Modeling points per player in the NBA

Franco Betteo

1 GSU Project 2023

1.1 Introduction

The goal of this project is to model and predict the winner of an incoming basketball NBA game. There are probably multiple models for that but I haven't seen one explicitly bayesian and I think there are good reasons to try. The goal is to predict the points each team will score (some distribution of points) and compare those distributions to calculate the probability that a random draw from the distribution of team A for that game will be greater than the random draw from team B.

One problem modeling team points is that a team, as a "subject", is not stable. There are some patterns, like teams historically more winners than others or usually candidates to win the title but with high variance.

- Each year the team composition can vary a lot depending on how active they are during the offseason trades and even during the course of the season there are trade dates where a team can modify their roster greatly (as today, May 2023, the Lakers are a clear example acquiring 5 players during the latest trade deadline, including 2 starters.)

- Another common situation are injuries, players can lose multiple games or even be out for the rest of the season due to injuries and that can affect greatly the win probabilities of their teams.

- Lastly, new players enter the league each year and can completely change the situation of a team (ex, getting a superstar from college basketball)

In order to counter that, the approach I would like to try is to model points per player. That will be the core of this work. With the distribution of points per player we can later aggregate for each game only the players that are available to play and we can use the model in the future even when rosters change drastically.
On top of that, the idea is to use a hierarchical model, so we can have a more stable prediction for players with low amount of games (low amount of observations) and even for rookies where we have 0 observation when they enter the league.

1.2 Data

The data is a processed dataset that I have created previously for other personal projects and for a presentation I had to do during my master's degree. The analysis done back then wasn't to predict points per player so this is novel with respect to that.
The raw data comes from the paid service of Mysportsfeeds.
The main information we will use for this analysis is at the player-game level. This means, for each game we have how many points each player scored and other features such as if the game has home or away and how many hours each player rested since the last game.
Initially we will use the 2021-2022 season as our training data and latest games as any test data required.
On top of this, at the modeling time I'm taking a random sample of 50 players because an initial test on full data took multiple hours to run and more complex models didn't start to sample in my computer.

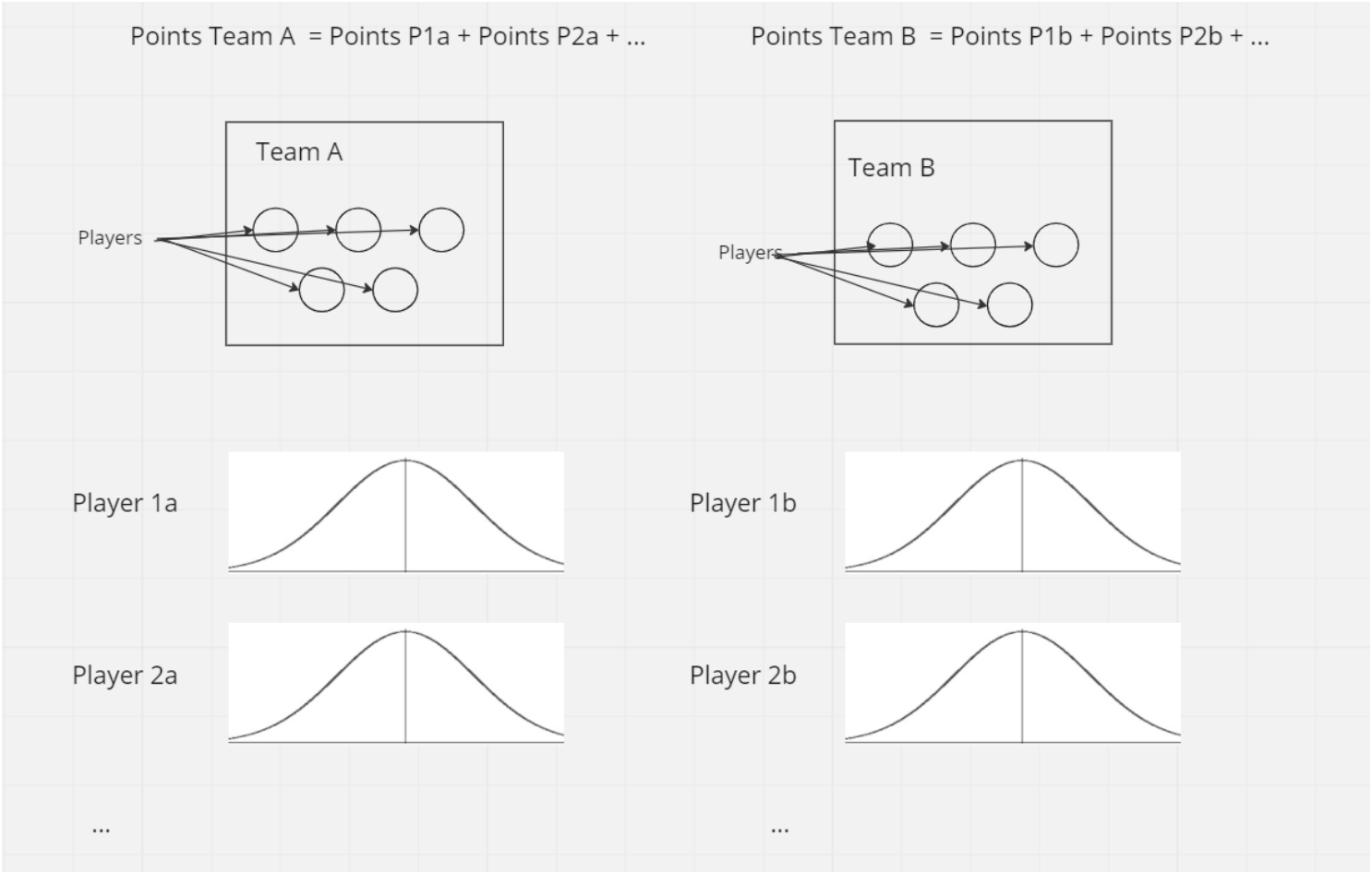


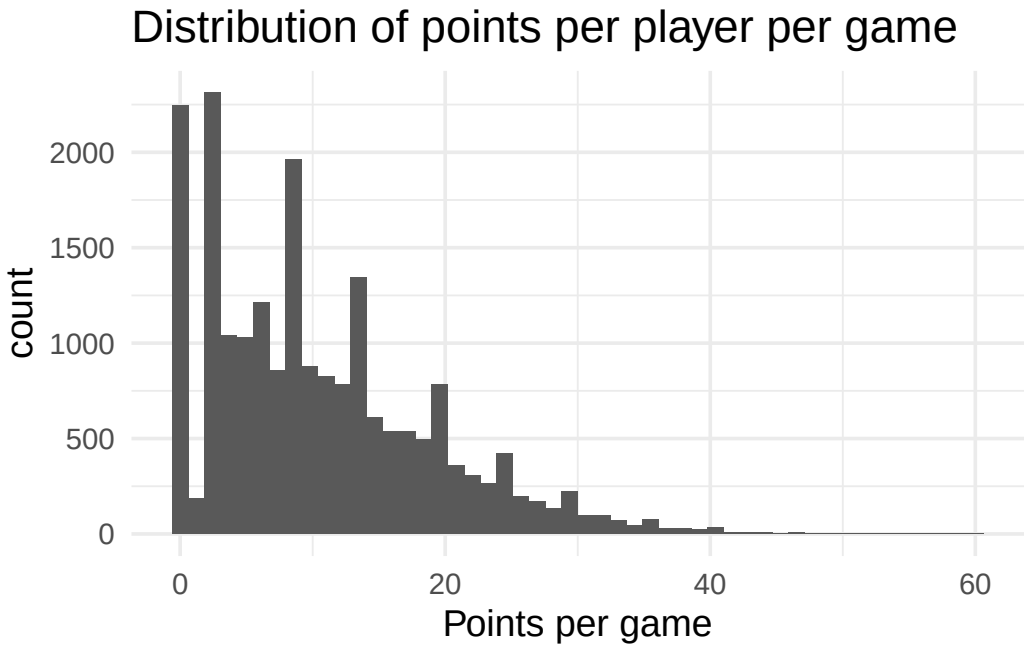
Figure 1: From players to teams

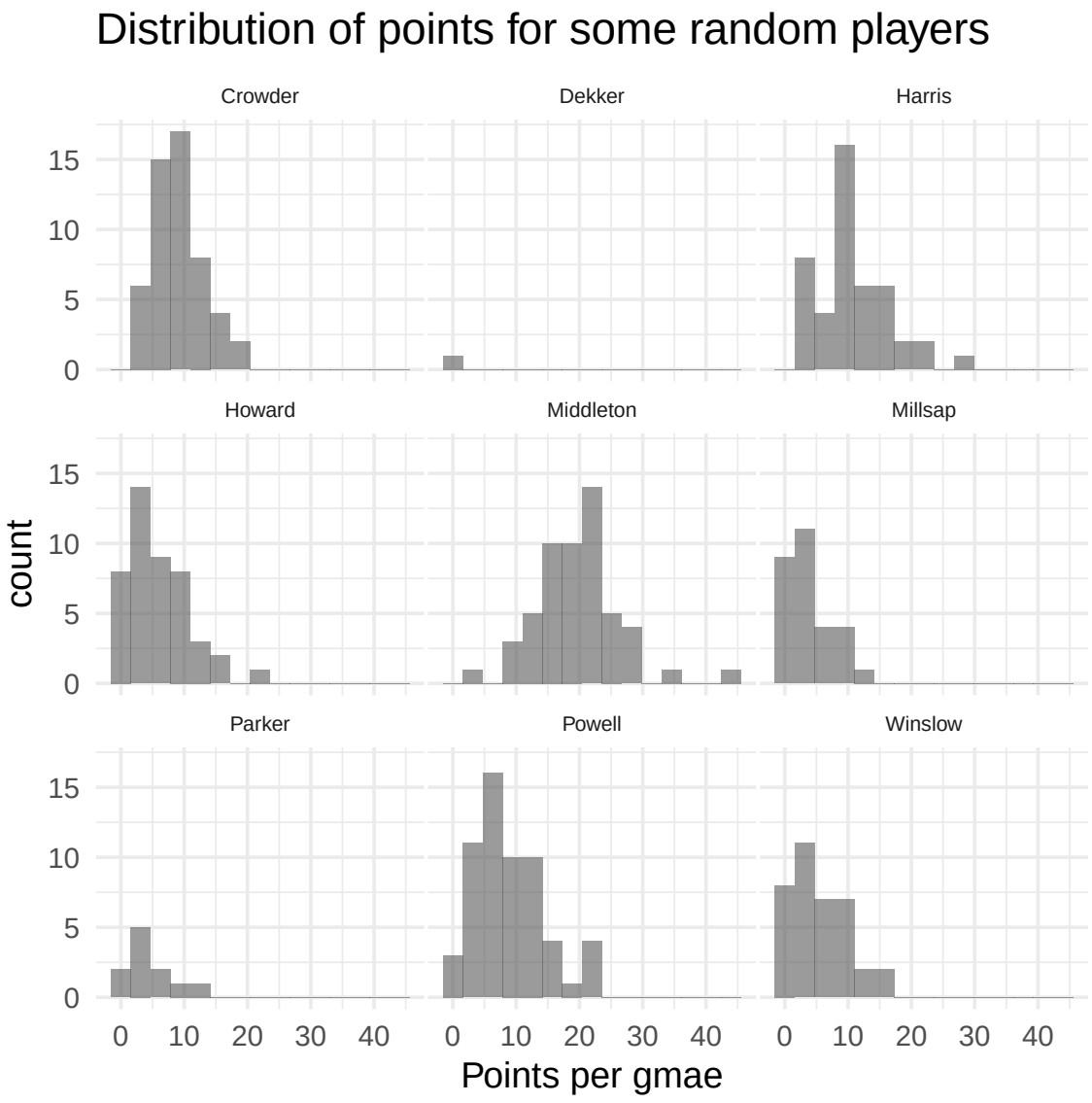
Not sure if it's because of my computer, bad priors or bad specification not allowing correct sampling or actually it was fine but computations are hard.

Here are the first 6 records of the data.

	playerId	firstName	lastName	position	teamId	startTime	pts	atHome	rest_hours
19175	31065	Miles	McBride	PG	83	2021-11-07T22:00:00.000Z	0	1	46.5
14739	17305	Keldon	Johnson	SG	106	2022-01-01T01:00:00.000Z	6	0	95.5
13384	17224	Terance	Mann	SF	102	2022-03-09T03:00:00.000Z	7	0	48.0
1239	9169	Tristan	Thompson	C	89	2022-03-05T00:30:00.000Z	4	1	24.5
10658	15206	Kevin	Knox II	PF	91	2022-02-25T01:00:00.000Z	2	0	100.0
18333	31039	Ziaire	Williams	SF	107	2022-03-13T23:00:00.000Z	11	0	100.0

It has data from 2021-10-19 to 2022-04-11.
It contains 20278 rows with 601 players from 30 teams. Each player has on average 34 observations.
The lowest is 1 and the maximum is 67
We have players with really low amount of samples where hierarchical model can be useful.





1.3 Models to fit.

First we will fit a no pooled linear regression where points per game is the target variable and each player will have an intercept coefficient and there will be a unique coefficient for the impact on points of resting hours between matches. The assumption is that players score more if they are allowed to rest more. Obviously there is a limit to that and probably it's not a linear effect but we will start from that and the data is clipped to 72 hours rest, so any longer distance between matches is as if was 3 days.

Later we will reproduce this same model but instead of having a no pooled model we will use a hierarchical model where the intercept for each player is no longer completely separate from the others but they all come from a common distribution. The resting hours will also be per player but coming from a general distribution.

For both cases we will use the normal likelihood as the players seem to have an approximately symmetrical distribution of points. I'm aware there are multiple problems with this such as:
* allows for negative points and will draw negative values * expects negative values for players with low mean and we now it is truncated at 0 * points per player are discrete

but despite that the normal distribution can do a not so poor job and I'm more familiar on how to fit that on Stan and I want to avoid any caveat and required experimentation with other likelihoods.

1.4 Priors

For the No pooled model each player will have a weakly informative Normal prior for the μ with large variance to let the data speak.
For the standard deviation each player will have an exponential prior but also allowing for a wide range of values.
The coefficient for rest hours will also have a weakly informative exponential prior. Exponential because we want as well the coefficient to be positive.

For the Hierarchical model each player will have their intercept coming from a truncated normal distribution, the standard deviation again from an exponential and the rest hours coefficient also from a truncated normal distribution. The super population parameters all come also from weakly informative truncated Normal distributions. All truncated because every coefficient makes sense if they are positive.
Probably there are better ways to do this but truncating and using normal distributions worked for sampling and to get positive coefficients.

1.5 Model configuration

For model fitting I'm using cmdstan with a .stan file with the definition of the model. I'm using parallelization of the chains.
In the appendix you can find both stan codes.

1.6 No pooled model

```
# # FITTING
# model_separate_rest_hours <- cmdstan_model(stan_file = "separate_rest_hours.stan")
#
# # Sampling from the model happens here:
# fit_separate_rest_hours <- model_separate_rest_hours$sample(data = stan_data, refresh=0, parallel_
#                                     show_messages=FALSE)
#
# fit_separate_rest_hours$save_object(file = "fit_separate_rest_hours.RDS")
```

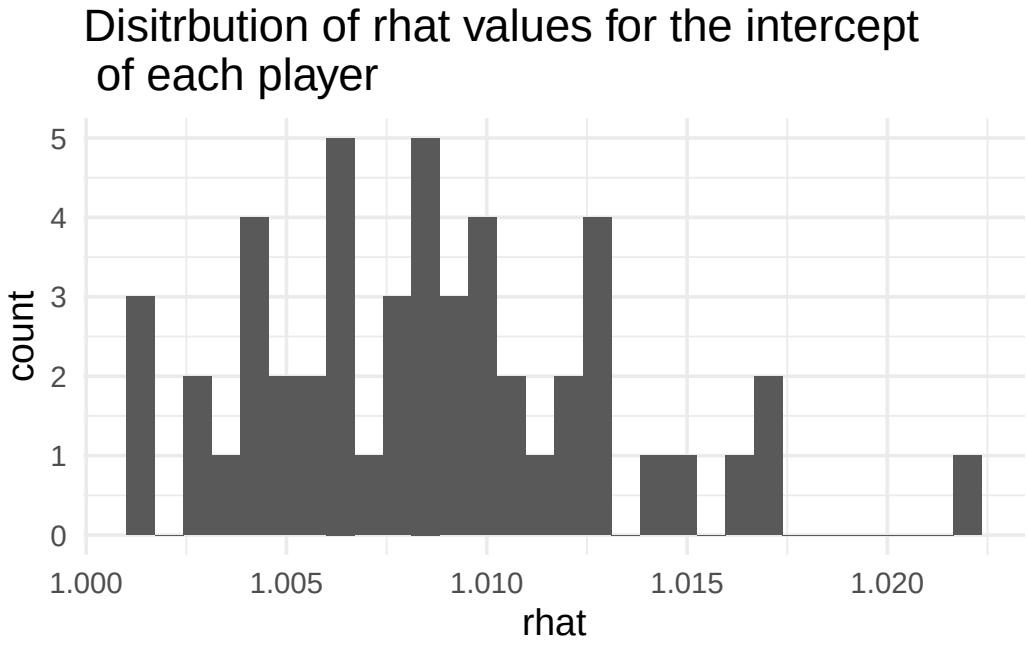
```
fit_separate_rest_hours = readRDS("fit_separate_rest_hours.RDS")
```

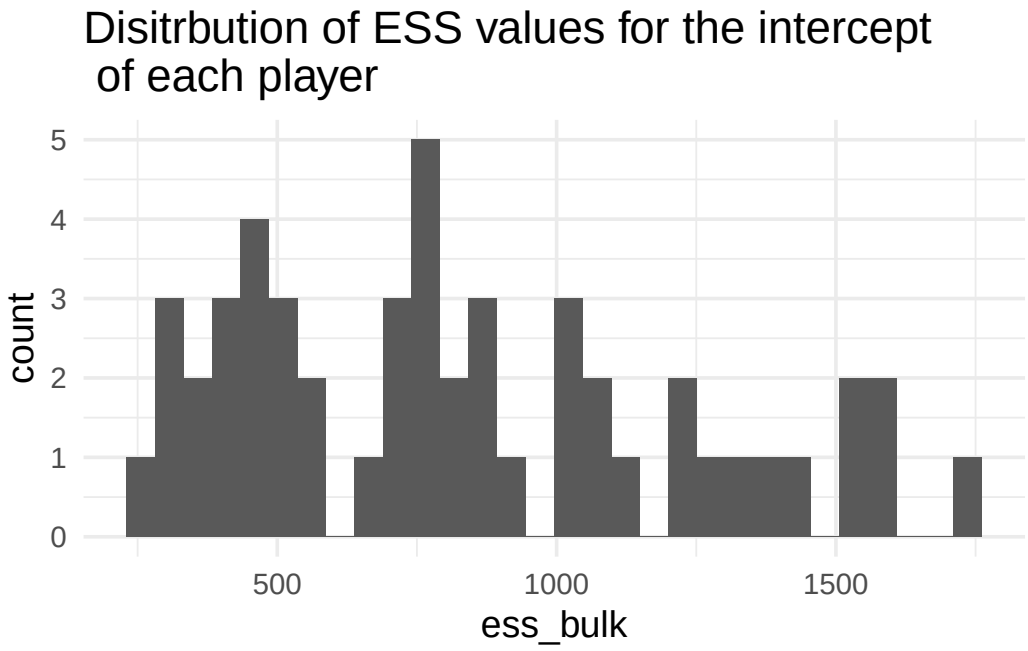
```
Processing csv files: C:/Users/Franco/AppData/Local/Temp/RtmpO0gfRz/separate_rest_hours-202305162311-1

Checking sampler transitions treedepth.
Treedepth satisfactory for all transitions.

Checking sampler transitions for divergences.
1022 of 4000 (25.55%) transitions ended with a divergence.
These divergent transitions indicate that HMC is not fully able to explore the posterior distribution.
Try increasing adapt delta closer to 1.
If this doesn't remove all divergences, try to reparameterize the model.

Checking E-BFMI - sampler transitions HMC potential energy.
E-BFMI satisfactory.

Effective sample size satisfactory.

Split R-hat values satisfactory all parameters.
```

```
Processing complete.
```

```
Warning in styling_latex_scale_down(out, table_info): Longtable cannot be
resized.
```

variable	mean	median	sd	q5	q95	rhat	ess_bulk	ess_tail
int_player[1]	3.3902666	3.386890	0.7176527	2.178555	4.576816	1.011739	546.1259	296.69832
int_player[2]	9.0010237	9.000285	0.6122492	8.037323	9.997650	1.004304	820.0158	2039.36689
int_player[3]	8.5928268	8.566000	0.7721184	7.328328	9.907708	1.016842	505.1475	92.76888
int_player[4]	10.7402526	10.783550	0.9373546	9.175781	12.356825	1.005824	856.3674	191.22526
int_player[5]	5.6402904	5.662165	0.8353798	4.297640	6.979160	1.012048	316.9035	231.27434
int_player[6]	0.8116305	0.033963	23.1130056	-37.881935	37.665025	1.007994	371.2698	173.59873



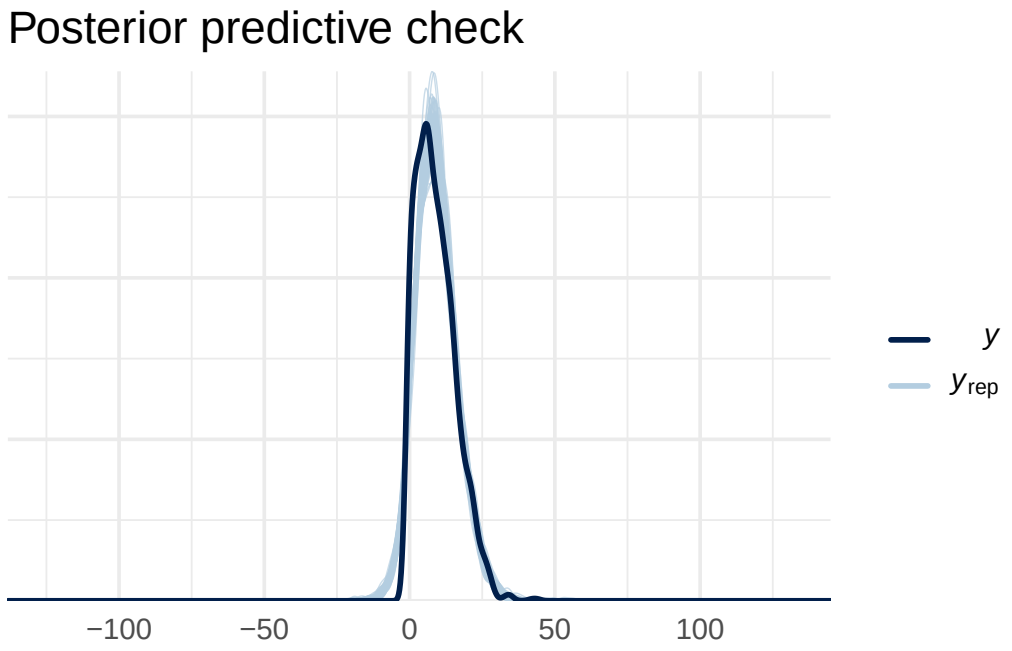


From the output of the fit of the model we can see that around 25% of the transitions end up in a divergence. However we can see that no parameter has a extremely high rhat. The largest is bout 1.02 and most are below 1.01.

The bulk effective sample size seems pretty healthy as most of the values are over 400 with only a few between 250 and 400.

If we look at the posterior predictive check we can see it's not bad but not perfect. On the weak side we can see the posterior allows for negative values, including a draw as low as -138.261 and one as high as 144.885 Then the mode is a bit shifted to the right in the posterior compared to the real data but not that bad.
The highest end of the distribution (not taking into account extreme draws) has a very decent fit.

```
ppc_dens_overlay(y = train$pts, yrep=as.matrix(sample_pred_draws)) +
  ggtitle("Posterior predictive check")
```



We calculate below the leave one out expected log point wise predictive density (elpd_loo) for this model. We will later compare it with the hierarchical model.

```
Computed from 4000 by 1556 log-likelihood matrix

         Estimate   SE
elpd_loo  -4877.5 35.6
p_loo       108.4 10.8
looic      9755.0 71.1
------
Monte Carlo SE of elpd_loo is NA.

Pareto k diagnostic values:
                         Count Pct.    Min. n_eff
(-Inf, 0.5]   (good)     1531  98.4%   19
 (0.5, 0.7]   (ok)         13   0.8%   39
   (0.7, 1]   (bad)        11   0.7%   14
   (1, Inf)   (very bad)    1   0.1%   4
See help('pareto-k-diagnostic') for details.
```

1.6.0.0.1 TODO: Sensitivty

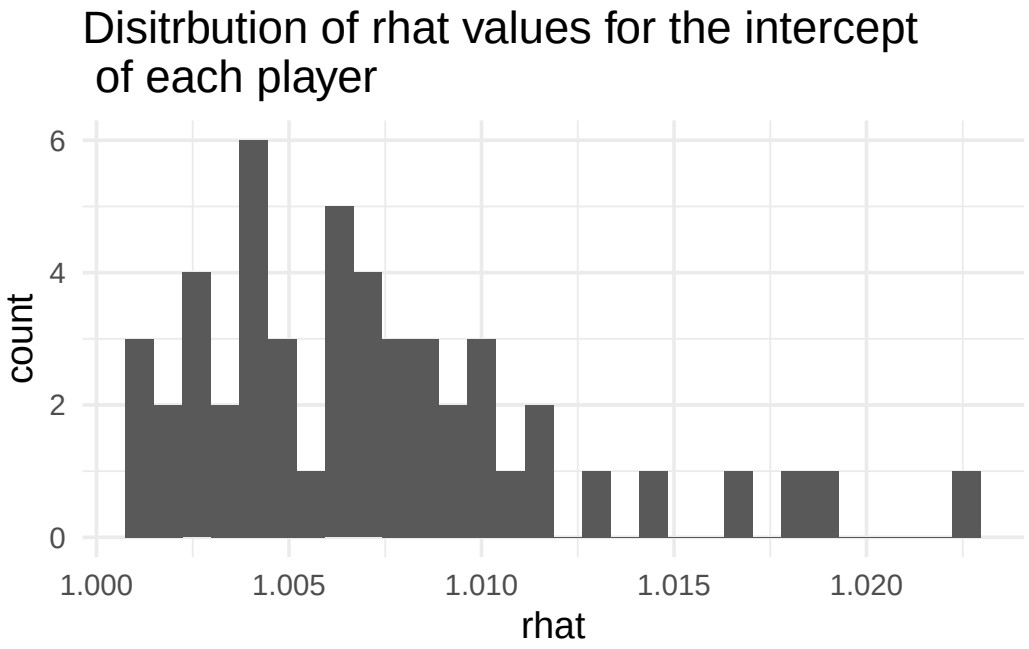
1.7 Hierarchical model

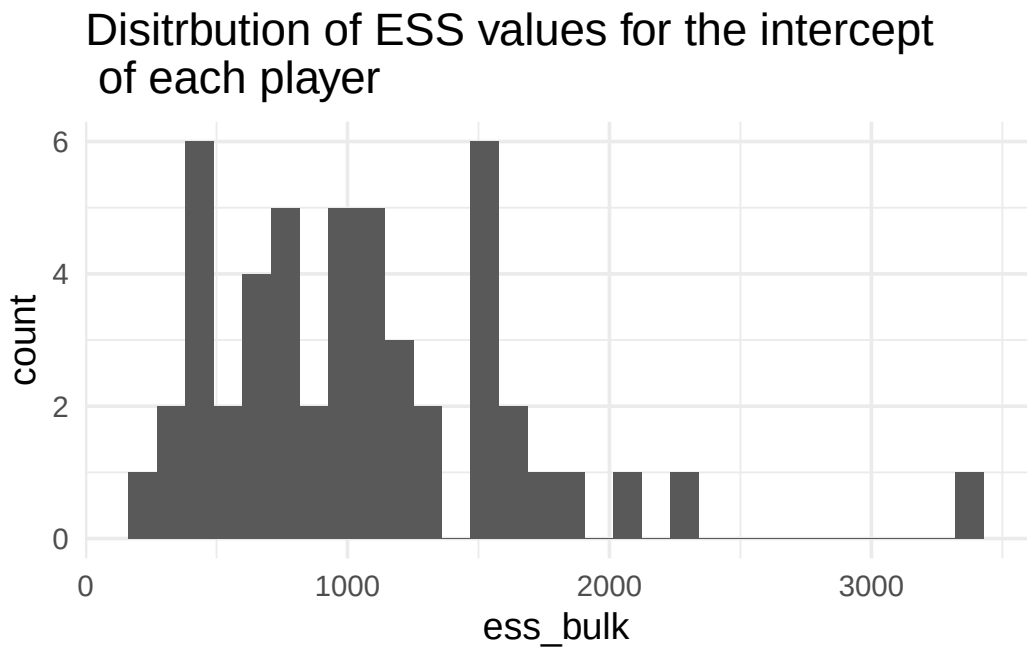
```
# hierarchical_rest <- cmdstan_model(stan_file = "hierarchical_rest_hours.stan")
#
# # Sampling from the model happens here:
# fit_hierarchical_rest <- hierarchical_rest$sample(data = stan_data, refresh=0, parallel_chains  =
#                                     show_messages=FALSE)
#
# fit_hierarchical_rest$save_object(file = "fit_hierarchical_rest.RDS")
```

```
fit_hierarchical_rest = readRDS("fit_hierarchical_rest.RDS")
```

From the output of the fit of the model we can see that around only 7% of the transitions end up in a divergence. However we can see that no parameter has a extremely high rhat. The largest is bout 1.02 and most are below 1.01.

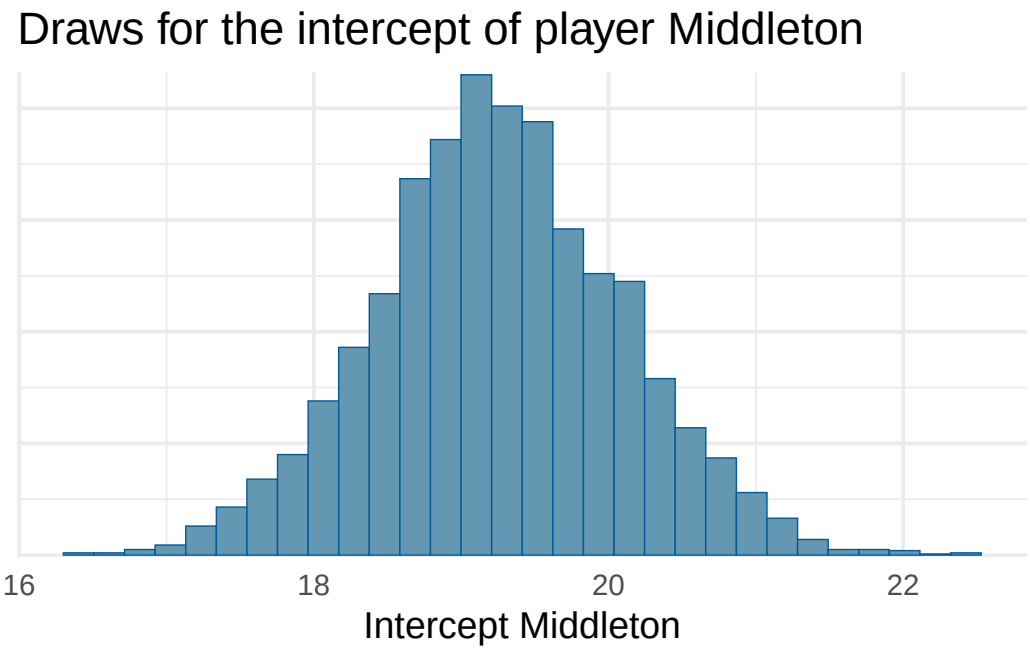
The bulk effective sample size seems pretty healthy for the intercept of each player (values above 400) but we can see issues for the parameter for resting hours. Most of them have bulk ESS lower than 100.



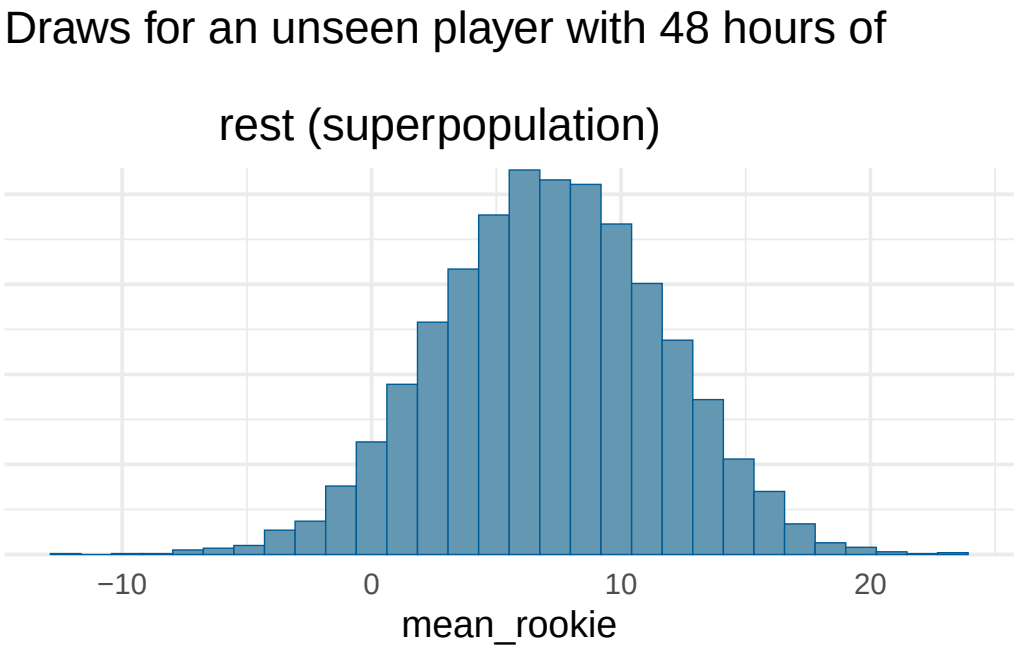



```
Warning in styling_latex_scale_down(out, table_info): Longtable cannot be
resized.
```

variable	mean	median	sd	q5	q95	rhat	ess_bulk	ess_tail
mean_player[1]	3.455437	3.414195	0.6734894	2.3888915	4.589370	1.006363	1204.4253	900.7739
mean_player[2]	9.033578	9.039365	0.5955054	8.0230305	10.020000	1.006948	683.2221	888.5209
mean_player[3]	8.465124	8.480065	0.6982539	7.2850280	9.621330	1.002953	769.1219	322.9217
mean_player[4]	10.589592	10.577650	0.9543285	9.0437845	12.135695	1.009409	485.9691	348.1366
mean_player[5]	5.738414	5.744380	0.7676649	4.4148015	7.046121	1.001919	1030.4890	485.3258
mean_player[6]	3.403138	2.521695	3.0501281	0.1047816	9.458927	1.001100	949.8827	1333.5679



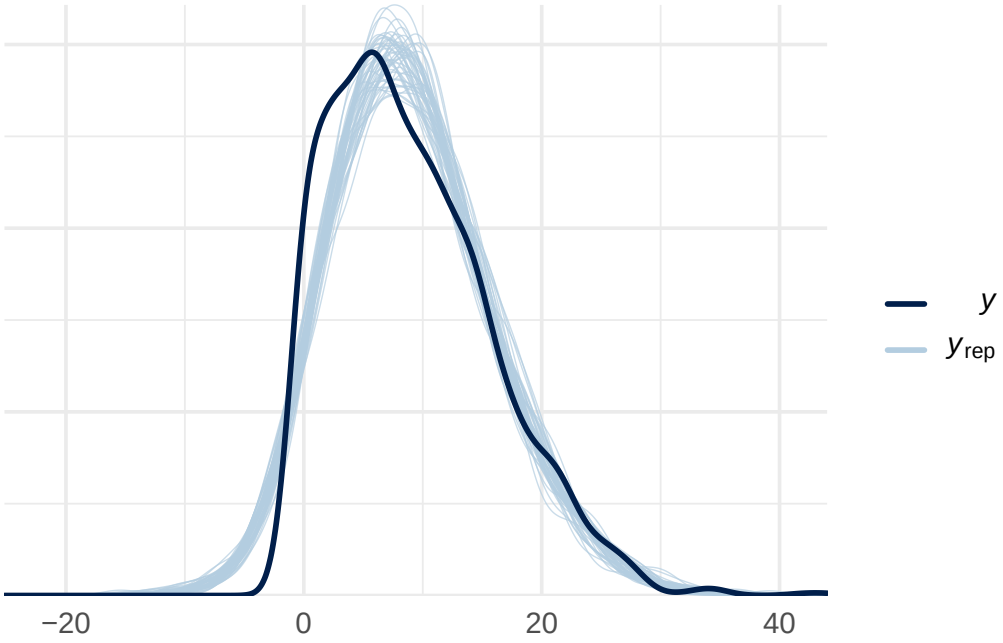

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



We can see again the negative draws in here…

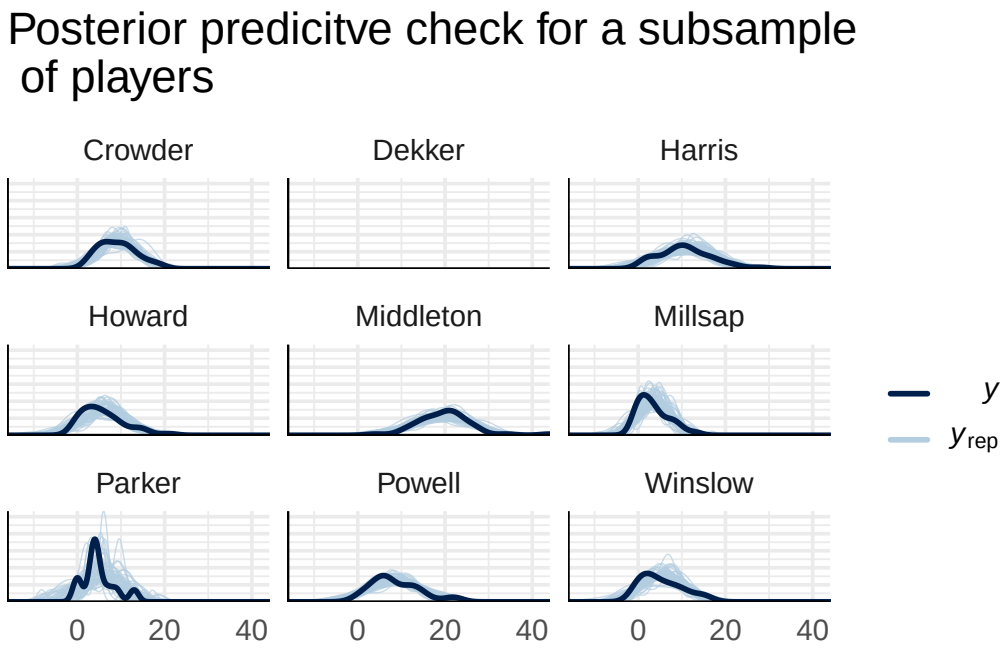
And if we look at the full posterior predictive distribution we can a see a similar shape as in the no pooled model, with still negative values but without extremes values. Now the lowest draw is -25.167 and the highest is 43.5642. The mode is still a bit shifted overall we have a decent fit.

```
$title
```




```
$subtitle
[1] "Posterior predictive check"

attr(,"class")
[1] "labels"
```

By player we can see also a decent fit for a subset of players despite any difference in the number of observations they might have. Dekker has only one observation so we don't a have a distribution to show.



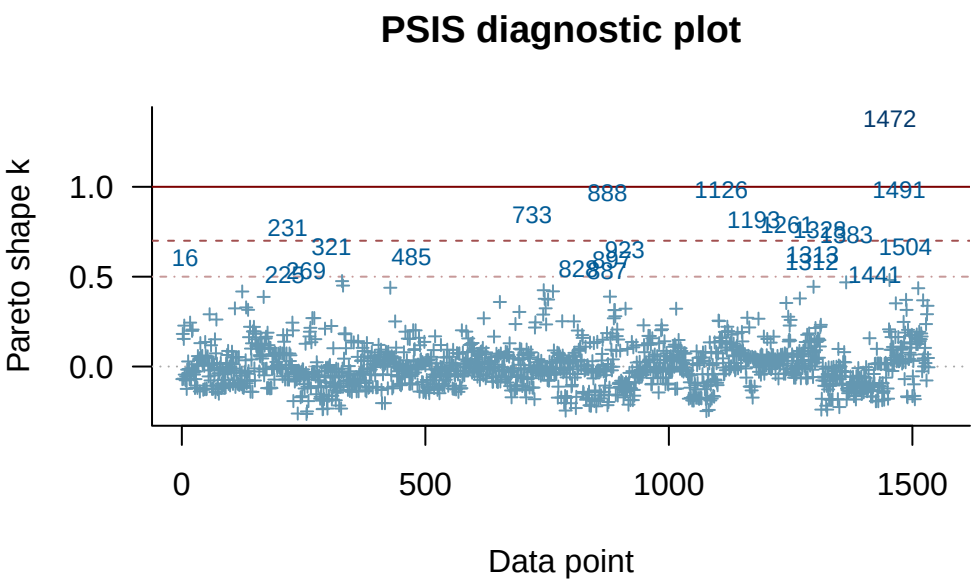

```
Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
Computed from 4000 by 1556 log-likelihood matrix

         Estimate   SE
elpd_loo  -4859.3 36.7
p_loo       102.6 10.4
looic      9718.6 73.4
------
Monte Carlo SE of elpd_loo is NA.

Pareto k diagnostic values:
                         Count Pct.    Min. n_eff
(-Inf, 0.5]   (good)     1533  98.5%   69
 (0.5, 0.7]   (ok)         13   0.8%   30
   (0.7, 1]   (bad)         9   0.6%   7
   (1, Inf)   (very bad)    1   0.1%   5
See help('pareto-k-diagnostic') for details.
```

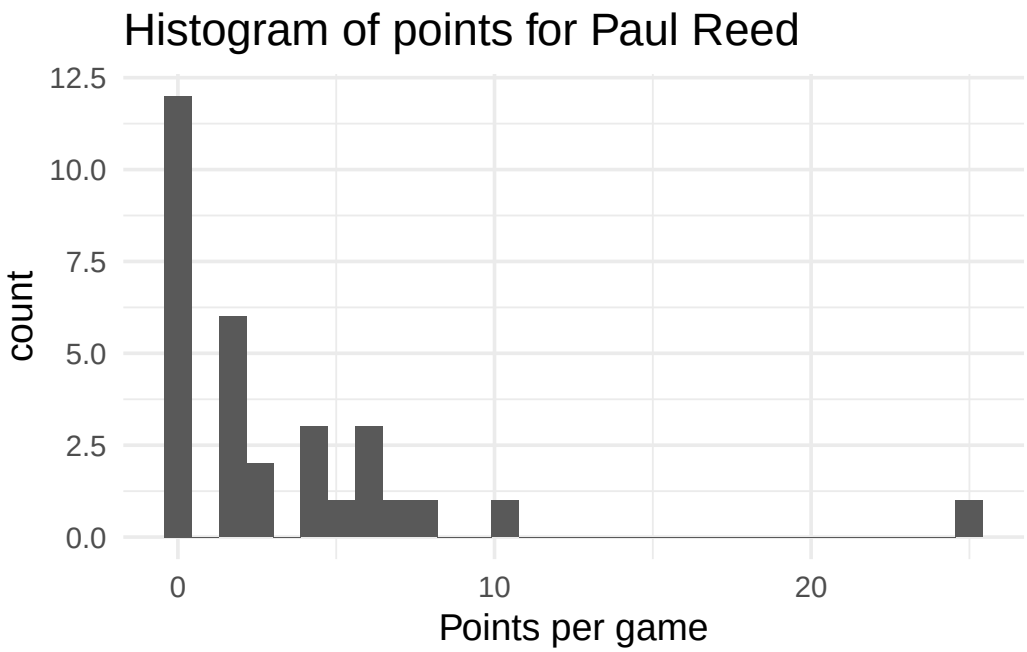
1.7.0.1 Loo comparison

The hierarchical model has the best predictive performance and the standard deviations of the differences are small compared to the actual difference so they don't influence the decision.

```
       elpd_diff se_diff
model2   0.0       0.0
model1 -18.2       4.8
```



The hierarchical model has around 1.5% observations with K statistic higher than 0.5 while the pooled model had 1.7%. Looking at the chart these higher k statistic observations seem to be just a few distributed across the different players.
More specifically they appear when a player has a surprisingly high number of points in a game compared to their other matches. For example, the highest K value, observation 1472, corresponds to Paul Reed.
We can see in the histogram he scores usually 0 points and almost sure less than 10 points but one game he scored 25. That one is the observation with the highest K pareto value.



1.8 Next steps

There are many things I want to do with the model and because of general lack of time and how inefficient I am with Stan and bayesian models yet I couldn't do or I decided to not dive more despite starting trying.
Among them:

- Try a negative binomial model to account for discrete positive values and being able to account for overdispersion.

- Add more variables such as home court, amount of matches won so far in the season, age, etc and think properly about the priors.
- Evaluate the model not only at the player level but in the general purpose that is which team wins. This involves summing the players points per draw of each team and compare.
- Use the model to predict on unseen data.
- Run the model with all the data (all the players and all the dates) since I took a sample because some trials were taking too long or not even sampling.
- Maybe use brms to get full use of other functions to diagnose the model and more out of the box working things since it seems to me that using Stan requires more work apart from defining everything. I think I lost time on that.

1.9 Appendix

1.9.0.0.1 No pooled

```
data {
  int<lower=0> N_observations;
  int<lower=0> N_players;
  array[N_observations] int player_idx;
  vector[N_observations] rest_hours;
  vector[N_observations] pts;
}

parameters {
  // Average points per game per player
  vector[N_players] int_player;

  // Standard deviation points per game per player
  vector<lower=0>[N_players] sd_player;

  real<lower=0> beta_rest;

}

transformed parameters {
    // deterministic transformation of parameters and data
    vector[N_observations] mu_player = int_player[player_idx] + beta_rest * rest_hours ;// linear mo
}




model {
  // Priors
  for (player in 1:N_players) {
    // Weakly informative. Vague. Based on assignment 7 logic.
    int_player[player] ~ normal(10, 100);
    // is this ok?
    sd_player[player] ~ exponential(0.05);
```

```
  }

  // Increase in points per hour rested. Is this ok?
  beta_rest ~ exponential(0.05);


  // Likelihood
  for (obs in 1:N_observations) {
    pts[obs] ~ normal(mu_player[obs] , sd_player[player_idx[obs]]);
  }


  // Best practice would be to write the likelihood without the for loop as:
  // weight ~ normal(mean_diet[diet_idx], sd_diet[diet_idx]);
}

generated quantities {

  // As I don't have covariates I can just simulate one time per player.
  array[N_players] real pts_player = normal_rng(int_player, sd_player);

  // vector[N_observations] mu_pred = mean_player[player_idx];
  // vector[N_observations] mu_pred = mean_player[player_idx]  + beta_rest * rest_hours ;
  vector[N_observations] sd_pred = sd_player[player_idx];
  array[N_observations] real pts_pred = normal_rng(to_array_1d(mu_player) , to_array_1d(sd_pred));


  // For LOO
  vector[N_observations] log_lik;
  for (n in 1:N_observations) {
    log_lik[n] = normal_lpdf(pts[n] | mu_player[n]  , sd_pred[n]);
    // log_lik[n] = normal_lpdf(pts[n] | mean_player[player_idx[n]] + beta_rest * rest_hours , sd_pl
  }
}
```

1.9.0.0.2 Hierarchical

```
data {
  int<lower=0> N_observations;
  int<lower=0> N_players;
  array[N_observations] int player_idx;
  array[N_observations] real rest_hours;
  vector[N_observations] pts;
}

parameters {
  // Average points per game per player
  vector<lower=0>[N_players] mean_player;

  // Standard deviation points per game per player
  vector<lower=0>[N_players] sd_player;
```

```
  vector<lower=0>[N_players] beta_rest;


  real<lower=0> mean_general;
  real<lower=0> sd_general;
  real<lower=0> mean_rest;
  real<lower=0> sd_rest;
}



transformed parameters {

    vector[N_observations] mu_player;
    // deterministic transformation of parameters and data
    for (obs in 1:N_observations) {
      mu_player[obs] = mean_player[player_idx[obs]] + beta_rest[player_idx[obs]] * rest_hours[obs] ;
    }
}

model {

    // Hierarchical
    mean_general ~ normal(10,20);
    sd_general ~ normal(0,30);

    mean_rest ~ normal(0,1);
    sd_rest ~ normal(0,1);


  // Priors
  for (player in 1:N_players) {
    // Weakly informative. Vague. Based on assignment 7 logic.
    mean_player[player] ~ normal(mean_general, sd_general);
    // is this ok?
    sd_player[player] ~ exponential(0.3);

    beta_rest[player] ~ normal(mean_rest, sd_rest);
  }


  // Likelihood
  for (obs in 1:N_observations) {
    // pts[obs] ~ normal(mean_player[player_idx[obs]] + beta_rest[player_idx[obs]] , sd_player[playe
    pts[obs] ~ normal(mu_player[obs] , sd_player[player_idx[obs]]) ;
  }


}

generated quantities {

  vector[N_observations] mu_pred;
```

```
  array[N_players] real pts_player = normal_rng(mean_player + beta_rest*48, sd_player);

  for (n in 1:N_observations) {
    mu_pred[n] = mean_player[player_idx[n]] + beta_rest[player_idx[n]]*rest_hours[n];
  }
  vector[N_observations] sd_pred = sd_player[player_idx];
  array[N_observations] real pts_pred = normal_rng(to_array_1d(mu_pred) , to_array_1d(sd_pred));


  real mean_rookie;
  mean_rookie = normal_rng(mean_general + mean_rest*48, sd_general);


  // For LOO
  vector[N_observations] log_lik;
  for (n in 1:N_observations) {
    log_lik[n] = normal_lpdf(pts[n] | mean_player[player_idx[n]] + beta_rest[player_idx[n]]*rest_hou
  }
}
```