Comprehensive Examination Responses

Jake Thompson

2017-04-01

Contents

1 Introduction

This document is a compressed version of a more detailed document available online. Specifically, this document aims to answers the specific questions for my written comprehensive examination. As such, this document may reference parts of the more complete document, but will also be comprehensive in containing all information needed to adequately address the questions.

2 Gather data

The first step to this analysis is to gather the data for the English Premier League. For this analysis, I will scrape the data from ESPN. To scrape the data, I define two functions, `scrape_league` and `scrape_team`. The `scrape_league` function takes in an ESPN URL for a given league and returns the league standings, as well as a URL for the homepage each club in the league. The `scrape_team` function takes the club specific URLs and scrapes the match schedule and results from each club. Both functions use a combination of the **purrr** (Wickham, 2016a), **rvest** (Wickham, 2016b), **dplyr** (Wickham & Francois, 2016), and **lubridate** (Grolemund, Spinu, & Wickham, 2016) packages to retrieve and format this information.

```
library(dplyr)
library(lubridate)
library(purrr)
library(rvest)
```

Table 1: English Premier League Games

date	home	away	h_goals	a_goals	competition
2016-08-13	Middlesbrough	Stoke City	1	1	Prem
2016-08-13	Burnley	Swansea City	0	1	Prem
2016-08-13	Crystal Palace	West Bromwich Albion	0	1	Prem
2016-08-13	Everton	Tottenham Hotspur	1	1	Prem
2016-08-13	Hull City	Leicester City	2	1	Prem
2016-08-13	Manchester City	Sunderland	2	1	Prem

```
safe_read_html <- safely(read_html)
epl <- scrape_league("http://www.espnfc.us/english-premier-league/23/table")
epl_games <- map2_df(.x = epl$club_url, .y = epl$club, .f = scrape_team)
```

After the game data is scraped, I filter to only include games within the Premier League, and do some cleaning (e.g., replace ESPN abbreviations with the real club name). The beginning of the data set can be seen in Table 1.

3 Estimate the models

I estimate the bivariate Poisson and the game random intercept model using Stan (Guo, Gabry, & Goodrich, 2016; Stan Development Team, 2016b). The Stan code for the bivariate Poisson model and the game random intercept model can be seen in Appendix **??** and **??** respectively. For each model, I estimate the offensive and defensive parameters for each team as random effects with a mean of 0 and an estimated variance. I also use a non-centered parameterization. This means that I define dummy parameters on a $\mathcal{N}(0, 1)$ scale, and then transform the values using the estimated random effect variances. This improves the efficiency of the estimator, and prevents the sampler from getting stuck when the estimated variances are small (Betancourt, 2016; Stan Development Team, 2016a). More details about each model can be seen in Section 2.

Because Stan can only take in numerical data, I first have to assign each team a numerical code to track which offensive and defensive effects belong to each team. I can then format the data for use in **rstan**, and estimate each model.

```
library(rstan)

stan_data <- list(
  num_clubs = nrow(team_counts),
  num_games = nrow(fit_data),
  home = fit_data$home_code,
  away = fit_data$away_code,
  h_goals = fit_data$h_goals,
  a_goals = fit_data$a_goals,
  homeg = fit_data$home_game
)

bip_stanfit <- sampling(bivpois, data = stan_data, chains = 3, iter = 7000,
  warmup = 2000, init = "random", thin = 5, cores = 3, algorithm = "NUTS",
  seed = 32011, control = list(adapt_delta = 0.99, max_treedepth = 15))

gri_stanfit <- sampling(gri, data = stan_data, chains = 3, iter = 7000,
  warmup = 2000, init = "random", thin = 5, cores = 3, algorithm = "NUTS",
```

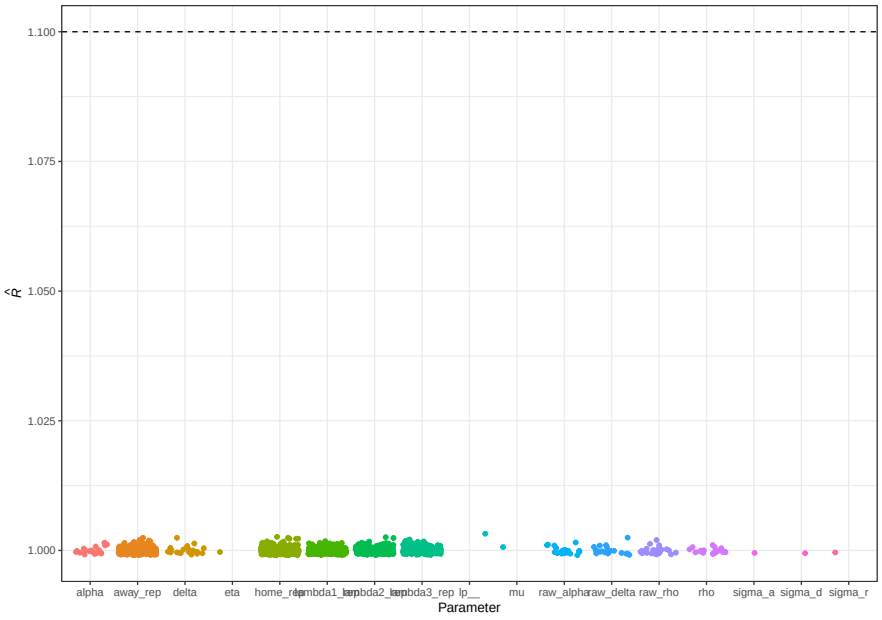

Figure 1: Rhat statistics for the estimated bivariate Poisson model.

```
  seed = 32011, control = list(adapt_delta = 0.99, max_treedepth = 15))
```

4 Results

4.1 Model convergence

Before examing the model results, it is important to ensure that the models have converged. Figures 1 and 2 show the $\hat{R}$ statistics for each model. If all values are below 1.1, then we can infer convergence of the chains. Both Figures 1 and 2 indicate that both models adequately converged.

4.2 Ratings comparison

To get ratings for each team, I will calculate the expected goals scored and conceded for each team against an average team at a neutral location. I can then compute and expected score margin, which can be used to rank the overall ability of each team. As can be seen in Table 2, the offensive and defensive effects estimated in the bivariate Poisson model (BIP) are smaller in magnitude than those of the game random intercept model (GRI). Figure 3 shows a comparison of the net ratings between the models. Here we can see that although the scale is smaller for the BIP model, there is a strong relationship between the two.

We can also compare the net ratings to the league standings by looking at the correlations between the net ratings and the points each team has scored in the league thus far. Table 3 shows that the GRI model ratings are more highly correlated with league standings. This provides preliminary evidence that the GRI model may be better suited to the data. However, a more thorough examination of model fit is needed to confirm this.

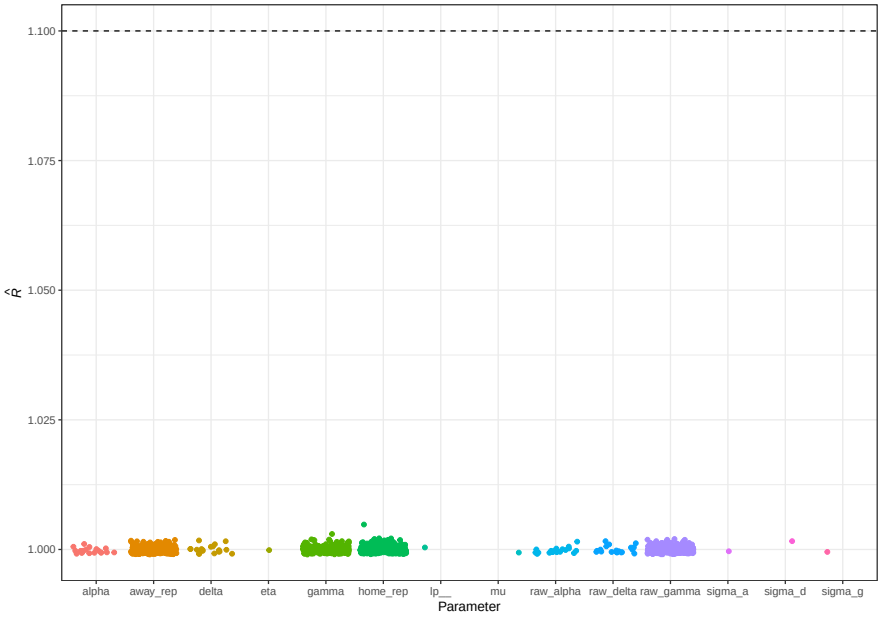

Figure 2: Rhat statistics for the estimated game random intercept model.

Table 2: Model Ratings

Club	League Points	BIP Net	GRI Net
Chelsea	69	0.42	0.73
Tottenham Hotspur	62	0.38	0.73
Liverpool	59	0.46	0.58
Manchester City	57	0.26	0.49
Manchester United	53	0.08	0.32
Arsenal	50	0.30	0.49
Everton	50	0.24	0.37
West Bromwich Albion	44	0.05	0.02
Stoke City	36	-0.09	-0.18
AFC Bournemouth	34	-0.07	-0.23
Southampton	34	-0.04	-0.07
Watford	34	-0.13	-0.31
Leicester City	33	-0.09	-0.25
West Ham United	33	-0.13	-0.26
Burnley	32	-0.07	-0.23
Crystal Palace	31	-0.07	-0.19
Hull City	27	-0.18	-0.57
Swansea City	27	-0.19	-0.51
Middlesbrough	22	-0.07	-0.25
Sunderland	20	-0.14	-0.51

Table 3: Correlation matrix of net ratings and league points.

	League Points	BIP Net	GRI Net
League Points	1.000	0.930	0.962
BIP Net	0.930	1.000	0.973
GRI Net	0.962	0.973	1.000

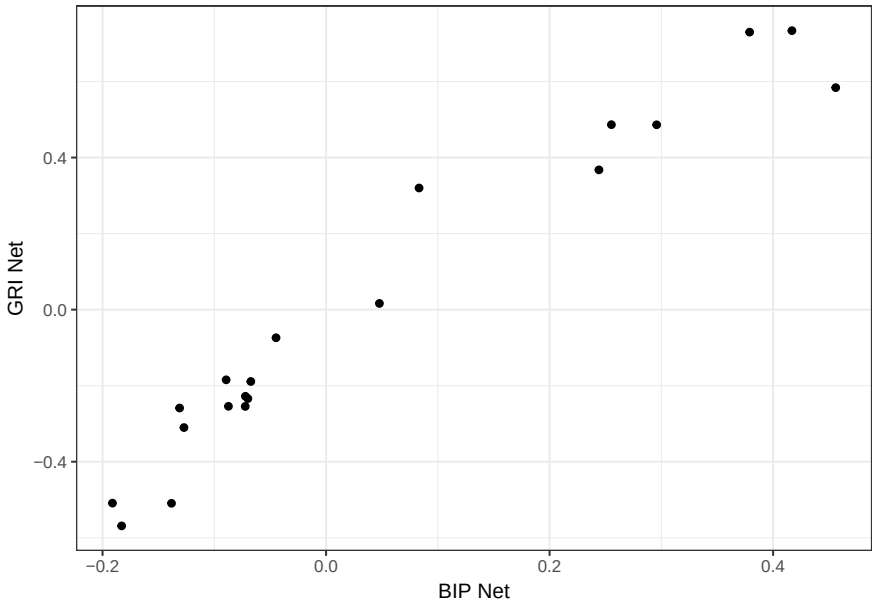

Figure 3: Comparison of net rating across models.

5 Model fit

Model fit can be assessed by posterior predictive model checks. Posterior predictive model checks refer to the generation of a replicated data set at each iteration of the MCMC chains. At each iteration, the current values of the parameters are used to generate a new data set. Thus, it is possible to get replicated data sets that take into account the uncertainty in the parameter estimates, and compare these replicated data sets to the observed data.

5.1 Recovery of score distributions

One way to compare the replicated data sets to the observed data is to look at the distributions of goals scores by the home and away teams. Figures 4 and 5 show that both models are able to recover the observed score distribution without any glaring deviations.

5.2 Prediction error

Alternatively, we could use the replicated data sets to get probabilities for the outcomes of each game (i.e., in what percent of the replicated data sets did the home team win, lose, or tie each game). The outcome probabilities can then be compared to the observed results to evaluate which model had lower prediction error. Using the predictions from the replicated data sets, the bivariate Poisson model gave the observed outcome the highest probability in 59.8 percent of the games, compared to 60.8 percent in the game random intercept model.

One problem with this approach is that it doesn't take into the actual values of the probabilities. For example, take two games where the home team won. In the first game, the home team had a 60 percent chance of winning. In the second game, the home teams had a 90 percent chance of winning. In both cases, the most likely outcome matches the observed outcome, so both instances are assigned a one in the binary loss function as a correct prediction. Alternatively, we could use the log loss function to look at how far the probability was from the observed event (Altun, Johnson, & Hofmann, 2003). In this example, the second game would be

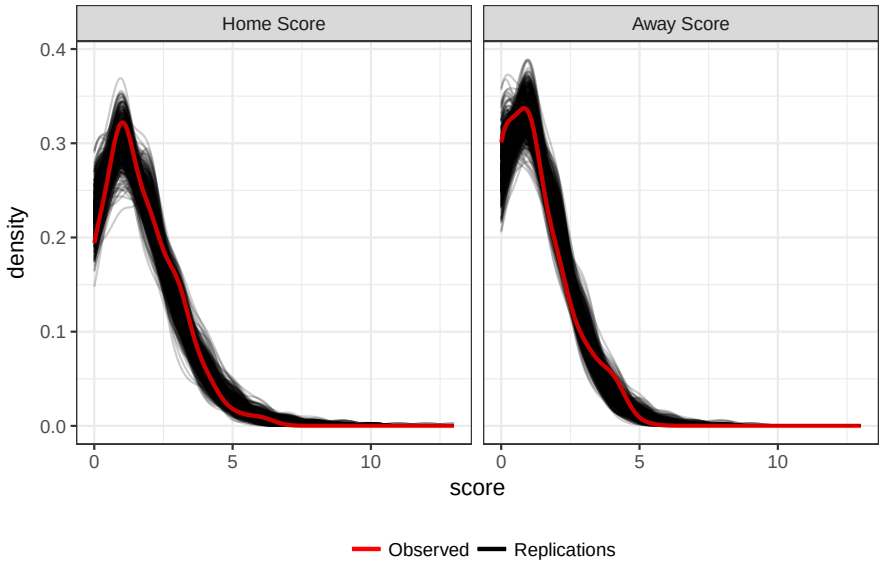

Figure 4: Bivariate Poisson recovery of observed score distributions.

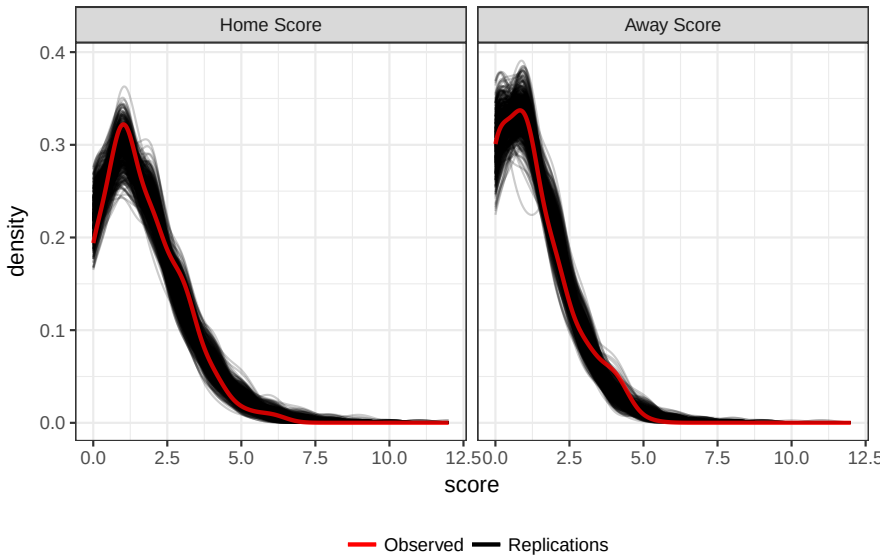

Figure 5: Game random intercept recovery of observed score distributions.

Table 4: Log loss comparison to baseline models.

Model	Log Loss
Game Random Intercept	1.615
Bivariate Poisson	1.649
Data Average	1.820
Equal Probabilities	1.910
Home Win	34.895

a better prediction, and have a lower log loss, because a probability of 0.9 is closer to the observed outcome (1) than a probability of 0.6.

The log loss for multiple games is defined in equation (1).

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij}) \tag{1}$$

In equation (1), N is number of observations (in this case the number of games), M is the number of outcomes (for soccer games this is 3: win, loss, and tie), y_{ij} is a binary indicator of whether outcome M occured for observation N (1) or not (0), and p_{ij} is the probability of outcome M for observation N. Thus, the log loss for a set of predictions is the log of the probability of the observed outcomes, summed over all observations. In the case of perfect predictions, the probability of the observed outcome would be 1, and the log probability would be 0. This means that the closer the log loss is to 0, the better the predictions are (Roy & McCallum, 2001).

The log loss for predictions from the replicated data sets is 1.649 in the bivariate Poisson model, and 1.615 in the game random intercept model. In isolation, the log loss can be hard to interpret. Instead, it is often useful to compare to baseline models. Table 4 shows the log loss for a variety of models. In the data average model, the probability of each outcome is set to the overall average for the entire data set. In the equal probabilities model, the probability for each outcome is set to 0.33. Finally, in the home win model, the probability of the home team winning is set to 1 and the probability of the other outcomes is set to 0. Both the bivariate Poisson and game random intercept models out perform all baseline models, but the game random intercept model out performs the bivariate Poisson model.

These findings suggest that the game random intercept model is providing better model-data fit. This is consistent with the findings of a small simulation study I conducted, which found that the game random intercept model was better able to recover the true parameters (Section 3). Given these findings, I would recommend using the game random intercept model for predictions moving forward. Accordingly, predictions based on this model for the major European domestic leagues (including the English Premier League) and the UEFA Champions League can be seen in Section 6.

Appendix

A Stan Models

A.1 Bivariate Poisson

```
data {
  int<lower=1> num_clubs;                          // number of clubs
  int<lower=1> num_games;                          // number of games
  int<lower=1,upper=num_clubs> home[num_games];    // home club for game g
```

```
  int<lower=1,upper=num_clubs> away[num_games];     // away club for game g
  int<lower=0> h_goals[num_games];                  // home goals for game g
  int<lower=0> a_goals[num_games];                  // away goals for game g
  int<lower=0,upper=1> homeg[num_games];            // home field for game g
}
parameters {
  vector[num_clubs] raw_alpha;                  // attacking intercepts
  vector[num_clubs] raw_delta;                  // defending intercepts
  vector[num_clubs] raw_rho;                    // covariance intercepts

  real mu;                                          // fixed intercept
  real eta;                                         // homefield
  real<lower=0> sigma_a;                            // attacking sd
  real<lower=0> sigma_d;                            // defending sd
  real<lower=0> sigma_r;                            // covariance sd
}
transformed parameters {
  vector[num_clubs] alpha;
  vector[num_clubs] delta;
  vector[num_clubs] rho;

  alpha = raw_alpha * sigma_a;
  delta = raw_delta * sigma_d;
  rho = raw_rho * sigma_r;
}
model {
  vector[num_games] lambda1;
  vector[num_games] lambda2;
  vector[num_games] lambda3;

  // priors
  raw_alpha ~ normal(0, 1);
  raw_delta ~ normal(0, 1);
  raw_rho ~ normal(0, 1);
  mu ~ normal(0, 10);
  eta ~ normal(0, 10);
  sigma_a ~ normal(0, 10);
  sigma_d ~ normal(0, 10);
  sigma_r ~ normal(0, 10);

  // likelihood
  for (g in 1:num_games) {
    lambda1[g] = exp(mu + (eta * homeg[g]) + alpha[home[g]] + delta[away[g]]);
    lambda2[g] = exp(mu + alpha[away[g]] + delta[home[g]]);
    lambda3[g] = exp(rho[home[g]] + rho[away[g]]);
  }
  h_goals ~ poisson(lambda1 + lambda3);
  a_goals ~ poisson(lambda2 + lambda3);
}
generated quantities {
  real lambda1_rep[num_games];
  real lambda2_rep[num_games];
  real lambda3_rep[num_games];
  int home_rep[num_games];
```

```
  int away_rep[num_games];

  for (g in 1:num_games) {
    lambda1_rep[g] = exp(mu + (eta * homeg[g]) + alpha[home[g]] + delta[away[g]]);
    lambda2_rep[g] = exp(mu + alpha[away[g]] + delta[home[g]]);
    lambda3_rep[g] = exp(rho[home[g]] + rho[away[g]]);

    home_rep[g] = poisson_rng(lambda1_rep[g] + lambda3_rep[g]);
    away_rep[g] = poisson_rng(lambda2_rep[g] + lambda3_rep[g]);
  }
}
```

A.2 Game random intercept

```
data {
  int<lower=1> num_clubs;                          // number of clubs
  int<lower=1> num_games;                          // number of games
  int<lower=1,upper=num_clubs> home[num_games];    // home club for game g
  int<lower=1,upper=num_clubs> away[num_games];    // away club for game g
  int<lower=0> h_goals[num_games];                 // home goals for game g
  int<lower=0> a_goals[num_games];                 // away goals for game g
  int<lower=0,upper=1> homeg[num_games];           // home field for game g
}
parameters {
  vector[num_clubs] raw_alpha;                     // attacking intercepts
  vector[num_clubs] raw_delta;                     // defending intercepts
  vector[num_games] raw_gamma;                     // game intercepts

  real mu;                                         // fixed intercept
  real eta;                                        // homefield
  real<lower=0> sigma_a;                           // attacking sd
  real<lower=0> sigma_d;                           // defending sd
  real<lower=0> sigma_g;                           // game sd
}
transformed parameters {
  vector[num_clubs] alpha;
  vector[num_clubs] delta;
  vector[num_games] gamma;

  alpha = sigma_a * raw_alpha;
  delta = sigma_d * raw_delta;
  gamma = sigma_g * raw_gamma;
}
model {
  vector[num_games] lambda1;
  vector[num_games] lambda2;

  // priors
  raw_alpha ~ normal(0, 1);                        // attacking random effects
  raw_delta ~ normal(0, 1);                        // defending random effects
  raw_gamma ~ normal(0, 1);                        // game random effects
  mu ~ normal(0, 10);
  eta ~ normal(0, 10);
```

```
  sigma_a ~ normal(0, 10);
  sigma_d ~ normal(0, 10);
  sigma_g ~ normal(0, 10);

  // likelihood
  for (g in 1:num_games) {
    lambda1[g] = exp(mu + (eta * homeg[g]) + alpha[home[g]] + delta[away[g]] + gamma[g]);
    lambda2[g] = exp(mu + alpha[away[g]] + delta[home[g]] + gamma[g]);
  }
  h_goals ~ poisson(lambda1);
  a_goals ~ poisson(lambda2);
}
generated quantities {
  int home_rep[num_games];
  int away_rep[num_games];

  for (g in 1:num_games) {
    home_rep[g] = poisson_rng(exp(mu + (eta * homeg[g]) + alpha[home[g]] + delta[away[g]] + gamma[g]));
    away_rep[g] = poisson_rng(exp(mu + alpha[away[g]] + delta[home[g]] + gamma[g]));
  }
}
```

References

Altun, Y., Johnson, M., & Hofmann, T. (2003). Investigating loss functions and optimization methods for discriminative learning of label sequences. In *Proceedings of the 2003 conference on empirical methods in natural language processing* (pp. 145–152). Association for Computational Linguistics.

Betancourt, M. (2016). *Diagnosing suboptimal cotangent disintegrations in hamiltonian monte carlo*. Retrieved from https://arxiv.org/abs/1604.00695

Grolemund, G., Spinu, V., & Wickham, H. (2016). *Lubridate: Make dealing with dates a little easier*.

Guo, J., Gabry, J., & Goodrich, B. (2016). *Rstan: R interface to stan*. Retrieved from https://CRAN.R-project.org/package=rstan

Roy, N., & McCallum, A. (2001). Toward optimal active learning through monte carlo estimation of error reduction. In *International conference on machine learning* (pp. 441–448). Williamstown: International Machine Learning Society.

Stan Development Team. (2016a). *Brief guide to stan's warnings*. Retrieved from http://mc-stan.org/misc/warnings.html

Stan Development Team. (2016b). *Stan modeling language: User's guide and reference manual*. Retrieved from http://mc-stan.org/documentation/

Wickham, H. (2016a). *Purrr: Functional programming tools*.

Wickham, H. (2016b). *Rvest: Easily harvest (scrape) web pages*. Retrieved from https://CRAN.R-project.org/package=rvest

Wickham, H., & Francois, R. (2016). *Dplyr: A grammar of data manipulation*. Retrieved from https://CRAN.R-project.org/package=dplyr