A Bayesian Approach to March Madness Prediction

Mohamad Daniel Bairakdar Jacob Cohen Hoang Luong Bryce Shellow

Abstract

In this work, we train a Bayesian logistic regression model on past college basketball season statistics to predict the outcome of the 2022 season March Madness games. We base our predictions on multiple factors, including a variety of regular season box score statistics, ordinal team rankings from multiple ranking systems, and March Madness seedings. We investigate the effects of including box score statistics from less recent seasons at the expense of ranking data, and explore the sensitivity of our model to prior specification for the different ordinal ranking systems. Our best model achieves an average cross validation accuracy of 66.4%, and an accuracy of 58.0% when tested on the 2022 season March Madness games.

1 Introduction

Sports analytics has been a vastly growing field, especially over the past few years. The analytics began in baseball with the Moneyball A's in the early 2000s, and has now expanded to almost all other sports. These include other team sports such as basketball, football, and soccer; individual sports like tennis and golf; and combat sports such as boxing and the UFC. We decided to attempt to predict the outcome of individual March Madness games. March Madness - a 68-team tournament that takes place at the end of the Division I basketball season - is a statistical goldmine with more possible outcomes than one can count. For every season, millions of people across the country make their brackets, but few are able to accurately predict the outcomes. In fact, there has never been a perfect bracket in the history of the tournament. The longest someone has ever gone with a perfect bracket is only 49 of the 63 games. This is in large part due to the fact that there are 9.2 quintillion possibilities for it. While we did not attempt to predict the bracket, we took a step in the direction of predicting it by attempting to predict the winner of any possible matchup between two tournament teams. In theory, once a strong model of this kind is built, it could be stacked sequentially to predict the bracket. In order to analyze each tournament matchup, we used regular season box score statistics, ordinal team rankings, and March Madness seedings to see if they were indicative of March Madness outcomes. We

fitted a Bayesian logistic regression using an MCMC computing algorithm and used cross-validation as a diagnostic tool to pick the best model. We then tested our model by predicting the outcomes of the 2022 season March Madness games. Ours is not the first work to attempt to predict the outcome of college basketball games. Brown (2019) builds a logistic regression for this purpose, but does not use a Bayesian approach. Nelson (2012) employs an approach similar to ours, but only uses box score statistics to predict the outcome, does not investigate the effect of adding or removing data from less recent seasons - data which could impact the prediction since the nature of the game has changed over the years - and does not investigate model sensitivity to prior specification.

2 Data

2.1 Data Source

We obtained our data from the Kaggle dataset "March Machine Learning Mania 2022 - Men's" [1], which contains single game regular season and tournament box score data dating back to the 1984-1985 college basketball season. Detailed box score data is only available starting from the 2002-2003 season, so we discard the data from earlier seasons. Due to the 2020 Covid-19 pandemic, the National Collegiate Athletic Association (NCAA) canceled the tournament for that year, and thus we did not include 2019-2020 regular season data as the corresponding tournament data was missing.

2.2 Covariates: Regular season data

The regular season data consisted of individual game logs containing variables indicating the year of the season, the day the game occurred in relation to the start of the season, the location of the game, number of overtimes that occurred in the game (if any), the team ID for the winning and losing teams, and their respective box score statistics. These statistics include the number of points scored, the number of field goals (FG) made and attempted, 3 point FG made and attempted, free throws (FT) made and attempted, offensive and defensive rebounds, assists made, turnovers committed, blocks and steals made, and personal fouls committed. The same data was available for the tournament. The Kaggle dataset also contained ordinal rankings from multiple basketball metric sites. For each team, this data included the season, ranking system name, ordinal rank, and day of the ranking since the start of the season - rankings are provided on a weekly basis. Finally, the dataset included the tournament seeding for each team. Figures 1a and 1b show samples of raw box score statistics and ranking data, respectively.

To prepare the data to be input into our model, for each season, we discard data for teams that did not participate in March Madness for that season. Next, we combined regular season team box

[1] https://www.kaggle.com/c/mens-march-mania-2022/data

	Season	DayNum	WTeamID	WScore	LTeamID	LScore	WLoc	NumOT	WFGM	WFGA	WFGM3
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>
1	2003	10	1104	68	1328	62	N	0	27	58	3
2	2003	10	1272	70	1393	63	N	0	26	62	8
3	2003	11	1266	73	1437	61	N	0	24	58	8
4	2003	11	1296	56	1457	50	N	0	18	38	3
5	2003	11	1400	77	1208	71	N	0	30	61	6
6	2003	11	1458	81	1186	55	H	0	26	57	6

… with 23 more variables: WFGA3 <dbl>, WFTM <dbl>, WFTA <dbl>, WOR <dbl>,
WDR <dbl>, WAst <dbl>, WTO <dbl>, WStl <dbl>, WBlk <dbl>, WPF <dbl>,
LFGM <dbl>, LFGA <dbl>, LFGM3 <dbl>, LFGA3 <dbl>, LFTM <dbl>, LFTA <dbl>,
LOR <dbl>, LDR <dbl>, LAst <dbl>, LTO <dbl>, LStl <dbl>, LBlk <dbl>,
LPF <dbl>

(a) Box score statistics

	Season	RankingDayNum	SystemName	TeamID	OrdinalRank
	<dbl>	<dbl>	<chr>	<dbl>	<dbl>
1	2003	35	SEL	1102	159
2	2003	35	SEL	1103	229
3	2003	35	SEL	1104	12
4	2003	35	SEL	1105	314
5	2003	35	SEL	1106	260
6	2003	35	SEL	1107	249

(b) Ordinal team rankings

Figure 1: Raw data samples

score winning and losing statistics by grouping by team ID and season to create "for" and "against" season averages. For certain variables like field goals attempted and made, we compute average field goal percentage, rather than an average for each of field goals attempted and made separately. We then created variables prefixed with "current" or "other." "current" indicates the "for" values for the team - in other words, the numbers that the team produced throughout the season - while "other" indicates "against" values, which are the averages and percentages produced by the team's opponents throughout the season. To obtain individual training instances for our model - where each instance consists of a March Madness matchup between two teams - for each variable we create, we compute a difference between the values for those variables for each team participating in a given game. For example, if Michigan is the first team listed in a Michigan-Kansas 2016 tournament match-up, and Michigan had an average FG% of 0.53 while Kansas had a FG% of 0.43 over the 2015-2016 regular season, then the covariate representing the difference in "current" FG% would be 0.10. We note that one of our variables represents difference in game winning percentage between the two teams pitted against each other; we did not include the difference in the losing percentage, as that would just be the opposite of the difference in the winning percentage.

As explained later, we were interested in evaluating model performance for models built using data spanning different timeframes; a model built using data from the 2016-2021 seasons versus a model built using the data from the 2003-2021 seasons. The ranking information we obtained depended on the number of seasons we used to build our model. Within a given timeframe of interest, there were rankings from certain ranking systems that were available for one season but not another. In those cases, we excluded those ranking systems. Other ranking systems had rankings for some teams but not others. We also excluded those ranking systems. An alternative approach would be to impute those values. Once we ended up with a list of ranking systems that met the above criteria, we then decided based on our prior knowledge of the quality of those rankings which ones to include and which to exclude. The ones we include in our models are: B Wilson Empirical (BWE), Massey (MAS), Pomeroy (POM), and Stat Fox (SFX). For each ranking system, we create two covariates: one that represents the ordinal ranking of the team over the entire regular season, and one that represents

the ordinal ranking right before the start of the tournament. This is to account for the fact that certain teams may be very good teams on average, but may be "out of form" during certain periods of time for many different reasons - most notably because of injuries of key players. In table 1, average rankings over the entire season correspond to variables with the substring "Mean." Lastly, the variable "SeedAdd" was created as an interaction term to account for the difference in competition between teams; there is a larger performance gap between a 9 seed versus a 16 seed than a 1 seed versus an 8 seed. In this case, both scenarios would result in a "SeedDiff" value of 7, whereas we could tell from the lower "SeedAdd" that the 1 and 8 seeds match more closely in skill than the other match-up.

All variables we created are shown in table 1, with their descriptive statistics for the 2016-2021 timeframe. See the Appendix, Section 8.2 for the code used to create the variables. The variables were normalized by subtracting their respective means and standard deviations.

Variable	Mean	Standard Deviation
winPctDiff	0.004052	0.1519
currScoreAvgDiff	0.7782	7.3043
otherScoreAvgDiff	0.3279	6.4514
currFGPctDiff	0.003390	0.0348
otherFGPctDiff	-0.0008	0.02852
currFG3PctDiff	0.003107	0.03848
otherFG3PctDiff	0.0008	0.0306
currFTPctDiff	0.0015	0.0519
otherFTPctDiff	-0.0036	0.0335
currRebAvgDiff	0.3975	3.9076
otherRebAvgDiff	-0.0555	3.3806
currAstAvgDiff	0.1906	2.7380
otherAstAvgDiff	0.08557	2.1618
currTOAvgDiff	0.04426	1.9082
otherTOAvgDiff	-0.1649	2.4988
currDefAvgDiff	-0.0884	2.3694
otherDefAvgDiff	0.0177	1.6250
SeedDiff	-0.5689	7.4985
BWEDiff	-6.4641	74.1355
MASDiff	-6.4671	71.4674
POMDiff	-6.9910	74.8927
SFXDiff	-6.0479	70.4773
POMMeanDiff	-7.7559	81.9615
BWEMeanDiff	-7.2479	79.8221
MASMeanDiff	-7.8061	80.4986
SFXMeanDiff	-6.4981	74.7913
SeedAdd	14.4132	5.8783

Table 1: Descriptive statistics for all variables we created for the 2016-2021 seasons

2.3 Outcome: March Madness data

From the tournament data, we only obtained the IDs for the winning and losing teams for each game. The tournament data was used to create our response variable, so the game statistics are not needed as we are only attempting to predict the outcome based on the regular season values. In order to obtain

a balanced dataset, within each March madness tournament, we randomly shuffle the games and split the resulting data in half so that the first half consists of games considered as "Wins," while the other half consists of games considered as "Losses".

3 Model

3.1 Modeling Approach

We model the problem of predicting the outcome of a March Madness game as a binary task using Bayesian Logistic Regression. When given two teams, Team i and Team j, our model answers the question: will Team i beat Team j when they are pitted against each other? If the answer is yes, the model predicts 1, and if not, it predicts 0. Note that the order in which the teams are input matters. That is, if the same teams i and j are input into the model but in the reverse order, then the model should predict the opposite of what it predicts the first time. Formally, we denote by y_{ij} the indicator variable that represents a win by team i when pitted against team j. If $y_{ij} = 1$, this means team i wins against team j, while 0 means team i loses. We model y_{ij} as follows:

$$\mu_{ij} = logistic(\alpha + \sum_{n=1}^{N} \beta_n x_{n_{ij}} + \beta_{ss} z_{ss_{ij}}) \tag{1}$$

$$y_{ij} \sim Bernoulli(\mu_{ij}) \tag{2}$$

Here, *logistic* is the logistic function $f(x) = \frac{1}{1+e^{-x}}$, N is equal to the number of covariates, α is the intercept term, and $x_{n_{ij}}$ is the difference between the average of variable x_n for team i and the average of variable x_n for team j. Finally, as discussed at the end of Section 2.2, we add an interaction term $z_{ss_{ij}}$, which is the sum of the seeds for teams i and j.

To fit our models, we use an MCMC algorithm using the R package Stan. Specifically, we use 4 chains that run for a total of 10000 iterations each, with 5000 warmup iterations.

3.2 Model specification

We build three models using the modeling approach just described. The essential characteristics of each model are shown in table 2. For Model 1, we investigate the effect of adding more historical regular season box score statistics at the expense of ranking data, as explained in Section 2.2. Given that we have limited prior information on which covariates are important for the prediction, we impose a very weakly informative prior on all regression coefficients and the intercept. For Model 2, we use only data from the 2016-2021 seasons. As explained previously, since the nature of the game has changed drastically over time, we hypothesized that this may have a significant effect on the model built. Additionally, using less historical data has the advantage of allowing us to use ordinal ranking

data, besides team seeds of course. Again, for Model 2, we impose a very weakly informative prior on all regression coefficients and intercept. Finally, Model 3 is similar to Model 2, except we impose a Normal(0, 1) prior on the coefficients corresponding to SFX and MAS rankings to induce shrinkage, since we believe the other two ranking systems are of better quality. This will allow us to investigate the sensitivity of our model to prior specification for ordinal rankings.

Model 1	**Model 2**	**Model 3**
2003-2021 Seasons	2016-2021 Seasons	Similar to Model 2
No Ordinal Rankings	Ordinal Rankings: BWE, POM, MAS, SFX	Generic weakly informative prior on SFX and MAS: Normal(0, 1)
Very weakly informative prior on all regression coefficients and intercept: Normal(0, 10)	Very weakly informative prior on all regression coefficients and intercept: Normal(0, 10)	Very weakly informative prior on remaining regression coefficients and intercept: Normal(0, 10)

Table 2: Characteristics of the three different models we built

4 Diagnostics

To determine which of our three models would serve as the best predictor, we performed cross-validation. For Model 1, we divide the dataset into 6 folds, where each fold is composed of data from three consecutive seasons, and perform 6-fold cross-validation. For Models 2 and 3, we perform 5-fold cross-validation, where each fold consists of data from a single season. The results are listed in table 3.

Model	**Cross validation accuracy (%)**
1	63.5
2	66.4
3	66.0

Table 3: K-fold cross validation accuracy for each model

To compute the accuracy, we repeatedly draw samples of each of the model's parameters from their posterior distributions, compute μ_{ij} from Equation 1 for each game in the validation set, and then sample y_ij as described in Equation 2 for each game. Accuracy is then computed by counting the number of correct predictions made by the model over the total number of predictions made. This process is done repeatedly as different parameter values are sampled from their joint posterior distribution, so that in the end we obtain an MCMC accuracy estimate, which is just the mean of the posterior accuracies generated in the MCMC process. In practice, this is done using the Stan function "gqs", which will compute the posterior distributions of the quantities specified in the "generated_quantities" block of a Stan model (see the "generated_quantities" block of our Stan model in the Appendix, Section 8.3

for more details).

We observe that Model 2 outperforms the other two models, with a 5-fold cross validation accuracy of 66.4%. We thus proceed to train a new model which uses the specification of model 2, trained on the whole 2016-2021 dataset. We check this model for convergence by examining traceplots for all the model's parameters. This is shown in figure 2. We observe that all parameters converge.

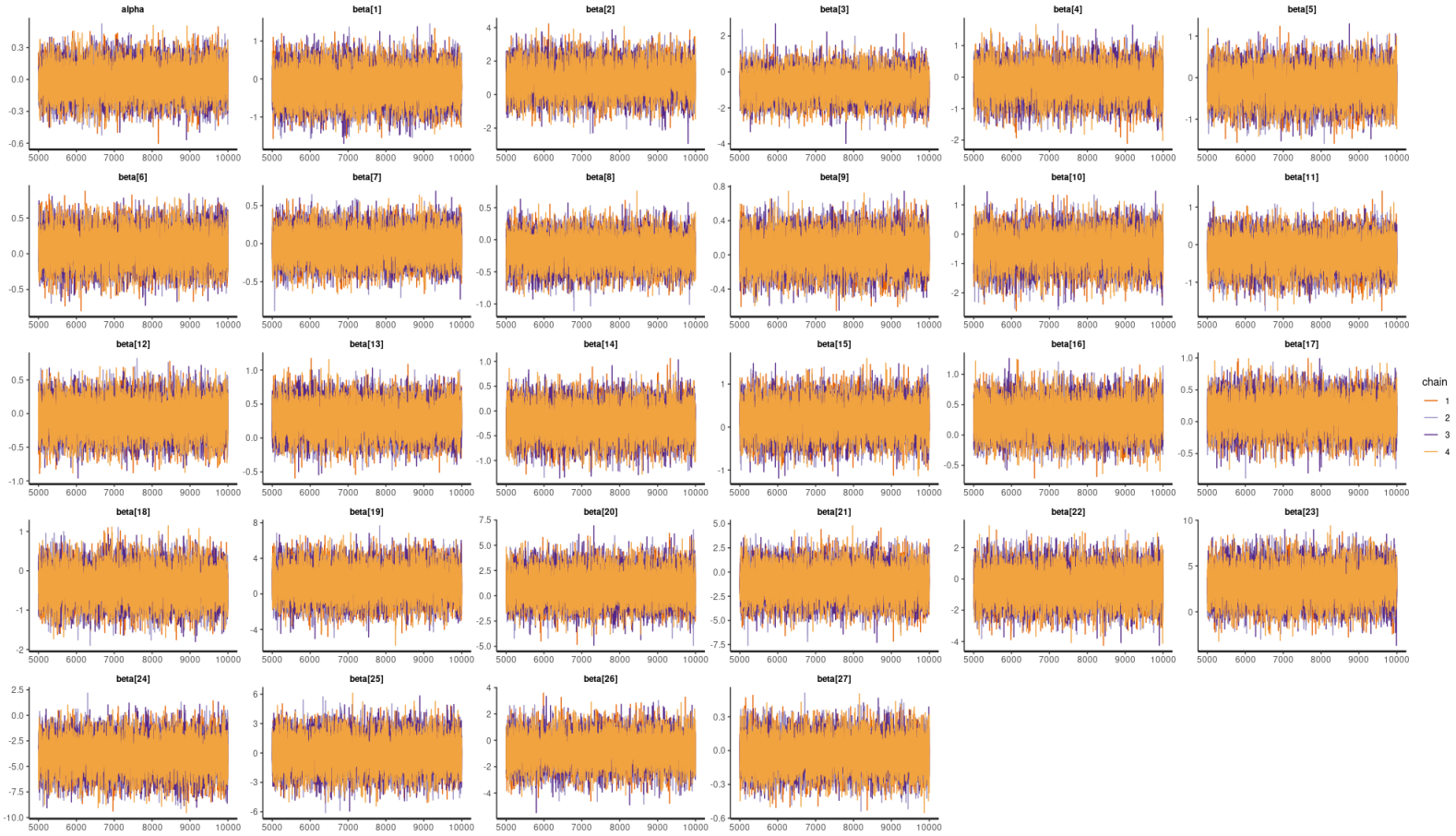

Figure 2: Traceplots for the best model's parameters. Barring the intercept term alpha, the order in which the parameters are listed corresponds to the order in which the variables are listed in table 1

Having checked convergence, we then use our model to make predictions for the games of the 2022 season. We scale all variables for the test set using the training set variables' mean and standard deviations. We obtain an accuracy of 58%. The accuracy for the test set is computed in the same way as it was computed in the case of cross-validation, described above.

5 Model interpretation

We next investigate the 80% and 90% credible intervals for model parameters. These are shown in figure 3. We note that except for beta[23] and beta[24], which correspond to the covariates POMMeanDiff and BWEMeanDiff, the 80% (and hence the 90%) credible intervals of all other parameters contain 0.

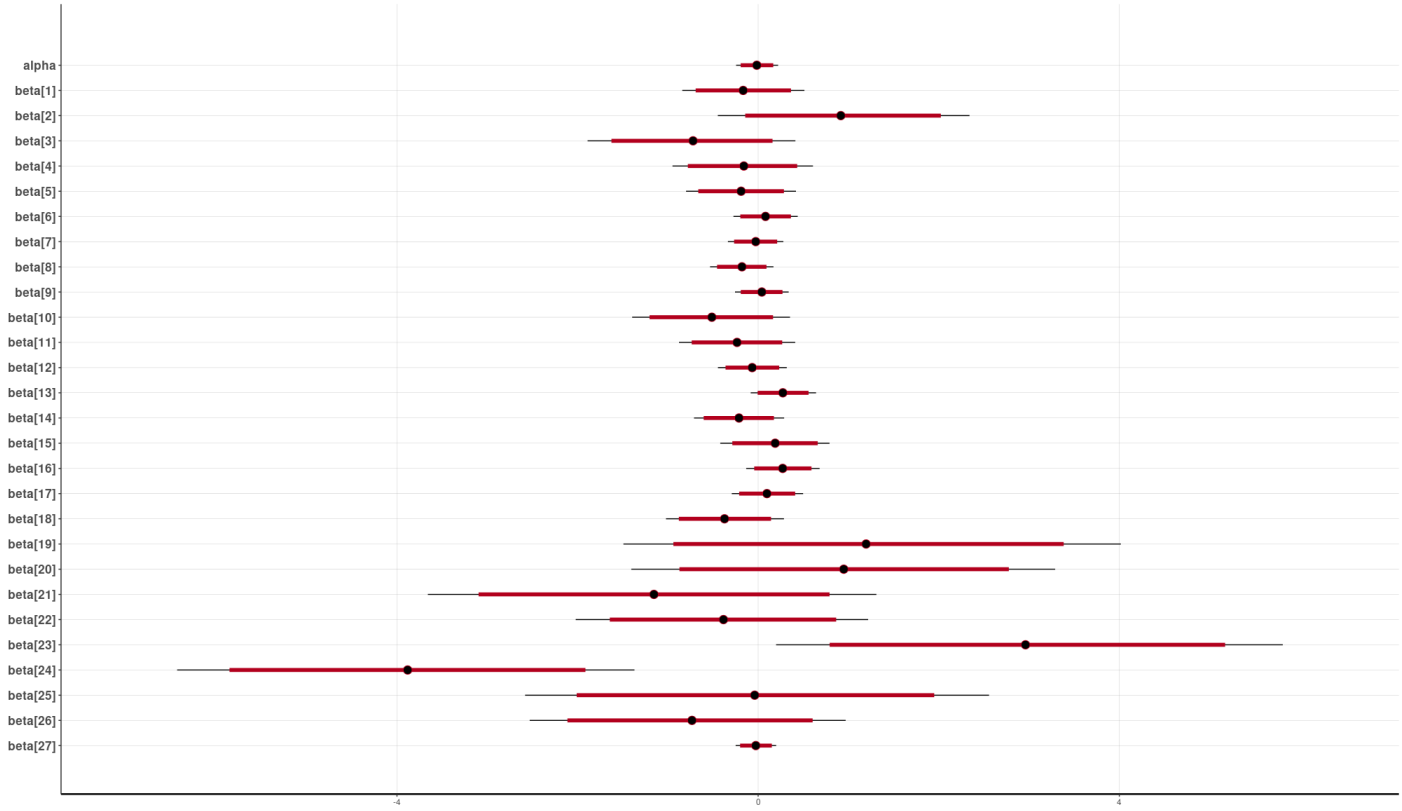

Figure 3: The 80% and 90% credible intervals for the parameters of our best model, based on the approach of Model 2. Barring the intercept term alpha, the order in which the parameters are listed corresponds to the order in which the variables are listed in table 1

6 Discussion

We first note that the cross validation accuracy results listed in table 3 indicate that our model is not really sensitive to prior specification. Specifically, we observe that the cross validation accuracy for Model 2 and Model 3 only differ by 0.4%. Model 2 only slightly outperforms Model 3. Furthermore, the results in Section 5 indicate that the most predictive covariates for the winner of a March Madness game are the difference of their average Pomeroy and B Wilson Empirical rankings. In particular, holding all other predictors constant, the expected change in the log odds ratio of winning a game when the POMMeanDiff between the first listed team and the second listed team increases by one standard deviation is around -4. Similarly, holding all other predictors constant, the expected change in the log odds ratio of winning a game when the BWEMeanDiff between the first listed team and the second listed team increases by one standard deviation is around positive 3. We note that the coefficient for POMMeanDiff makes sense; as the difference in the average Pomeroy ranking between the first team and the second team becomes more positive i.e the ordinal rank of the first team "decreases" (i.e increases in magnitude) or the ordinal rank of the second team "increases" (i.e decreases in magnitude) - and hence the difference between their average rankings becomes more positive - the log of the odds of winning decreases. On the other hand, the coefficient for BWEMeanDiff does not make a lot sense,

because its interpretation is the opposite of that of the average Pomeroy difference coefficient. Given the high quality of the BWE ranking system, we suspect that it our model that is flawed, rather than the ranking system being not credible. This is especially true given the mediocre accuracy results obtained by our model.

7 Conclusion

There were a plethora of challenges associated with our work which may have contributed to the mediocre results we obtained, especially with our test set. There are over 350 teams in Division I college basketball and only 68 make the tournament each year. On top of that, the rosters for each team vary greatly for each year, making year to year trends hard to predict. Additionally, college student athletes are amateurs and not professionals, making it hard to predict consistency. Unfortunately, the 2022 tournament was even more unpredictable than usual since there were many unexpected upsets including a 15 seed making the quarter finals for the first time, an 8 seed making the championship for the third time, and a few other double-digit seeds making it past the second round. It may be that logistic regression is not powerful enough to model the "madness" of March Madness. As an extension to the current work, we suggest testing more powerful models like Random Forest or XGBoost. Overall, we determined that it is very difficult to predict the outcomes of games in the NCAA tournament. Perfect accuracy may be nearly impossible to obtain. Indeed, Warren Buffett offers $1 billion to the first person who can accurately predict each game in the tournament, hinting at the near impossibility of the task given current statistical methods.

References

Brown, B. (2019). Predictive analytics for college basketball: Using logistic regression for determining the outcome of a game regression for determining the outcome of a game. https://scholars.unh.edu/cgi/viewcontent.cgi?article=1421&context=honors

Nelson, B. (2012). Modeling the ncaa tournament through bayesian logistic regression. https://dsc.duq.edu/cgi/viewcontent.cgi?article=1986&context=etd

8 Appendix

8.1 Header

```
library(plyr)
library(tidyverse)
library(rstan)
options(max.print = 100000)
options(mc.cores = parallel::detectCores())
set.seed(42)
```

8.2 Data Prep

```
#### Data Prep ####
marchMadnessAllGames <- read_csv(
                          "MDataFiles_Stage2/MNCAATourneyCompactResults.csv")
regSeason <- read_csv("MDataFiles_Stage2/MRegularSeasonDetailedResults.csv")
seeds <- read_csv("MDataFiles_Stage2/MNCAATourneySeeds.csv")
ranks <- read_csv("MDataFiles_Stage2/MMasseyOrdinals_thruDay128.csv")
teamIDToTeamName <- read_csv("MDataFiles_Stage2/MTeams.csv")

# Specify list of desired seasons for teams
# returns MM teams participating in MM seasons specified by "seasons"
getMMTeams <- function(seasons, marchMadnessGames) {
  marchMadnessGames <- marchMadnessGames %>%
                       filter(Season %in% seasons)
  wteams <- marchMadnessGames %>%
            select(WTeamID)
  lteams <- marchMadnessGames %>%
            select(LTeamID)
  marchMadnessTeams <- c(unlist(wteams$WTeamID), unlist(lteams$LTeamID))
  marchMadnessTeams <- unique(marchMadnessTeams)
  return(marchMadnessTeams)
}

# Must pass in all seasons of interest
getAvailableRankingSystems <- function(seasons, ranks, marchMadnessGames) {
  rankingSystems <- unique(ranks$SystemName)
  marchMadnessTeams <- getMMTeams(seasons, marchMadnessGames)
  ranks <- ranks %>%
           filter(Season %in% seasons, TeamID %in% marchMadnessTeams)

  availableRankingsAggregated <- list()
  for (team in marchMadnessTeams) {
    for (season in unique(ranks$Season)) {
      availableRankings <- ranks %>%
                           filter(TeamID == team, Season == season) %>%
                           select(SystemName) %>%
                           unique()
      availableRankingsAggregated <- availableRankingsAggregated %>%
                                     append(availableRankings)
    }
  }
  return(Reduce(intersect, availableRankingsAggregated))
}

# UNCOMMENT and run if want to check for different seasons
# availableRankingSystems <- getAvailableRankingSystems(c(2016:2022),
#                                                       ranks,
#                                                       marchMadnessAllGames)
# availableRankingSystems <- getAvailableRankingSystems(c(2003:2022),
#                                                       ranks,
#                                                       marchMadnessAllGames)

getSeedsPerSeason <- function(season, seeds) {
  seeds <- seeds %>%
           filter(Season == season)
  seeds <- seeds %>%
           mutate(Seed = str_sub(Seed, 2, 3))
  seeds$Seed <- as.integer(seeds$Seed)
  return(seeds)
}

# returns average season rank in each desired ranking system and last ranking
# right before beginning of March Madness for each team (rankingDayNum 133
# is not always available before the start of the tournament, so we use Day 128
# instead)
getRanksPerSeason <- function(season, desiredRankingSystems, ranks,
                              teams) {
  preMMranks <- ranks %>%
                filter(Season == season, SystemName %in% desiredRankingSystems,
                       TeamID %in% teams, RankingDayNum == 128) %>%
```

```
                select(-one_of("Season", "RankingDayNum")) %>%
                pivot_wider(names_from = SystemName, values_from = OrdinalRank)

  averageRegSeasonranks <- ranks %>%
                           filter(Season == season, SystemName %in%
                                  desiredRankingSystems, TeamID %in% teams) %>%
                           group_by(TeamID, SystemName) %>%
                           summarize(mean(OrdinalRank)) %>%
                           pivot_wider(names_from = SystemName, values_from =
                                       "mean(OrdinalRank)")
  if (nrow(averageRegSeasonranks) * ncol(averageRegSeasonranks) != 0) {
    colnames(averageRegSeasonranks) <- c("TeamID", paste(desiredRankingSystems,
                                         "Mean", sep=''))
  }
  ranks <- preMMranks %>%
           inner_join(averageRegSeasonranks, by="TeamID")
  return(ranks)
}

getMMTeamsPreTourneyStatsPerSeason <- function(season, regSeason,
                                               marchMadnessAllGames,
                                               seeds, ranks,
                                               desiredRankingSystems) {
  regSeason <- regSeason %>%
               filter(Season == season)
  seeds <- getSeedsPerSeason(season, seeds)
  MMTeams <- getMMTeams(season, marchMadnessAllGames)
  ranks <- getRanksPerSeason(season, desiredRankingSystems, ranks, MMTeams)
  regSeason <- regSeason %>%
               filter(WTeamID %in% MMTeams | LTeamID %in% MMTeams)
  dup1 <- regSeason %>%
          mutate(TeamID=WTeamID)
  dup2 <- regSeason %>%
          mutate(TeamID=LTeamID)
  regSeason <- bind_rows(dup1, dup2)
  MMTeamsRegSeasonStats <- tibble()
  for (team in MMTeams) {
    teamTibble <- regSeason %>%
                  filter(TeamID == team)
    winTibble <- teamTibble %>%
                 filter(WTeamID == team)
    winTibble <- winTibble %>%
                 select(-one_of("DayNum", "WPF", "LPF", "NumOT", "WLoc",
                                "TeamID"))
    colnames(winTibble) <- c("Season", "currTeamID", "currScore", "otherTeamID",
                             "otherScore", "currFGM", "currFGA", "currFGM3",
                             "currFGA3", "currFTM", "currFTA", "currOR",
                             "currDR", "currAst", "currTO", "currStl",
                             "currBlk", "otherFGM", "otherFGA", "otherFGM3",
                             "otherFGA3", "otherFTM", "otherFTA", "otherOR",
                             "otherDR", "otherAst", "otherTO", "otherStl",
                             "otherBlk")
    loseTibble <- teamTibble %>%
                  filter(LTeamID == team)
    loseTibble <- loseTibble %>%
                  select(-one_of("DayNum", "WPF", "LPF", "NumOT", "WLoc",
                                 "TeamID"))
    colnames(loseTibble) <- c("Season", "otherTeamID", "otherScore",
                              "currTeamID", "currScore", "otherFGM", "otherFGA",
                              "otherFGM3", "otherFGA3", "otherFTM", "otherFTA",
                              "otherOR", "otherDR", "otherAst", "otherTO",
                              "otherStl", "otherBlk", "currFGM", "currFGA",
                              "currFGM3", "currFGA3", "currFTM", "currFTA",
                              "currOR", "currDR", "currAst", "currTO", "currStl",
                              "currBlk")
    currTeamTibble <- bind_rows(winTibble, loseTibble)
    currTeamTibble <- currTeamTibble %>%
                      summarize(TeamID=unique(currTeamID),
                                winPct = nrow(winTibble) / (nrow(winTibble) +
                                                            nrow(loseTibble)),
                                lossPct = nrow(loseTibble)/ (nrow(winTibble) +
                                                             nrow(loseTibble)),
                                currScoreAvg = mean(currScore),
                                otherScoreAvg = mean(otherScore),
                                currFGPct = mean(currFGM)/mean(currFGA),
                                otherFGPct = mean(otherFGM)/mean(otherFGA),
                                currFG3Pct = mean(currFGM3)/mean(currFGA3),
                                otherFG3Pct = mean(otherFGM3)/mean(otherFGA3),
                                currFTPct = mean(currFTM)/mean(currFTA),
                                otherFTPct = mean(otherFTM)/mean(otherFTA),
                                currRebAvg = (mean(currOR) + mean(currDR)),
                                otherRebAvg = (mean(otherOR) + mean(otherDR)),
                                currAstAvg = mean(currAst),
                                otherAstAvg = mean(otherAst),
                                currTOAvg = mean(currTO),
                                otherTOAvg = mean(otherTO),
                                currDefAvg = (mean(currStl) + mean(currBlk)),
                                otherDefAvg = (mean(otherStl) + mean(otherBlk))
```

```
                                )
    MMTeamsRegSeasonStats <- MMTeamsRegSeasonStats %>%
                             bind_rows(currTeamTibble)
  }

  MMTeamsRegSeasonStats <- MMTeamsRegSeasonStats %>%
                           inner_join(seeds, by="TeamID")

  if (nrow(ranks) * ncol(ranks) != 0) {
    MMTeamsRegSeasonStats <- MMTeamsRegSeasonStats %>%
                             inner_join(ranks, by="TeamID")
  }
  return(MMTeamsRegSeasonStats)
}

computeStatDiff <- function(df1, df2) {
  diff <- df1 - df2
  colnames(diff) <- paste(colnames(diff), "Diff", sep="")
  return(diff)
}

computeSelectStatsAddition <- function(df1, df2, selectColumns) {
  df1 <- df1 %>%
         select(selectColumns)

  df2 <- df2 %>%
         select(selectColumns)
  addition <- df1 + df2
  colnames(addition) <- paste(colnames(addition), "Add", sep="")
  return(addition)
}

# season must match regSeasonStats season
getMarchMadnessMatchupsPerSeason <- function(season, MMTeamsRegSeasonStats,
                                             marchMadnessAllGames) {
  MMTeamsRegSeasonStats <- MMTeamsRegSeasonStats %>% select(-Season)
  marchMadnessGames <- marchMadnessAllGames %>%
                       filter(Season == season) %>%
                       sample_frac(1L)
  num_row <- nrow(marchMadnessGames)
  marchMadnessWins <- marchMadnessGames %>%
                      slice(1:floor(num_row/2)) %>%
                      select(WTeamID, LTeamID) %>%
                      mutate(firstTeamWinStatus = 1)
  colnames(marchMadnessWins)[1] <- "TeamID"
  marchMadnessLosses <- marchMadnessGames %>%
                        slice(floor(num_row/2)+1:num_row) %>%
                        select(LTeamID, WTeamID) %>%
                        mutate(firstTeamWinStatus = 0)
  colnames(marchMadnessLosses)[1] <- "TeamID"

  getTeamStatsByKey <- function(df, regSeasonStats, Key) {
    marchMadnessFirstTeamStats <- df %>%
                                  left_join(regSeasonStats,
                                            by=Key) %>%
                                   select(-c(1,2,3))
    return(marchMadnessFirstTeamStats)
  }

  marchMadnessWinsWinTeamStats <- getTeamStatsByKey(marchMadnessWins,
                                                    MMTeamsRegSeasonStats,
                                                    "TeamID")
  marchMadnessWinsLoseTeamStats <- getTeamStatsByKey(marchMadnessWins,
                                                     MMTeamsRegSeasonStats,
                                                     c("LTeamID" = "TeamID"))
  marchMadnessWinsStatDiff <- computeStatDiff(marchMadnessWinsWinTeamStats,
                                              marchMadnessWinsLoseTeamStats) %>%
                              select(-lossPctDiff)
  marchMadnessWinsStatAdd <- computeSelectStatsAddition(
                                                marchMadnessWinsWinTeamStats,
                                                marchMadnessWinsLoseTeamStats,
                                                c("Seed"))
  marchMadnessWins <- marchMadnessWins %>%
                      bind_cols(marchMadnessWinsStatDiff) %>%
                      bind_cols(marchMadnessWinsStatAdd)


  marchMadnessLossesLoseTeamStats <- getTeamStatsByKey(marchMadnessLosses,
                                                       MMTeamsRegSeasonStats,
                                                       "TeamID")
  marchMadnessLossesWinTeamStats <- getTeamStatsByKey(marchMadnessLosses,
                                                      MMTeamsRegSeasonStats,
                                                      c("WTeamID" = "TeamID"))
  marchMadnessLossesStatDiff <- computeStatDiff(marchMadnessLossesLoseTeamStats,
                                                marchMadnessLossesWinTeamStats) %>%
                                select(-lossPctDiff)
  marchMadnessLossesStatAdd <- computeSelectStatsAddition(
                                                  marchMadnessLossesLoseTeamStats,
```

```
                                                marchMadnessLossesWinTeamStats,
                                                c("Seed"))
  marchMadnessLosses <- marchMadnessLosses %>%
                        bind_cols(marchMadnessLossesStatDiff) %>%
                        bind_cols(marchMadnessLossesStatAdd)
  colnames(marchMadnessLosses)[2] <- "otherTeamID"
  colnames(marchMadnessWins)[2] <- "otherTeamID"
  clean_data <- marchMadnessLosses %>%
                bind_rows(marchMadnessWins) %>%
                sample_frac(1L)

  Season <- as_tibble(rep(season, nrow(clean_data)))
  colnames(Season) <- "Season"
  clean_data <- bind_cols(Season, clean_data)
  return(clean_data)
}

createDataset <- function(seasons, regSeason, marchMadnessAllGames, seeds,
                          ranks, desiredRankingSystems, teamIDToTeamName) {
  MMTeamsPreTourneyStats <- list()
  MMMatchups <- list()
  for (i in 1:length(seasons)) {
    season <- seasons[i]
    MMTeamsPreTourneyStats <- append(MMTeamsPreTourneyStats,
                                     list(getMMTeamsPreTourneyStatsPerSeason(
                                     season,
                                     regSeason,
                                     marchMadnessAllGames,
                                     seeds,
                                     ranks,
                                     desiredRankingSystems)))
    MMTeamsPreTourneyStats[[i]] <- MMTeamsPreTourneyStats[[i]] %>%
                                   select(Season, everything())
    MMMatchups <- append(MMMatchups, list(getMarchMadnessMatchupsPerSeason(
                                                season,
                                                MMTeamsPreTourneyStats[[i]],
                                                marchMadnessAllGames)))
  }
  MMDataset <- do.call(rbind, MMMatchups)

  MMDataset$TeamID <- (MMDataset %>% left_join(
                       teamIDToTeamName, on="TeamID"))$TeamName

  colnames(MMDataset)[2] <- "firstTeamName"
  colnames(MMDataset)[3] <- "otherTeamName"

  MMDataset$otherTeamName <- (MMDataset %>% left_join(
    teamIDToTeamName, by=c("otherTeamName" = "TeamID")))$TeamName

  return(MMDataset)
}

MM2016To2021Dataset <- createDataset(c(2016, 2017, 2018, 2019, 2021), regSeason,
                                     marchMadnessAllGames, seeds, ranks,
                                     c("POM", "BWE", "MAS", "SFX"),
                                     teamIDToTeamName)
# Descriptive statistics
MM2016To2021DatasetMeans <- MM2016To2021Dataset %>%
                            select(-one_of("Season", "firstTeamName",
                                           "otherTeamName",
                                           "firstTeamWinStatus")) %>%
                            summarise(across(everything(), mean))

MM2016To2021DatasetSds <- MM2016To2021Dataset %>%
                          select(-one_of("Season", "firstTeamName",
                                         "otherTeamName",
                                         "firstTeamWinStatus")) %>%
                          summarise(across(everything(), sd))

seasons <- c(2003:2021)
seasons <- seasons[seasons != 2020]
MM2003To2021Dataset <- createDataset(seasons, regSeason,
                                     marchMadnessAllGames, seeds, ranks,
                                     c(), teamIDToTeamName)
```

8.3 Stan Modeling

```
#### Modeling ####
createModel <- function() {
  if (!file.exists("logisticRegressionModel.stan")) {
    write("
    data {
      int<lower=0> N;   // number of data items
      int<lower=0> K;   // number of predictors
      matrix[N, K] X;   // predictor matrix
```

```
      int y[N];         // outcome vector
      int relativePrior;
      int shrinkageIdx[4];
      int noShrinkageIdx[23];
    }
    parameters {
      real alpha;       // intercept
      vector[K] beta;   // coefficients for predictors
    }
    model {
      // Priors:
      alpha ~ normal(0, 10);
      if (relativePrior) {
        for (k in shrinkageIdx) {
          beta[k] ~ normal(0, 1);
        }
        for (k in noShrinkageIdx) {
          beta[k] ~ normal(0, 10);
        }
      }
      else {
        beta ~ normal(0, 10);
      }
      // Likelihood
      y ~ bernoulli_logit(alpha + X * beta);
    }
    generated quantities {
      vector[N] y_preds;
      real correct = 0;
      real accuracy;
      for (n in 1:N) {
        y_preds[n] = bernoulli_logit_rng(alpha + X[n] * beta);
        correct += logical_eq(y_preds[n], y[n]);
      }
      accuracy = correct / N;
    }
    ", "logisticRegressionModel.stan")
  }
}

createModel()
```

8.4 Diagnostics

8.4.1 Cross Validation

```
#### Diagnostics ####
scaleTrainData <- function(data, exclude_columns) {
  covariates <- data %>% select(-one_of(c(unlist(exclude_columns))))
  means <- c()
  stdevs <- c()
  for (column in colnames(covariates)) {
    mean <- mean(covariates[[column]])
    means <- append(means, mean)
    stdev <- sd(covariates[[column]])
    stdevs <- append(stdevs, stdev)
    covariates[column] <- scale(covariates[column])
  }
  return(list(covariates, means, stdevs))
}

scaleTestData <- function(data, means, stdevs, exclude_columns) {
  covariates <- data %>% select(-one_of(c(unlist(exclude_columns))))
  for (i in 1:length(colnames(covariates))) {
    column <- colnames(covariates)[i]
    covariates[column] <- (covariates[column] - means[i]) / stdevs[i]
  }
  return(covariates)
}
createStanDataLists <- function(dataset, val_fold, relativePriors,
                                 outcome_column_name) {
  outcome <- dataset[outcome_column_name]
  outcome_train <- (dataset[dataset$fold != val_fold,])[outcome_column_name]
  outcome_val <- (dataset[dataset$fold == val_fold,])[outcome_column_name]
  dataset_train <- dataset[dataset$fold != val_fold,]
  dataset_val <- dataset[dataset$fold == val_fold,]
  exclude_columns <- c("firstTeamName", "otherTeamName", outcome_column_name,
                       "fold")
  scaleList <- scaleTrainData(dataset_train, exclude_columns)
  covariates_train <- scaleList[[1]]
  means <- scaleList[[2]]
  stdevs <- scaleList[[3]]
  covariates_val <- scaleTestData(dataset_val, means, stdevs, exclude_columns)
  noShrinkageIdx <- c(1:27)
  shrinkageIdx <-  c(20, 22, 25, 26)
```

```
  noShrinkageIdx <- noShrinkageIdx[!(noShrinkageIdx %in% shrinkageIdx)]
  train_list <- list(X=covariates_train, y=array(unlist(outcome_train)),
                     K=ncol(covariates_train),
                     N=nrow(covariates_train),
                     relativePrior=relativePriors,
                     shrinkageIdx=shrinkageIdx,
                     noShrinkageIdx=noShrinkageIdx
                     )

  val_list <- list(X=covariates_val, y=array(unlist(outcome_val)),
                   K=ncol(covariates_val),
                   N=nrow(covariates_val),
                   relativePrior=relativePriors,
                   shrinkageIdx=shrinkageIdx,
                   noShrinkageIdx=noShrinkageIdx
                   )
  return(list(train_list, val_list))
}

CV <- function(dataset, relativePriors) {
  if (length(unique(dataset$Season)) == 5) {
    dataset$fold <- dataset$Season - 2016
    dataset$fold[dataset$fold == 5] <- 4
  }
  else {
   dataset$fold <- mapvalues(dataset$Season,
                         from=c("2003","2004","2005", "2006", "2007", "2008",
                                "2009", "2010", "2011", "2012", "2013", "2014",
                                "2015", "2016", "2017", "2018", "2019", "2021"),
                         to=c(rep(0, 3), rep(1, 3),
                              rep(2, 3), rep(3, 3),
                              rep(4, 3), rep(5, 3)))
  }
  dataset <- dataset %>% select(-Season)
  to_return <- list()
  fold_fits <- list()
  valPerformances <- list()
  for(fold in 0:(length(unique(dataset$fold))-1)) {
    standDataLists <- createStanDataLists(dataset, fold, relativePriors,
                                                          "firstTeamWinStatus")
    trainList <- standDataLists[[1]]
    valList <- standDataLists[[2]]
    fit <-  stan(file = "logisticRegressionModel.stan", data = trainList,
                 iter=10000, seed=42)
    fold_fits <- append(fold_fits, fit)
    valPerformances <- append(valPerformances, gqs(fit@stanmodel,
                                                   draws = as.matrix(fit),
                                                   data = valList))
  }
  to_return <- list(fold_fits, valPerformances)
  return(to_return)
}

fits20162021NoRelativeRankingPrior <- CV(MM2016To2021Dataset, 0)
avgAccuracy20162021NoRelativeRankingPrior <- mean(c(66, 67, 66, 66, 67))
fits20162021WithRelativeRankingPrior <- CV(MM2016To2021Dataset, 1)
avgAccuracy20162021WithRelativeRankingPrior <- mean(c(66, 66, 66, 65, 67))
fits20032021NoRanking <- CV(MM2003To2021Dataset, 0)
avgAccuracy20032021NoRanking <- mean(c(63, 63, 64, 64, 63, 64))
```

8.4.2 Train best model

```
# Train best model
MM2016To2021DatasetDup <- MM2016To2021Dataset
MM2016To2021DatasetDup$fold <- 0
MM2016To2021DatasetDup <- MM2016To2021DatasetDup %>% select(-Season)
outcome <- MM2016To2021DatasetDup["firstTeamWinStatus"]
exclude_columns <- c("firstTeamName", "otherTeamName", "firstTeamWinStatus",
                     "fold")
scaleList <- scaleTrainData(MM2016To2021DatasetDup, exclude_columns)
covariates <- scaleList[[1]]
means <- scaleList[[2]]
stdevs <- scaleList[[3]]
noShrinkageIdx <- c(1:27)
shrinkageIdx <-  c(20, 22, 25, 26)
noShrinkageIdx <- noShrinkageIdx[!(noShrinkageIdx %in% shrinkageIdx)]
train_list <- list(X=covariates, y=array(unlist(outcome)),
                   K=ncol(covariates),
                   N=nrow(covariates),
                   relativePrior=0,
                   shrinkageIdx=shrinkageIdx,
                   noShrinkageIdx=noShrinkageIdx
)
fit <-  stan(file = "logisticRegressionModel.stan", data = train_list,
             iter=10000, seed=42)
```

```
pars <- c('alpha', 'beta[1]', 'beta[2]', 'beta[3]', 'beta[4]', 'beta[5]',
          'beta[6]', 'beta[7]', 'beta[8]', 'beta[9]', 'beta[10]', 'beta[11]',
          'beta[12]', 'beta[13]', 'beta[14]', 'beta[15]', 'beta[16]', 'beta[17]'
          , 'beta[18]', 'beta[19]', 'beta[20]', 'beta[21]', 'beta[22]',
          'beta[23]', 'beta[24]', 'beta[25]', 'beta[26]', 'beta[27]')
traceplot(fit, pars=pars)
covariates <- colnames(MM2016To2021DatasetDup)[4:30]
```

8.4.3 Test set Performance

```
# Model Test Set Performance
best_model <- fit
set.seed(42)
testSet <- createDataset(2022, regSeason,
                         marchMadnessAllGames, seeds, ranks,
                         c("POM", "BWE", "MAS", "SFX"), teamIDToTeamName)
testSet <- testSet %>% select(-Season)
outcome <- testSet["firstTeamWinStatus"]
testSet <- scaleTestData(testSet, means, stdevs, exclude_columns)
testList <- list(X=testSet, y=array(unlist(outcome)),
                 K=ncol(testSet),
                 N=nrow(testSet),
                 relativePrior=0,
                 shrinkageIdx=shrinkageIdx,
                 noShrinkageIdx=noShrinkageIdx
)
gqs(fit@stanmodel, draws=as.matrix(best_model), data=testList, seed=42)
```

8.5 Model Interpretation

```
#### Model interpretation ####
plot(fit, pars=pars, outer_level=0.90)
```