
```

```
dev.off()
```

```
## RStudioGD  
##      2
```

```
png("../OUTPUTS/Evening_Grouped_Residuals.png")  
# Grouping by night owl/early bird  
ggplot(evening_residuals, aes(x = fit, y = residuals)) +  
  geom_point(aes(color = `early bird/night owl`)) +  
  ggtitle("Evening Model Residuals")
```

```
## Error in ggplot(evening_residuals, aes(x = fit, y = residuals)): object 'evening_residuals' not found
```

```
dev.off()
```

```
## RStudioGD  
##      2
```

```
## HYPOTHESIS TESTING
```

```
# extract betas and standard errors from evening and morning models  
evening_betas <- eveningmodel$coefficients  
morning_betas <- morningmodel$coefficients  
evening_errors <- sqrt(diag(vcov(eveningmodel)))  
morning_errors <- sqrt(diag(vcov(morningmodel)))
```

```
# run t-test between evening and morning betas, using adjusted  
# significance threshold  
new_alpha = 0.05 / length(morning_betas)  
new_conf_level = 1 - new_alpha  
names_betas <- names(morning_betas)
```

```
for (i in 1:length(evening_betas)) {  
  betas <- c(evening_betas[i], morning_betas[i])  
  std_dev <- sqrt((evening_errors[i])^2 + (morning_errors[i])^2)  
  print(names_betas[i])  
  print(t.test(betas, mu=0, sd=std_dev, conf.level=new_conf_level))  
}
```

```
## [1] "(Intercept)"  
##  
##      One Sample t-test  
##  
## data:  betas  
## t = 0.84588, df = 1, p-value = 0.553  
## alternative hypothesis: true mean is not equal to 0  
## 99.66667 percent confidence interval:  
## -870.6655  878.4122  
## sample estimates:  
## mean of x  
##  3.873362  
##  
## [1] "`percent training sessions attended`"  
##  
##      One Sample t-test  
##  
## data:  betas  
## t = 235.28, df = 1, p-value = 0.002706  
## alternative hypothesis: true mean is not equal to 0  
## 99.66667 percent confidence interval:  
##  5.338873 51.376233
```

```

## sample estimates:
## mean of x
## 28.35755
##
## [1] "`overall fitness score`"
##
##      One Sample t-test
##
## data:  betas
## t = 71.138, df = 1, p-value = 0.008948
## alternative hypothesis: true mean is not equal to 0
## 99.66667 percent confidence interval:
## -39.54874 86.49971
## sample estimates:
## mean of x
## 23.47549
##
## [1] "`# extra strategy sessions attended`"
##
##      One Sample t-test
##
## data:  betas
## t = 10.223, df = 1, p-value = 0.06208
## alternative hypothesis: true mean is not equal to 0
## 99.66667 percent confidence interval:
## -28.14528 31.32865
## sample estimates:
## mean of x
## 1.591687
##
## [1] "`hours of sleep the night before game`"
##
##      One Sample t-test
##
## data:  betas
## t = 4.0246, df = 1, p-value = 0.155
## alternative hypothesis: true mean is not equal to 0
## 99.66667 percent confidence interval:
## -13.16658 13.73344
## sample estimates:
## mean of x
## 0.2834337
##
## [1] "`# meals on day prior to game`"
##
##      One Sample t-test
##
## data:  betas
## t = -39.954, df = 1, p-value = 0.01593
## alternative hypothesis: true mean is not equal to 0
## 99.66667 percent confidence interval:
## -5.850443 3.826102
## sample estimates:
## mean of x
## -1.012171
##
## [1] "`university year`"
##
##      One Sample t-test
##
## data:  betas
## t = -0.14756, df = 1, p-value = 0.9067
## alternative hypothesis: true mean is not equal to 0
## 99.66667 percent confidence interval:
## -58.20871 58.11883
## sample estimates:
## mean of x
## -0.04494004
##
## [1] "curling"
##
##      One Sample t-test
##
## data:  betas
## t = 0.68007, df = 1, p-value = 0.6198

```

```

## alternative hypothesis: true mean is not equal to 0
## 99.66667 percent confidence interval:
## -396.1205 398.9517
## sample estimates:
## mean of x
## 1.41558
##
## [1] "gymnastics"
##
## One Sample t-test
##
## data: betas
## t = -0.70831, df = 1, p-value = 0.6077
## alternative hypothesis: true mean is not equal to 0
## 99.66667 percent confidence interval:
## -201.1318 199.6454
## sample estimates:
## mean of x
## -0.7431868
##
## [1] "baseball"
##
## One Sample t-test
##
## data: betas
## t = 0.23462, df = 1, p-value = 0.8533
## alternative hypothesis: true mean is not equal to 0
## 99.66667 percent confidence interval:
## -439.2816 440.3622
## sample estimates:
## mean of x
## 0.5403108
##
## [1] "martial_arts"
##
## One Sample t-test
##
## data: betas
## t = 5.9754, df = 1, p-value = 0.1056
## alternative hypothesis: true mean is not equal to 0
## 99.66667 percent confidence interval:
## -241.3075 256.8949
## sample estimates:
## mean of x
## 7.793683
##
## [1] "frisbee"
##
## One Sample t-test
##
## data: betas
## t = 4.8419, df = 1, p-value = 0.1297
## alternative hypothesis: true mean is not equal to 0
## 99.66667 percent confidence interval:
## -302.0848 317.8004
## sample estimates:
## mean of x
## 7.857811
##
## [1] "table_tennis"
##
## One Sample t-test
##
## data: betas
## t = 0.58084, df = 1, p-value = 0.665
## alternative hypothesis: true mean is not equal to 0
## 99.66667 percent confidence interval:
## -509.1493 512.2557
## sample estimates:
## mean of x
## 1.553208
##
## [1] "basketball"
##
## One Sample t-test

```

```
##
## data: betas
## t = 4.8114, df = 1, p-value = 0.1305
## alternative hypothesis: true mean is not equal to 0
## 99.66667 percent confidence interval:
## -268.1941 282.0563
## sample estimates:
## mean of x
## 6.931109
##
## [1] "night_owl"
##
##      One Sample t-test
##
## data: betas
## t = 0.066247, df = 1, p-value = 0.9579
## alternative hypothesis: true mean is not equal to 0
## 99.66667 percent confidence interval:
## -1477.006 1478.031
## sample estimates:
## mean of x
## 0.5125065
```

```
## Finding the best morning & evening players
```

```
# Make a hypothetical version of the data where every game was played
# in the or in the evening
morningized_games <- all_games
morningized_games$morning <- 1
morningized_games$evening <- 0

eveningized_games <- all_games
eveningized_games$morning <- 0
eveningized_games$evening <- 1

# Set the variables we see as mutable to the average to remove their effects.
# We decided the training and strategy sessions could be controlled
# the coach on real teams, so we shouldn't consider their impact
# on an individual's score.
morningized_games$`percent training sessions attended` <- max(all_games$`percent training sessions attended`)
morningized_games$`# extra strategy sessions attended` <- max(all_games$`# extra strategy sessions attended`)
eveningized_games$`percent training sessions attended` <- max(all_games$`percent training sessions attended`)
eveningized_games$`# extra strategy sessions attended` <- max(all_games$`# extra strategy sessions attended`)

# Then predict how the players would have scored. We can use
# overallmodel for this, because we know that the only night_owl
# has a significant difference between morning and evening games
# and its interaction with that has been accounted for.

# It's better to use overallmodel to avoid bias due to
# non-statistically significant, but still potentially impactful,
# differences between overallmodel and morning/evening model
morningized_games$`game score` <- predict.lm(overallmodel, morningized_games)
eveningized_games$`game score` <- predict.lm(overallmodel, eveningized_games)

# Adjust by the residual, in case there is some confounding variable
# biasing them. (The residual in these dataframes is still the
# residual from all_games, which is useful here.)
morningized_games$`game score` <- morningized_games$`game score` + morningized_games$residual
eveningized_games$`game score` <- eveningized_games$`game score` + eveningized_games$residual

# Then, average these by player
morning_players <- morningized_games %>% group_by(`student label`) %>% summarize(
  mean = mean(`game score`),
  sd = sd(`game score`)
)
top_morning <- top_n(morning_players, 20, mean)
top_morning <- top_morning[order(top_morning$mean, decreasing = TRUE), ]

evening_players <- eveningized_games %>% group_by(`student label`) %>% summarize(
  mean = mean(`game score`),
  sd = sd(`game score`)
)
```



```

top_evening <- top_n(evening_players,20,mean)
top_evening <- top_evening[order(top_evening$mean,decreasing = TRUE),]

best_morning <- c()
best_evening <- c()
for(i in 1:20) {
  if(!(top_morning$`student label`[i] %in% best_evening)) {
    if(top_morning$`student label`[i] ==
      top_evening$`student label`[i]) {
      if(top_morning$mean[i] - mean(top_morning$mean) >
        top_evening$mean[i] - mean(top_evening$mean)) {
        best_morning <- append(best_morning, top_morning$`student label`[i])
      } else {
        best_evening <- append(best_evening, top_evening$`student label`[i])
      }
    } else {
      best_morning <- append(best_morning, top_morning$`student label`[i])
      best_evening <- append(best_evening, top_evening$`student label`[i])
    }
  }
}

# Keep only the top 10 from each. Those are our teams!
best_morning <- best_morning[1:10]
write.csv(data.frame(best_morning), "../PROCESSED_DATA/west_coast_team.csv")
best_evening <- best_evening[1:10]
write.csv(data.frame(best_evening), "../PROCESSED_DATA/east_coast_team.csv")

```

```
## Find our regular season team
```

```

# Create another set for all noon games
noonized_games <- all_games
noonized_games$morning <- 0
noonized_games$evening <- 0

# Remove the players used for east coast/west coast cups
noonized_games <- filter(noonized_games,!(`student label` %in% best_morning))
noonized_games <- filter(noonized_games,!(`student label` %in% best_evening))
morningized_games <- filter(noonized_games,!(`student label` %in% best_morning))
morningized_games <- filter(noonized_games,!(`student label` %in% best_evening))
eveningized_games <- filter(noonized_games,!(`student label` %in% best_morning))
eveningized_games <- filter(noonized_games,!(`student label` %in% best_evening))

# Again, we want to remove the effect of the training and strategy sessions.
# However, because we care about absolute scores, and not relative scores,
# we can't just assign them all to the mean. Instead, it makes more sense
# to assign them to the max: we want our players to be the best, so
# we will send them to as many training and strategy sessions as possible.
noonized_games$`percent training sessions attended` <- max(all_games$`percent training sessions attended`)
noonized_games$`# extra strategy sessions attended` <- max(all_games$`# extra strategy sessions attended`)
noonized_games$`game score` <- predict.lm(overallmodel, noonized_games)
noonized_games$`game score` <- noonized_games$`game score` + noonized_games$residual

morningized_games$`percent training sessions attended` <- max(all_games$`percent training sessions attended`)
morningized_games$`# extra strategy sessions attended` <- max(all_games$`# extra strategy sessions attended`)
morningized_games$`game score` <- predict.lm(overallmodel, morningized_games)
morningized_games$`game score` <- morningized_games$`game score` + morningized_games$residual

eveningized_games$`percent training sessions attended` <- max(all_games$`percent training sessions attended`)
eveningized_games$`# extra strategy sessions attended` <- max(all_games$`# extra strategy sessions attended`)
eveningized_games$`game score` <- predict.lm(overallmodel, eveningized_games)
eveningized_games$`game score` <- eveningized_games$`game score` + eveningized_games$residual

adjusted_games <- rbind(noonized_games,morningized_games,eveningized_games)

# Summarize the average of the three by player
all_players <- adjusted_games %>% group_by(`student label`) %>% summarize(
  mean = mean(`game score`),
  sd = sd(`game score`)
)

# Naively select the best 10, and compare everyone to the 10th best


```

```
best_10 <- top_n(all_players,10,mean)
best_10 <- best_10[order(best_10$mean,decreasing = TRUE),]


all_players <- all_players %>% mutate(
  chance_better_than_10 = pnorm((mean - best_10$mean[10])/sqrt(sd^2 + best_10$sd[10]^2))
)

# In order to have a 5% chance of being better than the tenth best
# player in 10 of the 20 games, a player needs to have:
# 20 C 10 * x^10 = 0.05
# x = 0.22
# a 22% chance of being better than the tenth best player on a
# single game.

# All the remaining players:
ggplot(all_players, aes(x = mean, y = sd)) +
  geom_point() +
  ggtitle("Players for general season") +
  scale_x_continuous(limits = c(0,100))
```

 plot of chunk unnamed-chunk-9

```
# Same, but only the players with a >5% chance of being better
# than the 10th best player on average in over half the games
ggplot(filter(all_players, chance_better_than_10 > 0.22), aes(x = mean, y = sd)) +
  geom_point() +
  ggtitle("Top cut of players for general season") +
  scale_x_continuous(limits = c(0,100))
```

 plot of chunk unnamed-chunk-9

```
# Splits the players by standard deviation, the organizes them
# in each SD group by mean, keeping only the 10 best means

## DO NOT RUN, TAKES ~13.5 HOURS

# bucketed_players <- partition(all_players,"sd","mean",2.5,10)
#
# # Test how many runs it'll take
# a <- numeric(length(bucketed_players))
# a[1] <- 10
# i <- 0
# while(sum(a)==10){
#   a <- increment_amounts(bucketed_players,a)
#   i <- i + 1
# }
# print(paste(i, "runs incoming"))
#
# win_chances <- read_csv("../PROCESSED_DATA/confidence_win_rates.csv")
#
# all_possible_teams <- bucket_combo_teams(bucketed_players, 4, win_chances$chance_winning,
#                                           win_chances$score_difs, match_ups, 0, i)
#
# 13.5 hour run successfully done and saved here, please don't overwrite
# write.csv(all_possible_teams, "../PROCESSED_DATA/all_possible_teams.csv")
all_possible_teams <- read_csv("../PROCESSED_DATA/all_possible_teams.csv")
```

```
## New names:
## Rows: 369512 Columns: 13
## — Column specification
## _____ Delimiter: "," dbl
## (13): ...1, player1, player2, player3, player4, player5, player6, player7, player8...
## i Use `spec()` to retrieve the full column specification for this data. i Specify the
## column types or set `show_col_types = FALSE` to quiet this message.
## • `` -> `...1`
```

```
# Double check these teams with higher precision to guarantee they're
# the best

# Takes ~13 minutes to check if you limit it to priority <= 30
# because it's extremely high precision

# best_teams <- all_possible_teams %>% mutate(
```

```

#   avg_place = ntile(avg_wins,nrow(all_possible_teams)),
#   und_place = ntile(undefeated_chance,nrow(all_possible_teams)),
#   priority = nrow(all_possible_teams)
# )
#
# for(i in 1:nrow(best_teams)) {
#   best_teams[i,"priority"] = min(best_teams[i,"und_place"],
#                                   best_teams[i,"avg_place"])
# }
#
# best_teams <- best_teams[order(best_teams$priority,
#                                 decreasing = FALSE),]
#
# # You can just let this run for however long you have, it goes back
# # through the existing data and retabulates the the success measures
# # with maximum precision, in descending order- from the ones that
# # performed best on the imprecise run to the ones that performed
# # worst
# for(i in 1:nrow(best_teams)) {
#   players <- filter(all_players,`student label` %in% best_teams[i,1:10])
#   team_pdf <- score_pdf_from_players(players$mean,players$sd)
#   team_cdf <- score_cdf_from_players(players$mean,players$sd)
#   results <- success_measures(win_chances$chance_winning, win_chances$score_difs,
#                                team_pdf, team_cdf, match_ups, 2,
#                                mean(players$mean) - 8 * sqrt(sum(players$sd^2)),
#                                mean(players$mean) + 8 * sqrt(sum(players$sd^2)))
#   best_teams[i,"avg_wins"] <- results[1]
#   best_teams[i,"undefeated_chance"] <- results[2]
#   if(best_teams[i,"priority"] %% 10 == 0) {
#     print(paste(best_teams[i,"priority"],"out of",nrow(best_on_average_teams),"rows"))
#   }
# }
#
# # Notably, the #1 spot doesn't change on this one
# best_on_average_teams <- best_teams[order(best_teams$avg_wins,
#                                             decreasing = TRUE),]
# best_on_average_teams <- best_on_average_teams[1:10,]
#
# best_undefeated_teams <- best_teams[order(best_teams$undefeated_chance,
#                                             decreasing = TRUE),]
# best_undefeated_teams <- best_undefeated_teams[1:10,]
#
# write.csv(best_on_average_teams, "../PROCESSED_DATA/best_on_average_teams.csv")
# write.csv(best_undefeated_teams, "../PROCESSED_DATA/best_undefeated_teams.csv")

best_on_average_teams <- read_csv("../PROCESSED_DATA/best_on_average_teams.csv")

```

```

## New names:
## Rows: 10 Columns: 13
## — Column specification
## _____ Delimiter: "," dbl
## (13): ...1, player1, player2, player3, player4, player5, player6, player7, player8...
## i Use `spec()` to retrieve the full column specification for this data. i Specify the
## column types or set `show_col_types = FALSE` to quiet this message.
## • `` -> `...1`

```

```
best_undefeated_teams <- read_csv("../PROCESSED_DATA/best_undefeated_teams.csv")
```

```

## New names:
## Rows: 10 Columns: 13
## — Column specification
## _____ Delimiter: "," dbl
## (13): ...1, player1, player2, player3, player4, player5, player6, player7, player8...
## i Use `spec()` to retrieve the full column specification for this data. i Specify the
## column types or set `show_col_types = FALSE` to quiet this message.
## • `` -> `...1`

```

The R session information (including the OS info, R version and all packages used):

```
sessionInfo()
```

```
## R version 4.2.2 (2022-10-31)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Monterey 12.6
##
## Matrix products: default
## LAPACK: /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats      graphics  grDevices  utils      datasets  methods    base
##
## other attached packages:
## [1] tinytex_0.43      MASS_7.3-58.1    rstudioapi_0.14  haven_2.5.1      lubridate_1.9.2
## [6] forcats_1.0.0     stringr_1.5.0    dplyr_1.1.0      purrr_1.0.1      readr_2.1.4
## [11] tidyr_1.3.0       tibble_3.1.8     ggplot2_3.4.1    tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
## [1] highr_0.10        pillar_1.8.1     compiler_4.2.2   tools_4.2.2      bit_4.0.5
## [6] evaluate_0.19     lattice_0.20-45  nlme_3.1-160     lifecycle_1.0.3  gtable_0.3.1
## [11] timechange_0.2.0  mgcv_1.8-41      pkgconfig_2.0.3  rlang_1.0.6      Matrix_1.5-1
## [16] cli_3.6.0         parallel_4.2.2   xfun_0.36        knitr_1.41       withr_2.5.0
## [21] generics_0.1.3    vctrs_0.5.2      hms_1.1.2        bit64_4.0.5      grid_4.2.2
## [26] tidyselect_1.2.0  glue_1.6.2       R6_2.5.1         fansi_1.0.3      vroom_1.6.0
## [31] farver_2.1.1      tzdb_0.3.0       magrittr_2.0.3   splines_4.2.2    scales_1.2.1
## [36] ellipsis_0.3.2    colorspace_2.0-3 labeling_0.4.2    utf8_1.2.2       stringi_1.7.12
## [41] munsell_0.5.0     crayon_1.5.2
```

```
Sys.time()
```

```
## [1] "2023-03-20 00:07:11 PDT"
```