A Bayesian analysis of predicting the electrical output of a combined cycle power plant

Khang Nguyen, Son Le

Abstract

This report presents a Bayesian workflow on modeling electrical output of a combined cycle power plant. First, this report provides an overview of the dataset and the analysis problem. Then, we implement a linear model and a generalized additive model (GAM) using R's brms package which uses Stan as the backend. It was discovered that the GAM is more suitable to the data than the linear model, indicated by the better performance in posterior predictive checks and approximate leave-one-out cross validation.

Contents

1 Introduction

Traditionally, thermodynamic systems such as thermal power plants have been modeled and simulated using many mathematical nonlinear equations which are often challenging to construct and computationally hard to solve (Liu et al., 2012). While these systems of equations can provide a deep understanding of the behavior of a thermodynamic system, they require complete knowledge of the numerous components of the system. On the other hand, developing a model for a power plant plays an important role in reducing pollution generated by the plant and optimizing its operational efficiency, thus maximizing the profit for the owners.

To overcome the challenges of thermodynamic approaches for modeling thermal power plants, many researchers have applied machine learning methods such as artificial neural networks (e.g., Smrekar et al, 2010; Tüfekci, 2014). However, these machine learning methods, being frequentist approaches, usually do not take into account the quantification of the uncertainty of the results and the domain knowledge that the researcher has of the problem. This report presents a full Bayesian approach to modeling the electrical energy output of a combined cycle power plant, a type of thermal power plant.

Estimating the relationships between a response variable and explanatory variables is known as regression analysis. This report concerns two Bayesian regression models for estimating the relationship between the full-load output of a combined cycle power plant (CCPP) and a few ambient conditions of the plant. A detailed description of the analysis problem is provided below.

This report presents a Bayesian workflow in building a linear model and a generalized additive model to model the electrical output of a CCPP. The linear model models the response variable, the electrical output of the CCPP, as a linear combination of the explanatory variables, the ambient conditions of the plant. However, the relationship between the response variable and explanatory variables are unknown. This motivates a generalized additive model (GAM). In a GAM, the response variable (or a function of it) depends on a linear combination of functions of explanatory variables. These functions can apply flexible, nonlinear transformations of the explanatory variables, allowing for the estimation of a possibly nonlinear relationship between the response variable and the explanatory variables.

We build these two models with the `brms` R-language package which uses the Stan probabilistic programming framework as the backend. Stan uses the Markov Chain Monte Carlo (MCMC) and Hamiltonian Monte Carlo (HMC) algorithm, which is a flexible algorithm in fitting Bayesian models, on which we can do Bayesian inference by including our domain knowledge into the problem as priors and describing the uncertainty of the results.

The structure of the remaining parts of the report is as follows. In section 2, we provide a description of the dataset and the analysis problem. In sections 3 and 4, we present in detail the linear model and the generalized additive model (GAM) respectively. For each section, we:

- describe in detail the model (linear model in section and GAM in section 4)
- present the priors for the model and how we fit it,
- present the MCMC and HMC convergence diagnostics,
- show the posterior predictive checks to check the goodness of fit of the model to the data,
- conduct leave-one-out cross-validation,
- assess the predictive performance of the model,
- and perform sensitivity analysis to determine whether the choice of priors greatly influence posterior inference

In section 5, we compare the linear model with the GAM and determine which model is better. In the last section, we discuss the issues we encountered and potential improvements, provide a conclusion of the data analysis, and reflect on what we learned while making the project.

2 Description of the dataset and the analysis problem

2.1 Dataset description

The dataset on the combined cycle power plant (CCPP) was obtained from the University of California Irvine Machine Learning Repository. The data was donated to the repository by Tüfekci (2014) and Kaya et al. (2012), who retrieved the dataset from a CCPP which collected the data during six years (674 days) of working on full load. Originally, the dataset was in 674 different files, but the authors merged all of them into one file, shuffled the file, and donated the merged file to the repository. Tüfekci created the merged dataset in order to run several machine learning methods with cross-validation to find the best machine learning method for the task of predicting full-load electrical power output of the CCPP (Tüfekci, 2014). Our analysis differs from theirs in the way that they used frequentist machine learning methods and we did a full Bayesian analysis.

According to Tüfekci (2014), the CCPP, whose name was not revealed, has a nameplate capacity of 480 MW and features two 160-MW gas turbines and one steam turbine. In a CCPP, electricity is generated by a combination of heat-generating turbines. Tüfekci described that a gas turbine is sensitive to atmospheric pressure (AP), ambient temperature (AT), and relative humidity (RH). Meanwhile, a steam turbine is sensitive to exhaust vacuum (V). In this report, these ambient conditions are used as explanatory variables to explain the variance in the response variable, the net hourly electrical energy output (PE) of the plant. A snapshot of the dataset is shown in the table below.

```
dat <- read_excel("data/Folds5x2_pp.xlsx")
knitr::kable(head(dat), caption = "The combined cycle power plant data.")
```

Table 1: The combined cycle power plant data.

AT	V	AP	RH	PE
14.96	41.76	1024.07	73.17	463.26
25.18	62.96	1020.04	59.08	444.37
5.11	39.40	1012.16	92.14	488.56
20.86	57.32	1010.24	76.64	446.48
10.82	37.50	1009.23	96.62	473.90
26.27	59.44	1012.23	58.77	443.67

According to Tüfekci (2014):

- The explanatory variable AP (atmospheric pressure) is measured in millibars and varies from 992.89 to 1033.30 mbar.
- The explanatory AT (ambient temperature) is measured in degrees Celsius and varies from 1.81 °C and 37.11 °C.
- The explanatory variable RH (relative humidity) is a percentage point and varies from 25.56% to 100.16%.
- The explanatory variable V (exhaust vacuum) is measured in centimeters of mercury and varies from 25.36–81.56 cmHg.
- The response variable PE (full-load electrical output) is measured in megawatts and varies from 420.26–495.76 MW.

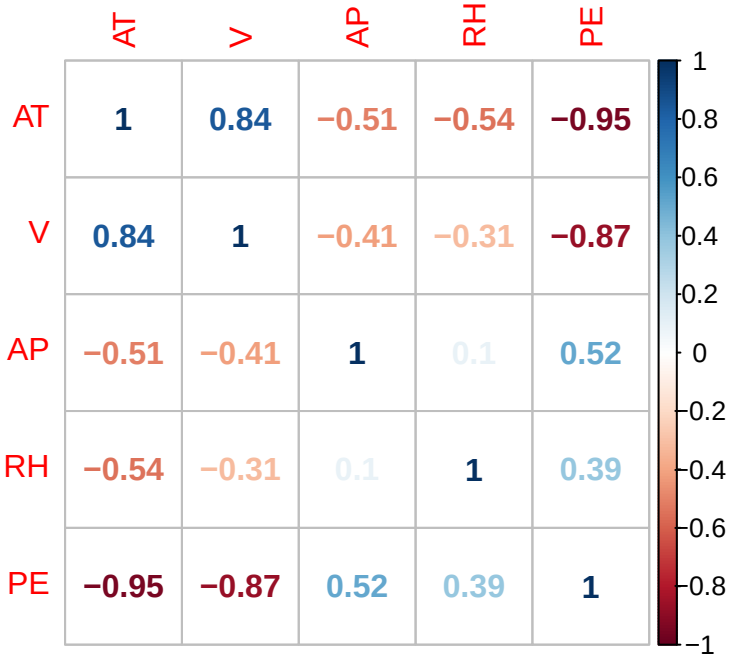
These ranges are confirmed in the table below.

```
knitr::kable(summary(dat), digits=1, caption="Some statistics of the data")
```

In total, there are 9568 data points. The structure of the dataset (not the structure of the datapoints themselves) is also simple: there are no prespecified groups since records from the 674 days have been

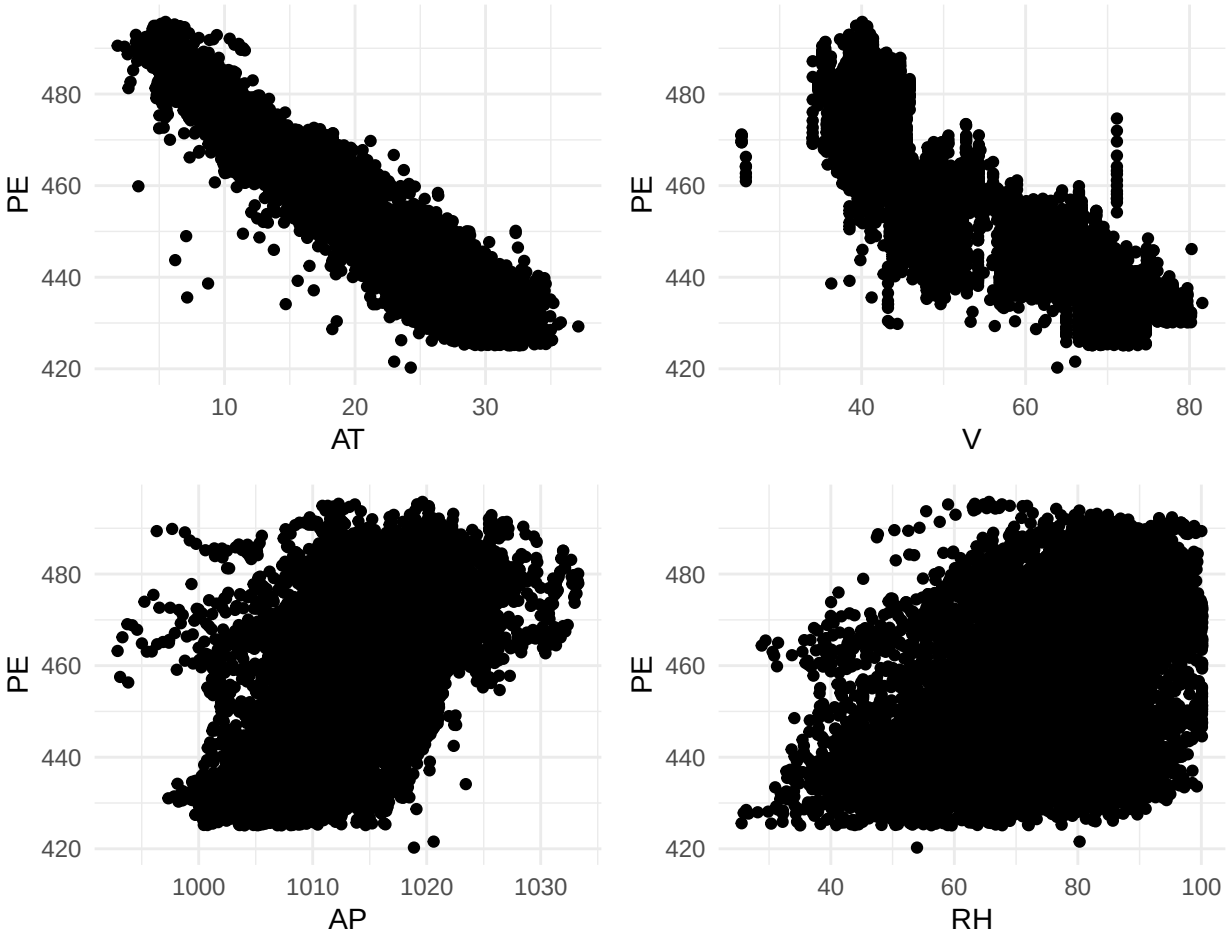
merged and shuffled; and there are only four explanatory variables to predict the (univariate) response variable. In the figure below, we show the correlations between different variables of the data.

```
corrplot::corrplot(cor(dat), method="number")
```



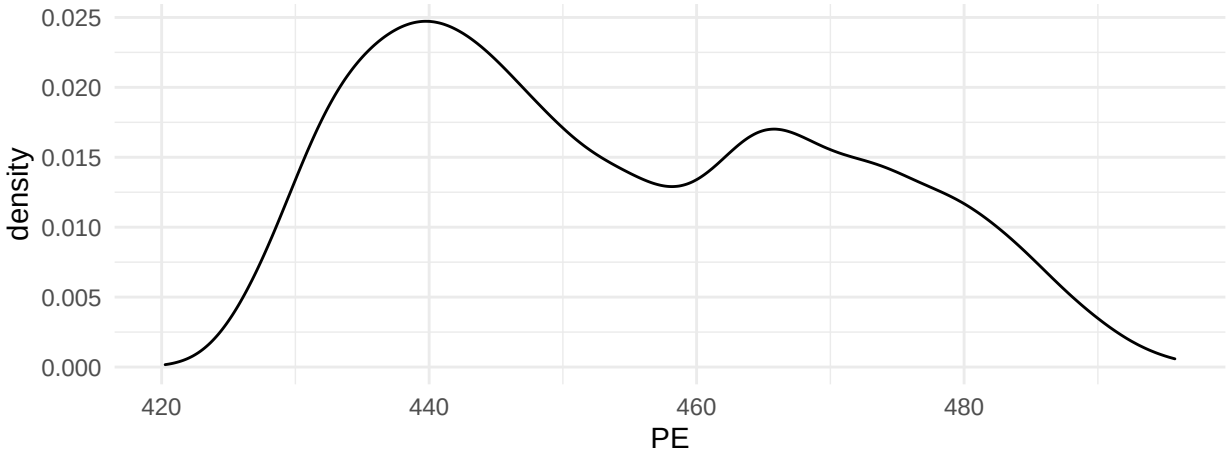
The response variable PE has strong negative correlations with the explanatory variables AT and V. In contrast, there is a positive correlation between AT and V. In general, we would expect that the higher AT and/or V are, the lower PE is. In the figures below, we show the pairwise scatterplots between pairs of response-explanatory variables

```
pairs <- GGally::ggpairs(dat)
gridExtra::grid.arrange(pairs[5,1], pairs[5,2], pairs[5,3], pairs[5,4])
```

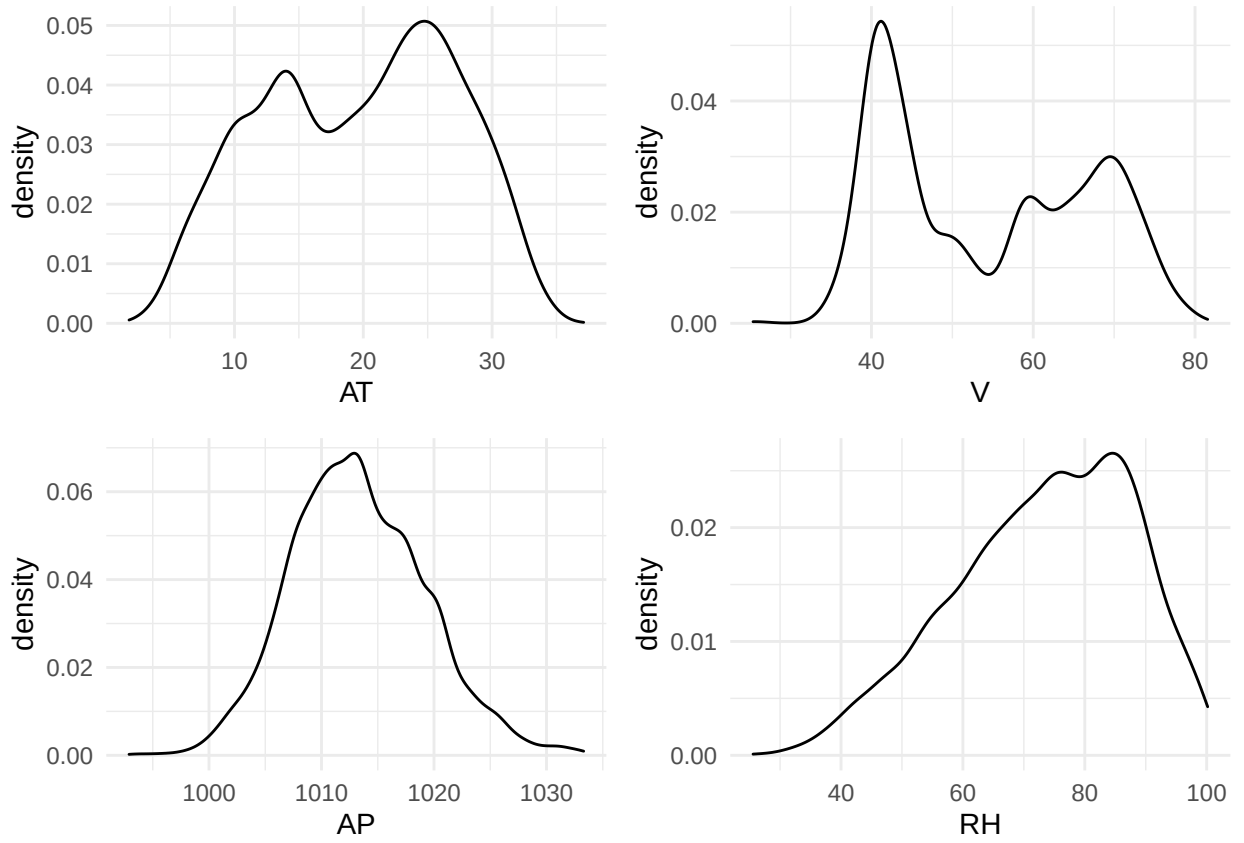


There seems to be a linear relationship between AT and PE as well as between AP and PE. Meanwhile, there seems to be a nonlinear relationship between V and PE. The relationship between RH and PE does not seem to follow any clear pattern. In the figures below, we show the densities of the variables.

```
pairs[5,5]
```



```
gridExtra::grid.arrange(pairs[1,1], pairs[2,2], pairs[3,3], pairs[4,4])
```



The density of interest is the density of PE, the response variable. This density has a quite complex shape with two peaks.

Because we had a lot of data to work with, it could take us a long time to fit the Stan models. Therefore, we decided to randomly take 80% of the data to fit the models. The remain 20% were used to evaluate the predictive performance of the models. We could split the dataset like this as the dataset does not have a group structure, time-dependent structure, or any other special structure. We split the dataset into two smaller sets: a `train` set and a `test` set.

```
train_test_split <- function(data, ratio=0.8) {
  set.seed(100)
  sample <- sample.split(data[, 1, drop=T], SplitRatio=ratio)
  data.train <- subset(data, sample == T)
  data.test <- subset(data, sample == F)
  rm(.Random.seed, envir=.GlobalEnv)
  list(data.train, data.test)
}
```

One problem with the data is that the variables are not on the same scale. Some variables vary very little compared to others. Thus, we decided to change the measurements of some variables so that they are on the same scale and/or range. By doing this, we believe we also help the Hamiltonian Monte Carlo (HMC) algorithm of Stan to sample more efficiently. The following conversions were performed:

- Exhaust vacumn V was converted from cmHg to mmHg; these all are units to measure pressure.
- Ambient temperature AT was converted from degrees Celsius to degrees Kelvin. This brings the range

of this variable closer to those of other variables.

- Relative humidity (RH), being percentage points, was converted to per-mille (‰) points.
- Atmospheric pressure AP and full-load electrical power output PE were unchanged.

```
dat_modified <-
  dat %>% mutate(
    V = V*10, # from cmHg to mmHg
    RH = RH*10, # from percent to per-mille (‰)
    AT = AT %>% conv_unit("C", "K") # from Celsius to Kelvin
  ) # %<-% is from from zeallot package
```

```
c(train, test) %<-% (train_test_split(dat_modified))
```

There are other methods of data transformation such as standardization or normalization. We decided against doing them because we want to keep the interpretability of the regression coefficients, and we did not want to change the scale of the response variable.

Table 2: Some statistics of the converted data

AT	V	AP	RH	PE
Min. :275.0	Min. :253.6	Min. : 992.9	Min. : 255.6	Min. :420.3
1st Qu.:286.7	1st Qu.:417.4	1st Qu.:1009.1	1st Qu.: 633.3	1st Qu.:439.8
Median :293.5	Median :520.8	Median :1012.9	Median : 749.8	Median :451.6
Mean :292.8	Mean :543.1	Mean :1013.3	Mean : 733.1	Mean :454.4
3rd Qu.:298.9	3rd Qu.:665.4	3rd Qu.:1017.3	3rd Qu.: 848.3	3rd Qu.:468.4
Max. :310.3	Max. :815.6	Max. :1033.3	Max. :1001.6	Max. :495.8

Table 3: Some statistics of the training data

AT	V	AP	RH	PE
Min. :275.0	Min. :253.6	Min. : 992.9	Min. : 255.6	Min. :420.3
1st Qu.:286.6	1st Qu.:417.4	1st Qu.:1009.0	1st Qu.: 633.4	1st Qu.:439.7
Median :293.5	Median :520.8	Median :1012.9	Median : 749.8	Median :451.7
Mean :292.8	Mean :543.1	Mean :1013.2	Mean : 733.4	Mean :454.4
3rd Qu.:298.9	3rd Qu.:665.4	3rd Qu.:1017.2	3rd Qu.: 849.5	3rd Qu.:468.7
Max. :310.3	Max. :802.5	Max. :1033.3	Max. :1001.6	Max. :495.8

Table 4: Some statistics of the testing data

AT	V	AP	RH	PE
Min. :278.1	Min. :253.6	Min. : 994.6	Min. : 288.1	Min. :425.1
1st Qu.:286.9	1st Qu.:417.3	1st Qu.:1009.2	1st Qu.: 633.0	1st Qu.:439.9
Median :293.5	Median :523.0	Median :1013.0	Median : 749.1	Median :450.8
Mean :292.9	Mean :542.8	Mean :1013.5	Mean : 731.6	Mean :454.0
3rd Qu.:298.8	3rd Qu.:664.9	3rd Qu.:1017.4	3rd Qu.: 842.6	3rd Qu.:467.6
Max. :307.0	Max. :815.6	Max. :1033.2	Max. :1001.5	Max. :495.4

2.2 Analysis problem

For this dataset, we want to fit regression models to predict the output variable by using explanatory variables. In other words, full-load electrical power output PE of the combined cycle power plant (CCPP) is predicted by the ambient temperature AT, the atmospheric pressure AP, the relative humidity RH, and the exhaust vacuum V as these explanatory variables have an effect on the generation of electricity in the CCPP, as described by Tüfekci (2014). The exploratory data analysis motivated us to build two models: a linear model and a generalized additive model (GAM).

Linear models are easy to implement and serve as baseline models in regression analysis. In brief, our linear model assumes that the relationship between the response variable and the explanatory variables is linear:

$$y_i = \mathbf{x_i}^T \beta + \epsilon_i, \quad i = 1, \ldots, n$$
$$\epsilon_i \sim N(0, \sigma)$$

where y_i is the response variable, $\mathbf{x_i}$ is a column vector of explanatory variables, n is the number of data points, β is a vector of regression coefficients (weights), and ϵ_i is an error term. Our model assumes that the errors are independent and identically distributed (i.i.d.) and have a normal distribution with mean 0 and standard deviation σ. The normal distribution is parameterized by the mean and the standard deviation. More detail on the linear model is provided in section 3.

The exploratory data analysis motivated us to implement a nonlinear model as in general, the relationship between the response variable and the explanatory variables do not seem to be linear. Since there are a lot of nonlinear models to choose from, it is difficult for us to decide on a specific nonlinear model (polynomials, exponentials, etc.). A class of models that can capture nonlinear relationships is generalized additive model (GAM). The high-level idea of a GAM is that we capture the impact of the explanatory variables on the response variable with smooth functions (of the explanatory variables) that "learn" the data and adapt their forms according to the data. We then add up these smooth functions to estimate the response variable. We provide a deeper explanation of our GAM in section 4.

3 Linear model

3.1 Model description

In this section, we will discuss the models that we have used to apply the Bayesian workflow with a linear model to modeling the electrical output of CCPP. Being a linear model, it has the following equation: $PE \sim AT+AP+P+RH$. By inputting the response and explanatory variables into `brms`'s bf function, we generated the model code in Stan with the help of `brms`. The inputted value was $brm(bf(PE \sim AT + AP + P + RH)$. This model was first built as a test of how to use `brms` but was chosen to be one of the models presented in this report due to decent performance.

`bf()` is the formula of the model; it tells `brm()` the relationship between the response variable and explanatory variables. As we described above, `PE` is a sum of the explanatory variables `AT`, `V`, `AP`, and `RH` multiplied by weights.

3.2 Priors

Viewing the default priors

```
prior = get_prior(PE ~ ., train, family = gaussian())
knitr::kable(prior, caption="Default prior for linear model")
```

Table 5: Default prior for linear model

prior	class	coef	group	resp	dpar	nlpar	bound	source
	b							default
	b	AP						default
	b	AT						default
	b	RH						default
	b	V						default
student_t(3, 451.7, 20.7)	Intercept							default
student_t(3, 0, 20.7)	sigma							default

For this linear model, we tested two different sets of weakly, informative priors.

The first set of priors is:

$$
\begin{aligned}
w &\sim N(0,1) \\
b &\sim Student - t(3,0,1) \\
\sigma &\sim Student - t(3,0,1)
\end{aligned}
$$

According to Stan developer guide to prior choice (Stan development team, 2020), both $N(0,1)$ and $Student-t(3,0,1)$ are generic weakly informative priors that can be used for pretty much anything. Since this set of priors was used for the first run of the linear model, we figured some generic priors would be the best choice.

The second set of priors is:

$$
\begin{aligned}
w &\sim N(0,5) \\
b &\sim N(450,50) \\
\sigma &\sim exponential(0.05)
\end{aligned}
$$

This set of priors was chosen after we had gained some knowledge of the data. The $N(0,5)$ for coefficients was chosen based on the trends of the explanatory variables. For σ, $exponential(0.05)$ was used to minimize the error. Lastly, for b, $N(450,50)$ was the choice of prior because it fits with the distribution of PE. According to (Tüfekci, 2014), the PE value ranges from 420 to 495.

```
prior1 <- c(set_prior("normal(0,1)", class = "b", coef = "AP"),
            set_prior("normal(0,1)", class = "b", coef = "AT"),
            set_prior("normal(0,1)", class = "b", coef = "RH"),
            set_prior("normal(0,1)", class = "b", coef = "V"),
            set_prior("student_t(3,0,1)", class = "Intercept"),
            set_prior("student_t(3,0,1)", class = "sigma"))

prior2 <-
  prior(normal(0, 5), class=b) +
  prior(exponential(0.05), class=sigma) +
  prior(normal(450, 50), class=Intercept)
```

3.3 Model fitting

To fit the actual model with `brms`, we used the following code:

```
fit1 <- brm(bf(PE ~ AT + V + AP + RH), train, refresh = 0, sample_prior = TRUE, prior = prior1,
            file="models/linear1", seed  = 123, control = list(adapt_delta = 0.97))
fit1 <- add_criterion(fit1,"loo", file="models/linear1")
```

```
fit2 <- brm(bf(PE ~ AT + V + AP + RH), train, refresh = 0, sample_prior = TRUE, prior = prior2,
            file="models/linear2", seed  = 123, control = list(adapt_delta = 0.97))
fit2 <- add_criterion(fit2,"loo", file="models/linear2")
```

We will go through the code and provide explanations as needed. Most of the arguments were omitted which means that they used the default values from stan.fit. Among the arguments of notice are `chains = 4`, `iter = 2000`, `warmup = floor(iter/2)`. This means 4 Markov chains were used, and each chain was run for 2000 iterations with 1000 iterations of warmup. Those mentioned default values work well for us, so we did not change anything.

Among the provided arguments are `refresh`, `prior`, `sample_prior`, `seed`, and `adapt_delta`.

- `refresh = 0` suppresses the printing of the training process which helps keep the report tidy.
- `prior` is to provide the priors for the Stan model.
- `sample_prior = TRUE` to make sure that the Stan model contains samples of priors distributions. This enables us to perform prior predictive checks if needed.
- `seed` is for reproducibility.
- Adapt_delta is one of the parameters that can be included in the optional Control list passed to Stan or the sampling function. It is for setting the target acceptance rate to avoid post-warmup divergences.
- File to save the results in a file specified by the argument

The add_criterion(fit,"loo") ensures that we can use LOO for our diagnostic check

The Stan code of the model is generated by `brms` with the following command:

```
code_brm_linear_1 <- stancode(fit1)
```

Since the Stan code for both models are the same (the only difference is the priors), we will only talk about the code for one model. In this case, it is the model with prior set 1.

```
// generated with brms 2.14.4
functions {
}
data {
  int<lower=1> N;  // total number of observations
  vector[N] Y;  // response variable
  int<lower=1> K;  // number of population-level effects
  matrix[N, K] X;  // population-level design matrix
  int prior_only;  // should the likelihood be ignored?
}
```

This part is the data declaration in Stan. Most of the parameters are given by the dataset itself. The only exception is the `prior_only` part which was given by the above Stan code. It is an indicator of whether to only sample from the prior (equals 1 if `sample_prior="only"`)

```
transformed data {
  int Kc = K - 1;
  matrix[N, Kc] Xc;  // centered version of X without an intercept
  vector[Kc] means_X;  // column means of X before centering
  for (i in 2:K) {
    means_X[i - 1] = mean(X[, i]);
    Xc[, i - 1] = X[, i] - means_X[i - 1];
  }
```

`Xc` is the matrix of coefficients without an intercept. `means_X` is used to center the said `Xc` matrix

```
parameters {
  vector[Kc] b;  // population-level effects
  real Intercept;  // temporary intercept for centered predictors
  real<lower=0> sigma;  // residual SD
}
transformed parameters {
}
```

`b` here is the weights that get to multiplied with the coefficients in the regression formula. `sigma` corresponds to σ, the standard deviation of the error terms. There are no `transformed parameters` for this model.

Next, we look at the `model` block.

```
model {
  // likelihood including all constants
  if (!prior_only) {
    target += normal_id_glm_lpdf(Y | Xc, Intercept, b, sigma);
  }
  // priors including all constants
  target += normal_lpdf(b[1] | 0,1);
  target += normal_lpdf(b[2] | 0,1);
  target += normal_lpdf(b[3] | 0,1);
  target += normal_lpdf(b[4] | 0,1);
  target += student_t_lpdf(Intercept | 3,0,1);
  target += student_t_lpdf(sigma | 3,0,1)
    - 1 * student_t_lccdf(0 | 3,0,1);
}
```

`normal_id_glm_lpdf` stands for Normal-Id Generalised Linear Model. This is actually linear regression in Stan code. The first line means the log normal probability density of `target` given `Intercept`+`Xc`*`b` and `sigma`, where the same `Intercept` and `sigma` is used for all observations

The lines after the first one are where the specified priors end up.

```
generated quantities {
  // actual population-level intercept
  real b_Intercept = Intercept - dot_product(means_X, b);
  // additionally draw samples from priors
  real prior_b_1 = normal_rng(0,1);
  real prior_b_2 = normal_rng(0,1);
  real prior_b_3 = normal_rng(0,1);
  real prior_b_4 = normal_rng(0,1);
  real prior_Intercept = student_t_rng(3,0,1);
  real prior_sigma = student_t_rng(3,0,1);
  // use rejection sampling for truncated priors
  while (prior_sigma < 0) {
    prior_sigma = student_t_rng(3,0,1);
  }
```

In the `generated quantities` block, because the data has not been centered before running the Stan code, the "actual population-level intercept" is the `Intercept` sampled by Stan after removing the difference in the pre-centered coefficients * weights.

Since we set `sample_prior="TRUE"`, the `generated quantities` block also included some parameters that are sampled from the priors. They are useful if we want to conduct a prior predictive check.

3.4 Convergence diagnostics

All of these values were gotten in the first run of the models.

First, we check $\hat{R}$ values. $\hat{R}$ is a potential scale reduction factor. $\hat{R}$ measures the ratio of the mean within the variance of the chains to the variance of the pooled draws across chains. These are equal if all chains are at equilibrium, and $\hat{R} \sim 1.00$. Empirically, if $\hat{R}$ is larger than 1.05, it usually means that the sequences have yet to converge, and then we should run more iterations to reach convergence. For $\hat{R}$'s of both models, we can look at the summaries below. The $\hat{R}$ values for all parameters are close to 1, showing that the models have converged.

```
summary(fit1)
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: PE ~ AT + V + AP + RH
##    Data: train (Number of observations: 7724)
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 4000
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept  1003.17     13.59   976.99  1029.69 1.00     2239     2355
## AT           -1.98      0.02    -2.02    -1.95 1.00     2462     2271
## V            -0.02      0.00    -0.02    -0.02 1.00     3046     3042
## AP            0.06      0.01     0.03     0.08 1.00     2570     2535
## RH           -0.02      0.00    -0.02    -0.01 1.00     3432     3193
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     4.51      0.04     4.44     4.59 1.00     2876     2108
##
## Samples were drawn using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

```
summary(fit2)
```

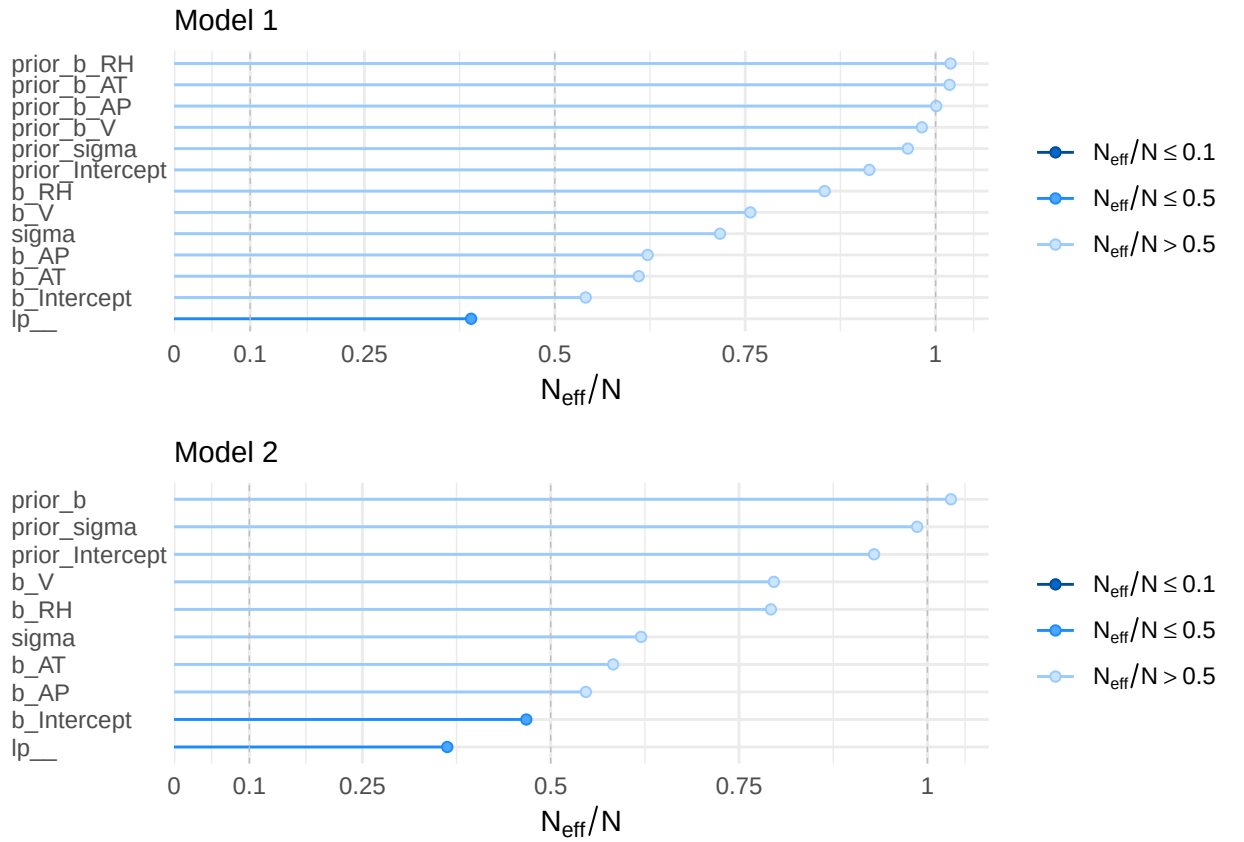
```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: PE ~ AT + V + AP + RH
##    Data: train (Number of observations: 7724)
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 4000
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept  1003.21     13.70   976.68  1029.96 1.00     1894     1969
## AT           -1.98      0.02    -2.02    -1.95 1.00     2355     2121
## V            -0.02      0.00    -0.02    -0.02 1.00     3209     2901
## AP            0.06      0.01     0.03     0.08 1.00     2189     1875
## RH           -0.02      0.00    -0.02    -0.01 1.00     3192     3220
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     4.52      0.04     4.45     4.59 1.00     2496     2292
```

```
##
## Samples were drawn using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

Next, we check the effective sample size n_{eff}. In Stan context, the drawn samples are usually autocorrelated within each chain; n_{eff} measures the number of explanatory samples with the same estimation quality as the total sample size. The higher the n_{eff}'s are, the better the samples are. By looking at the summaries above, we see that the n_{eff} values seem high enough. We could also inspect the ratio n_{eff}/N, where N is the total sample size:

```r
# reference for code: https://mc-stan.org/bayesplot/articles/visual-mcmc-diagnostics.html
compare_neff <- function(plot1, plot2, ncol = 2, ...) {
  bayesplot_grid(
    plot1, plot2,
    grid_args = list(ncol = ncol),
    subtitles = c("Model 1", "Model 2"),
    ...
  )
}

compare_neff(mcmc_neff(neff_ratio(fit1$fit)) +
               yaxis_text(hjust = 0),
             mcmc_neff(neff_ratio(fit2$fit)) +
               yaxis_text(hjust = 0),
             ncol = 1)
```



For this kind of particular ratio, values smaller than 0.1 indicate that something went wrong, values from

0.1 to 0.5 are good, and those larger than 0.5 are high. In the plots, we see that the n_{eff}'s for all of the coefficients are all high. The only exception is `b_Intercept` from priors set 2. However, it is still good enough.

Now, we examine HMC-specific convergence diagnostics. We first look for divergence transitions, which mean that the HMC algorithm has problems sufficiently exploring the neighborhoods of the posterior distribution around those transitions.

```
rstan::check_divergences(fit1$fit)
```

```
## 0 of 4000 iterations ended with a divergence.
```

```
rstan::check_divergences(fit2$fit)
```

```
## 0 of 4000 iterations ended with a divergence.
```

We did not encounter any divergent transitions, which means that the HMC algorithm worked well. Finally, we check the tree depth. Stan sets an upper limit on the depth of the trees that the NUTS algorithm, a type of HMC used by Stan, evaluates in each iteration. An iteration that saturated the maximum tree depth means that the algorithm stopped early to avoid long running times (Stan Development Team, 2020).

```
rstan::check_treedepth(fit1$fit)
```

```
## 0 of 4000 iterations saturated the maximum tree depth of 10.
```

```
rstan::check_treedepth(fit2$fit)
```

```
## 0 of 4000 iterations saturated the maximum tree depth of 10.
```

We got none of the iterations with saturated the maximum tree depth, which is quite good enough.

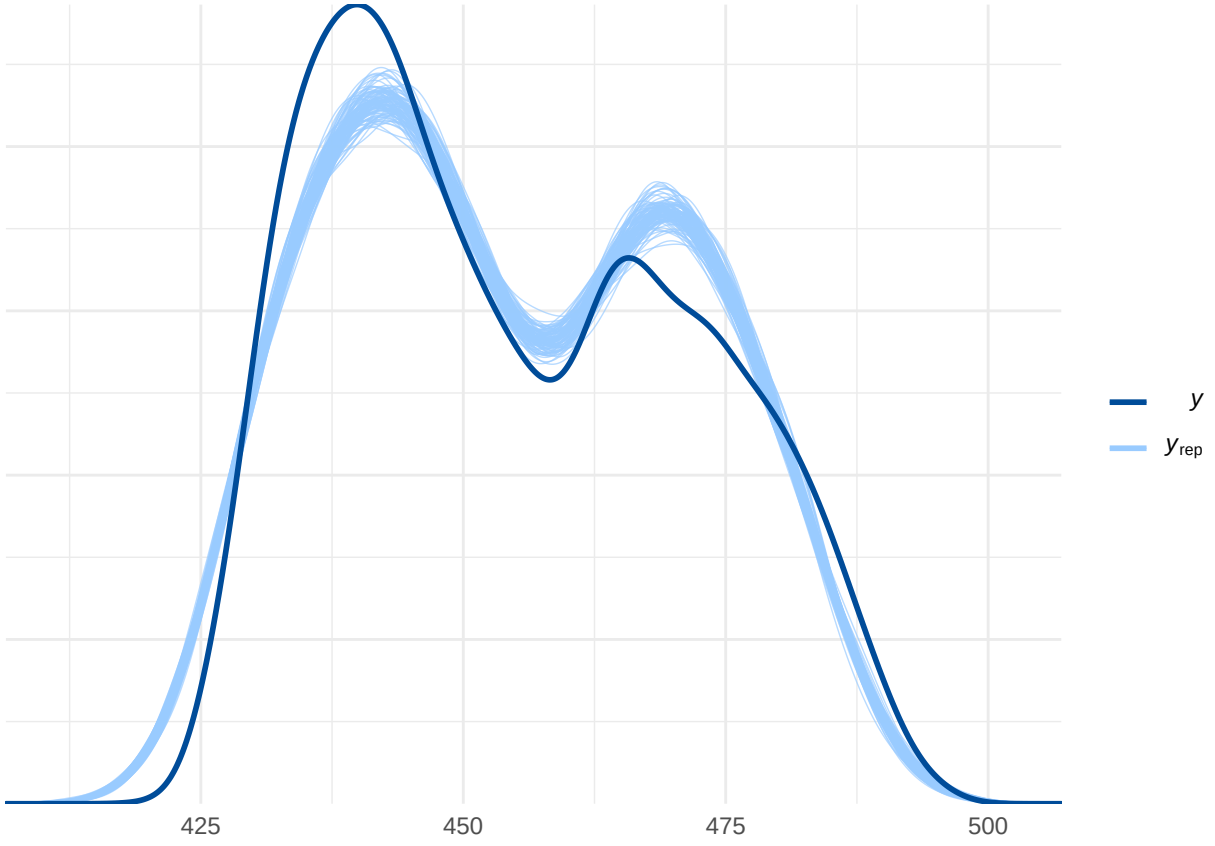
3.5 Posterior predictive checks

Although the checks above showed no problems with the model, how well did the model describe the data? Posterior predictive checking helps us evaluate the model; if the model was a good fit, we should be able to simulate the data or other posterior quantities of interest and obtain similar results to the observed data (Gabry et al., 2019). We can simulate the data used for posterior predictive checks by sampling from the posterior predictive distribution:

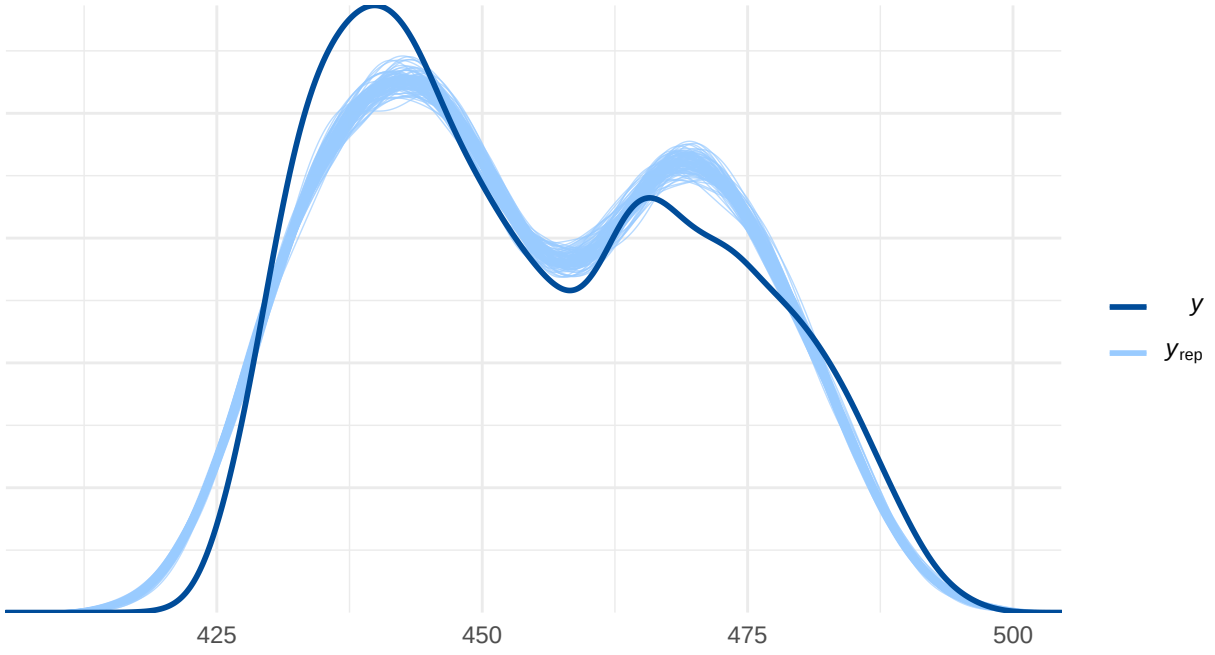
$$p(\widetilde{y} \,|\, y) = \int p(\widetilde{y} \,|\, \theta)\, p(\theta \,|\, y)\, d\theta$$

where θ are the model parameters, $\widetilde{y}$ are the data to be simulated (simulated `PE`, or `yrep` for replication), and y is the observed data. We usually also condition on explanatory variables. First, we compare the distribution of observed `PE` (`y`) to the distributions of 100 replicated datasets (`yrep`), for each prior set.

```
pp_check(fit1, type = "dens_overlay", nsamples = 100)
```



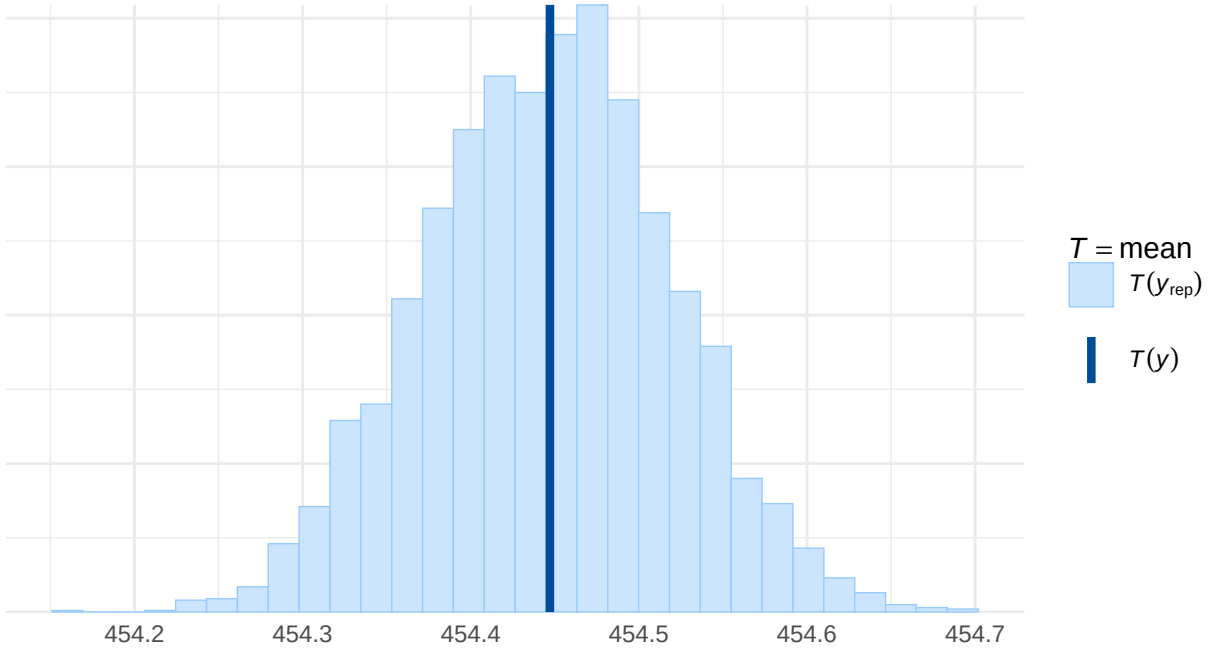

```
pp_check(fit2, type = "dens_overlay", nsamples = 100)
```



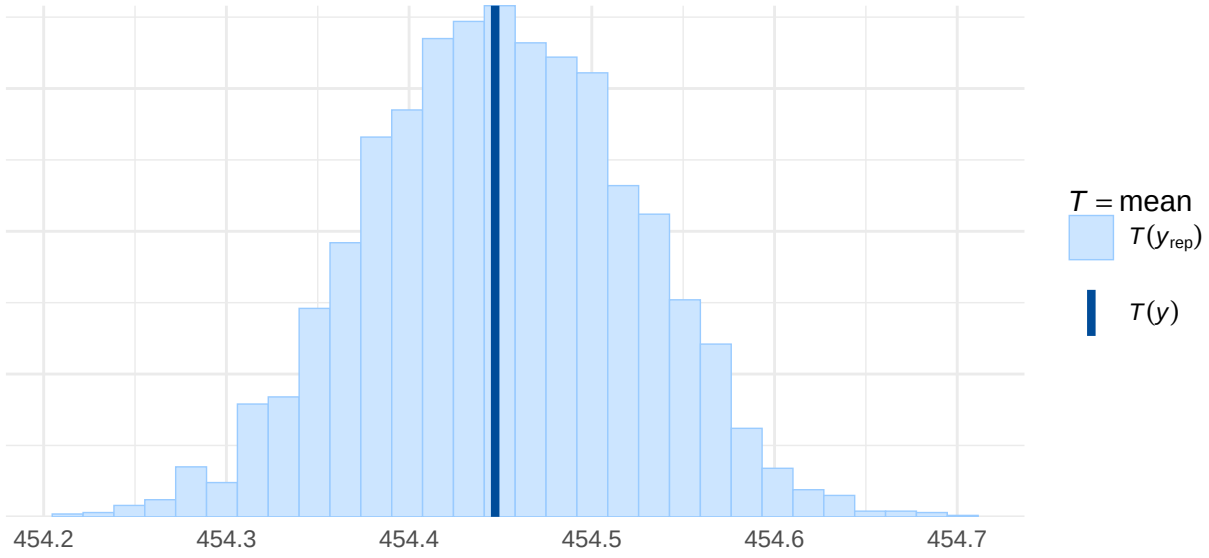
Despite not fitting exactly to the kernel density estimate, the posterior predictive replicates of both models are similar enough. To see how well the models capture the means, we compare the mean of the observed data to the means of the replicated datasets. We compute the means for all posterior predictive samples,

which number 4000 since we ran 4 chains, each with 2000 iterations, 1000 of which are warm-up iterations and are discarded when the posterior samples are extracted from the fit object.

```
pp_check(fit1, type = "stat", stat = 'mean', nsamples = 4000)
```



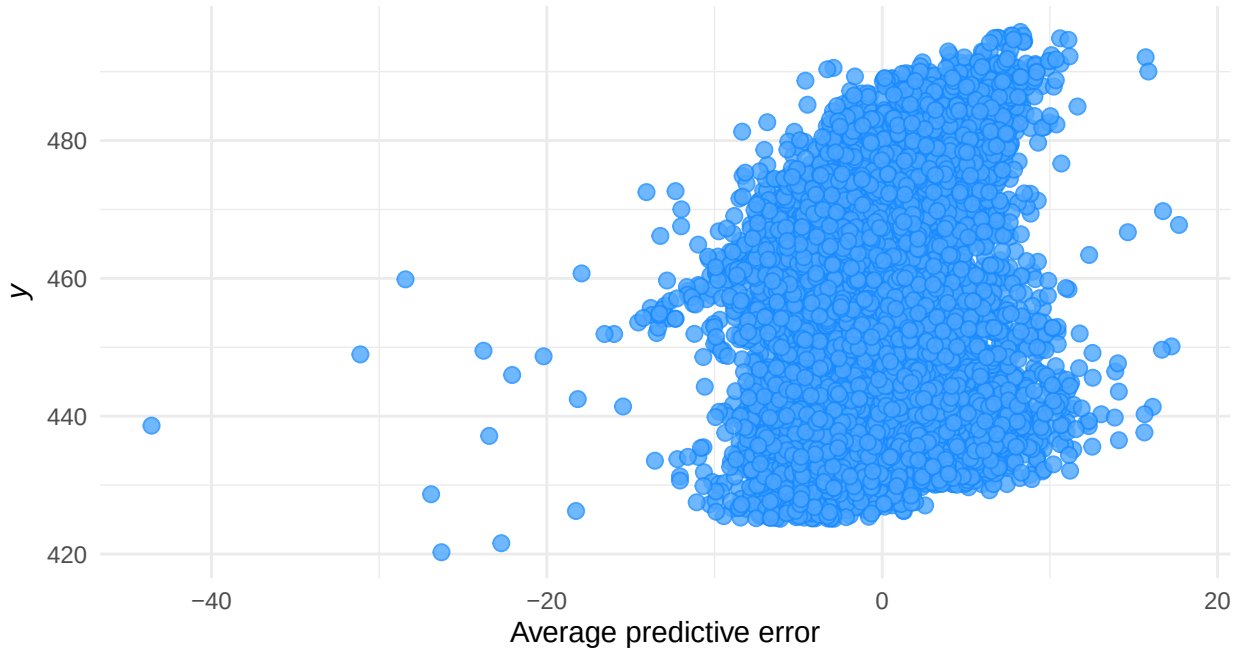

```
pp_check(fit2, type = "stat", stat = 'mean', nsamples = 4000)
```



We can clearly see that the means of the replicated datasets `yrep` are distributed with the center being the observed mean. We could also look at the scatterplot of the average predictive errors. For each observation y, the average predictive error is the average of the predictive errors corresponding to y computed over the samples from the posterior predictive distribution. We average overall posterior predictive samples.

```
pp_check(fit1, type='error_scatter_avg', nsamples = 4000)
```

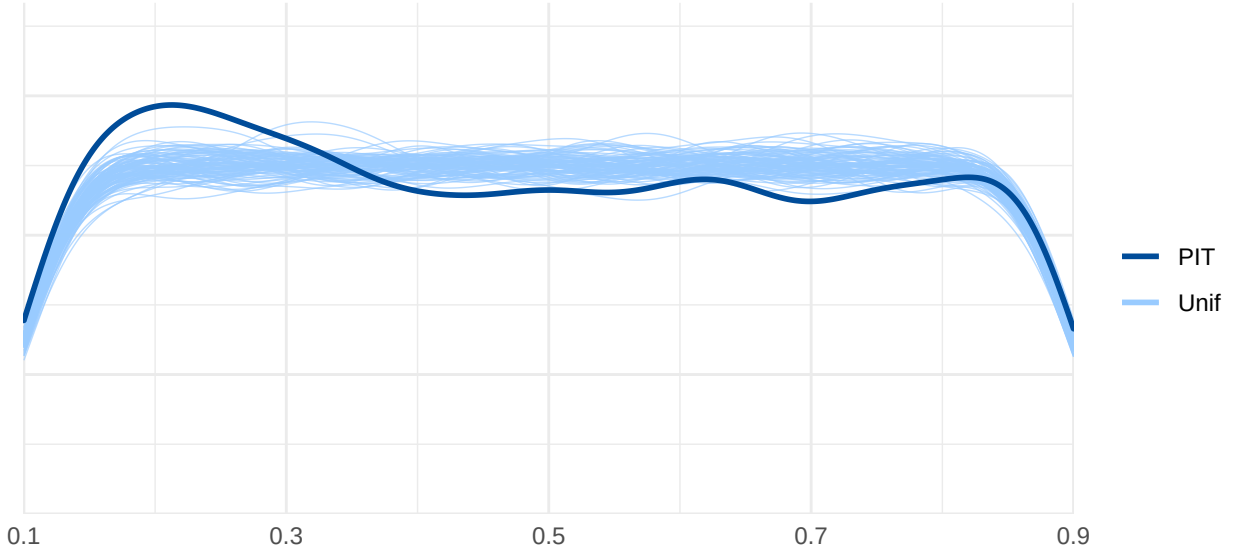

```
pp_check(fit2, type='error_scatter_avg', nsamples = 4000)
```



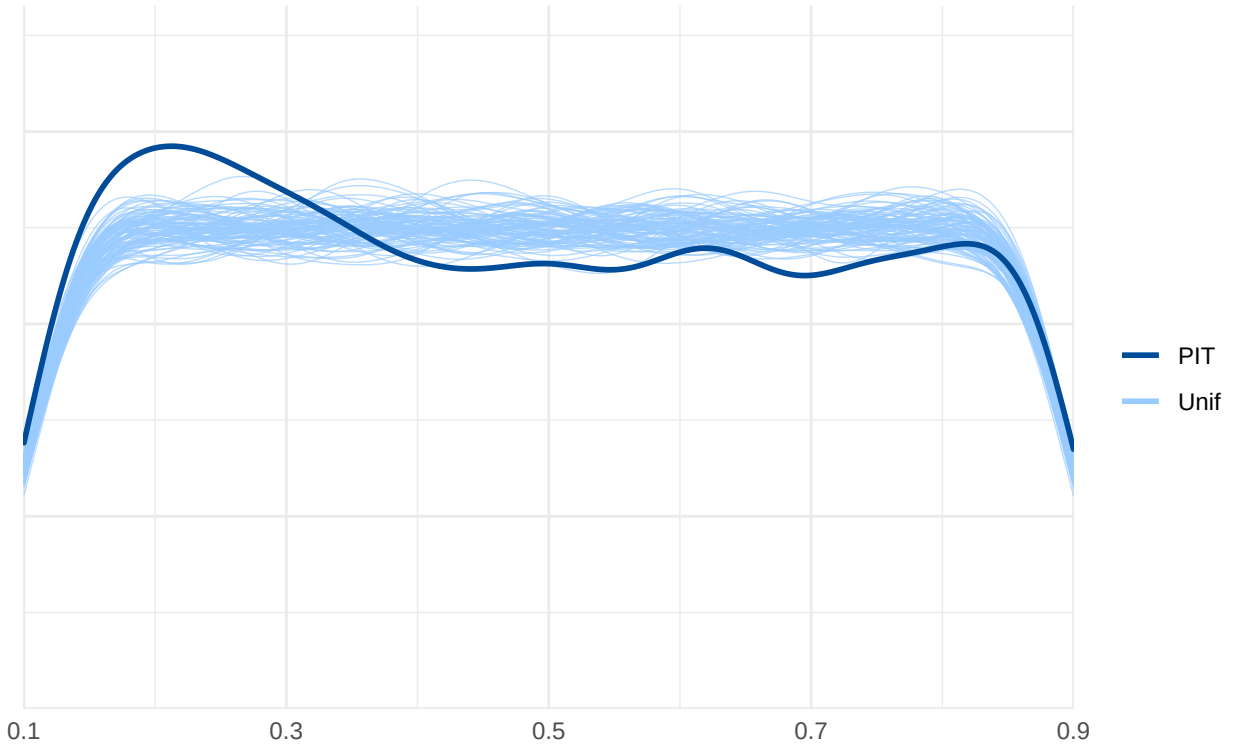
Even though most of the errors centered around 0 for Average predictive error, there are a few large errors. This shows that the linear models have not been that successful in learning the distribution of the response variable.

Leave-one-out (LOO) cross-validation (CV) partially avoids double use of data, and the test statistics are better calibrated (Gabry et al., 2019). We can check the calibration of marginal predictions with probability integral transformation (PIT) checks.

```
pp_check(fit1, type = "loo_pit_overlay", nsamples = 4000)
```




```
pp_check(fit2, type = "loo_pit_overlay", nsamples = 4000)
```



In the plot above, the thick curve is the density of computed LOO probability integral transforms, and the thin curves are simulations from the standard uniform distribution. If the model is calibrated, the thick curve should have a similar shape to the thin curves. We can see that the two models are decently calibrated.

3.6 Leave-one-out cross-validation

The linear models have been fitted and the posterior predictive checks show that the models are decent. However, we would like to measure the predictive accuracy of the models, which could be done by leave-one-out (LOO) cross-validation (CV). However, performing exact LOO-CV (leave one data point out at a time) demands refitting the models many times. An approximate method for LOO-CV is Parento smoothed importance sampling (PSIS) LOO-CV (Vehtari et al., 2017). A quantity that measures the predictive accuracy of the model is the expected log pointwise predictive density (ELPD) which can be estimated with PSIS-LOO. Below, we compute the Bayesian LOO estimate of ELPD `elpd_loo`. We also compute the estimate for the effective number of parameters `p_loo`, the difference between `elpd_loo` and non cross-validated log posterior predictive density, as well as `looic`, the LOO information criterion (`=-2*elpd_loo`).

```
loo1 <- loo(fit1)
loo2 <- loo(fit2)
knitr::kable(loo1$estimates, digits=1, caption="LOO estimates for linear model, prior set 1")
```

Table 6: LOO estimates for linear model, prior set 1

	Estimate	SE
elpd_loo	-22607.8	88.5
p_loo	7.2	0.7
looic	45215.6	177.0

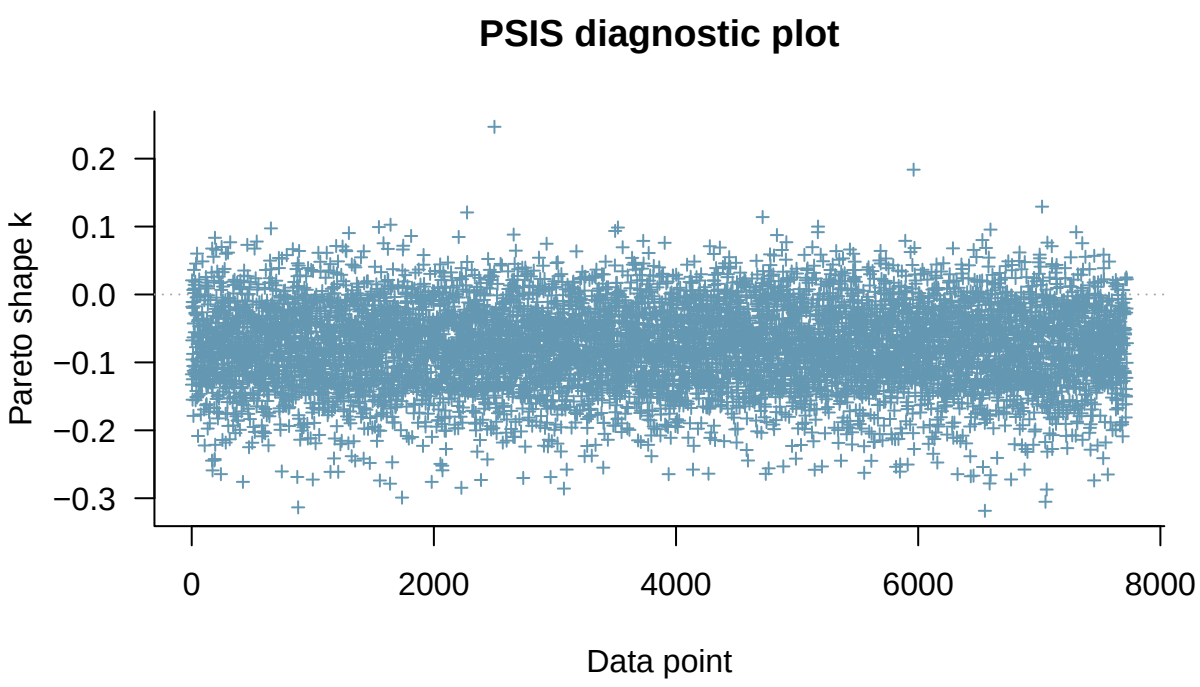
```
knitr::kable(loo2$estimates, digits=1, caption="LOO estimates for linear model, prior set 2")
```

Table 7: LOO estimates for linear model, prior set 2

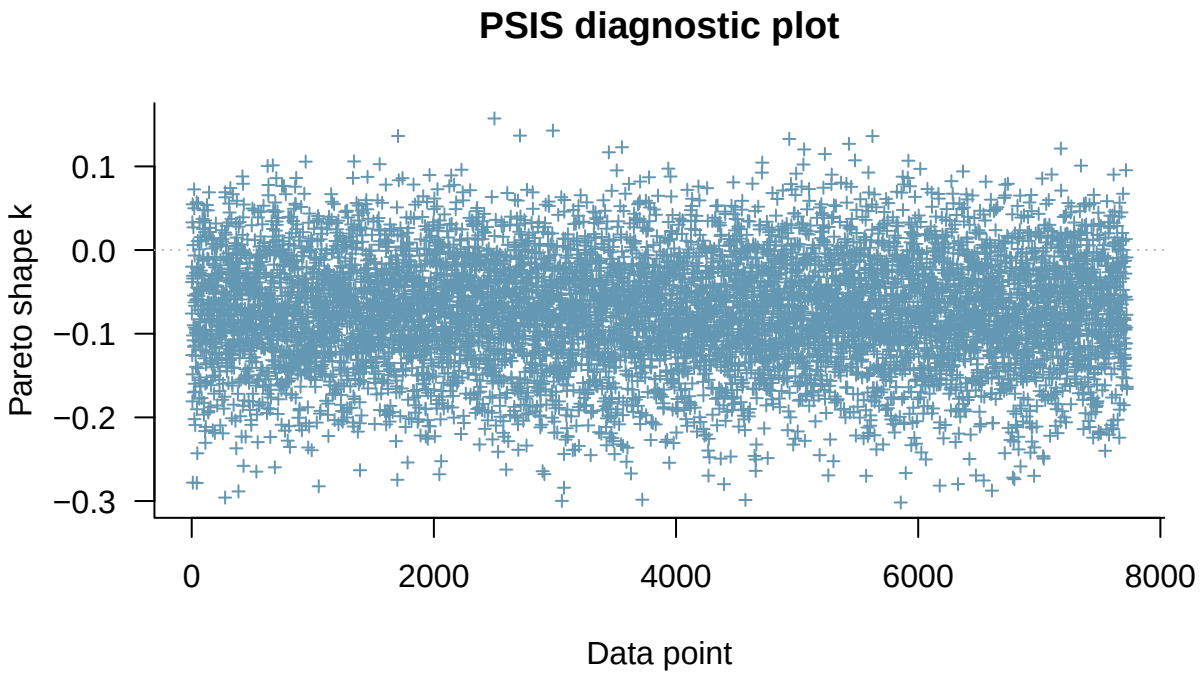
	Estimate	SE
elpd_loo	-22607.7	88.5
p_loo	7.1	0.7
looic	45215.5	176.9

To estimate the reliability of PSIS-LOO estimates, we compute the Pareto $\hat{k}$ estimate for each data point. If $\hat{k} < 0.5$, the corresponding component of PSIS_LOO estimates is reliable and accurate. If $0.5 \leq \hat{k} \leq 0.7$, the estimate can still be considered to be reliable. If $\hat{k} > 0.7$, the model may be biased and misspecified; the PSIS-LOO estimates for the model is unreliable, too optimistic, and overestimating the predictive accuracy of the model.

```
plot(loo1)
```




```
plot(loo2)
```



K values diagnostic plot show that all k values are low (< 0.7). This means that the above PSIS-LOO estimates for the linear model should be reliable.

3.7 Sensitivity analysis

Comparing the two linear models together using LOO, we can see that using prior set 2 is marginally better than using prior set 1. This makes sense since the 2nd set of priors were created based on the data and their trends while the 1st set of priors only consists of generic weakly informative priors.

```
comp <- loo_compare(loo1, loo2)
print(comp)
```

```
##      elpd_diff se_diff
## fit2  0.0       0.0
## fit1 -0.1       0.1
```

3.8 Predictive performance

We try assessing the predictive performance of the model on the `test` data. The two metrics that we will use are the Bayesian R^2 (see Gelman et al., 2019 for more information) and the root mean square error (RMSE). We calculate RMSE by using posterior samples of the expected value of the posterior predictive distribution evaluated on `test` data, for each posterior draw of the parameter θ^s. That is, the number of RMSE values that we calculate equals the number of posterior draws, which is 4000.

```
pp_rmse <- function(fit, newdata, ytrue=newdata$PE) {
  yrep <- posterior_epred(fit, newdata=newdata)
  rmse <- function(y) sqrt(mean((y - ytrue)^2))
  rmses <- as.data.frame(apply(yrep, MARGIN=1, rmse))
  colnames(rmses) <- "RMSEs"
  p1 <- mcmc_hist(rmses, binwidth=0.002) +
    yaxis_text() + ylab("Frequency")
  print(p1)
  # Estimate is the mean
  knitr::kable(posterior_summary(rmses), digits=3, caption="Estimates for RMSE")
}

bayes_r2 <- function(fit, newdata) {
  r2s <- bayes_R2(fit, newdata=test, summary=F)
  p1 <- mcmc_hist(r2s, binwidth=0.0001) +
    yaxis_text() + ylab("Frequency")
  print(p1)
  # Estimate is the mean
  knitr::kable(posterior_summary(r2s), digits=5, caption="Estimates for R2")
}
```

```
pp_rmse(fit2, test)
```

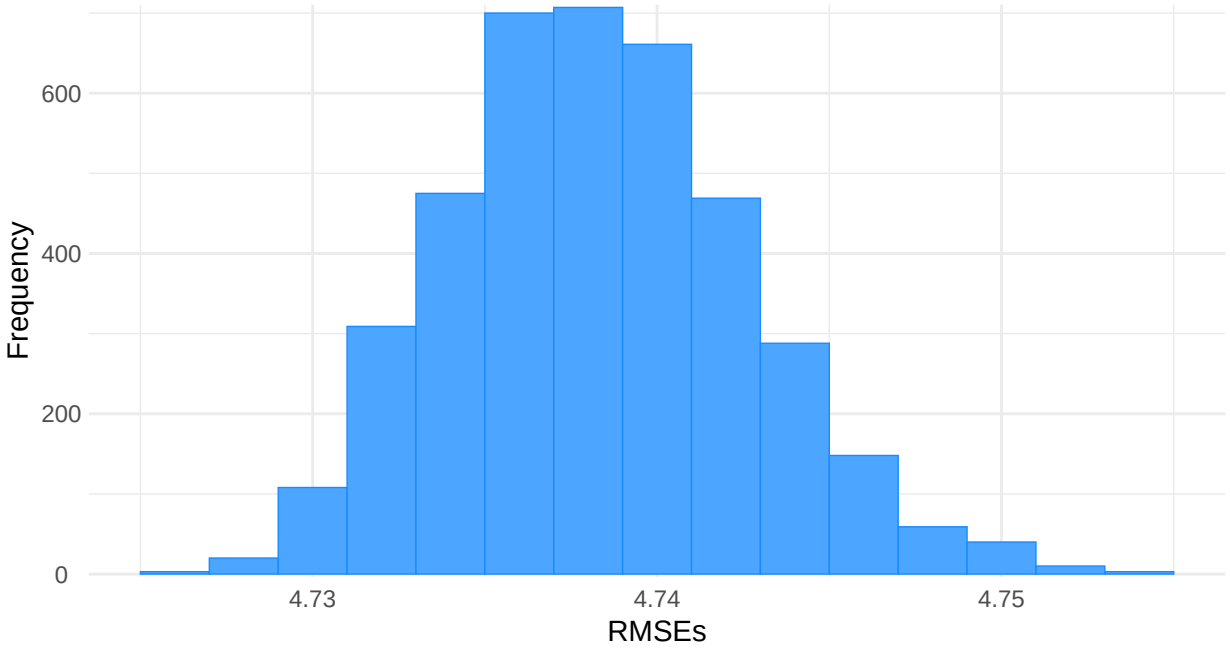


Table 8: Estimates for RMSE

	Estimate	Est.Error	Q2.5	Q97.5
RMSEs	4.738	0.004	4.731	4.747

```
bayes_r2(fit2, test)
```

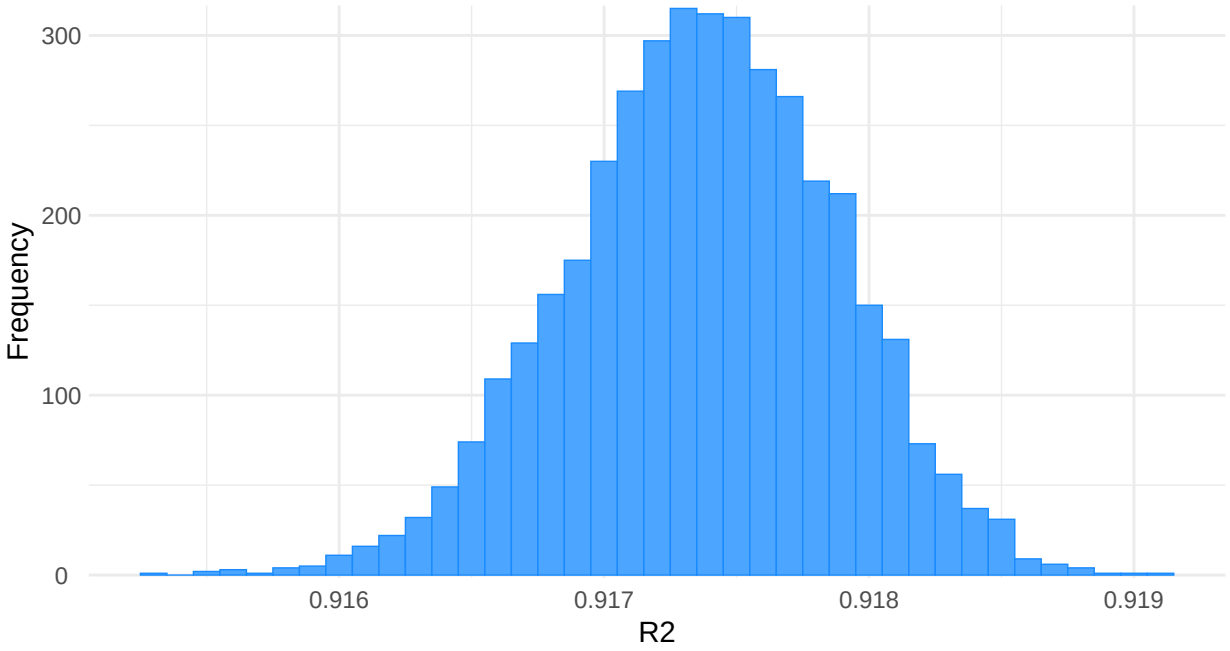


Table 9: Estimates for R2

	Estimate	Est.Error	Q2.5	Q97.5
R2	0.91737	0.00051	0.91636	0.91833

These statistics are not too useful as they are evaluated on a test dataset of arbitrary size, thus increasing the uncertainty of the statistics. If future datasets have the same size as the test set, then the statistics computed above can be considered useful; however, as we do not know the size of future sets, the uncertainty of the predictive performance of the model, measured by those statistics, is arbitrary.

4 Generalized Additive Model

4.1 Model description

Even though the linear model serves as a good baseline model, in most regression problems, it is usually the case that the relationship between the response variable and the explanatory variables is nonlinear, which could not be correctly captured by linear models or generalized linear models. Polynomial regression is a potential method to estimate such nonlinear relationships. However, polynomial regression starts to fail when the nonlinear relationships become complex as, e.g., it may overfit in some regions of the data and underfit in other regions.

Generalized additive models (GAMs) are models that could be used to identify nonlinear relationships. A brief explanation of GAMs is as follows. In a simple GAM, the relationships between individual explanatory variables and the response variable are captured through smooth functions which can be nonlinear. We get the response variable by adding up these smooth relationships:

$$g(y) = f(x_1) + f(x_2) + f(x_3) + \cdots + f(x_d) + \epsilon$$

where y is the response variable, x_i are the explanatory variables, each f is a smooth function of the explanatory variables, and ϵ is the error term which is assumed i.i.d. and $N(0, \sigma)$ distributed. For this explanation, we choose g to be the identity function ($g(y) = y$) and consider only one explanatory variable x to simplify the explanation. We can write the model as follows:

$$\begin{aligned} y &\sim N(\mu, \sigma) \\ \mu &= f(x) \end{aligned}$$

A smooth function $f(x)$ can be represented as a linear combination of transformations of x in some basis that we choose.

$$f(x) = \sum_{j=1}^{d} B_j(x)\beta_j$$

where each B_j is a basis function (transformation) of x in the chosen basis. Representing $f(x)$ as a sum of these basis functions is called a basis expansion. An example is the quadratic polynomial basis:

$$\mu = f(x) + \epsilon = \beta_0 + beta_1 x + \beta_2 x^2$$

If there are, e.g., two explanatory variables, and the response variable is modeled as a sum of two quadratic polynomial of each variable:

$$\begin{aligned}\mu &= f(x_1) + f(x_2) \\ &= \beta_0 + \beta_1 x_1 + \beta_2 x_1 + \gamma_0 + \gamma_1 x_2 + \gamma_2 x_2^2 \\ &= \beta_0 + \gamma_0 + \beta_1 x_1 + \gamma_1 x_2 + \beta_2 x_1 + \gamma_2 x_2^2\end{aligned}$$

Splines are one type of smooth function. A spline can be thought of as a piecewise polynomial. The points that connect the polynomial pieces are called knots, which may have to be manually selected. There are many kinds of splines. In this report, we use the thin plate spline basis for the smooth functions as it is the default for smooth terms of `brms` and it helps us avoid the knot selection problem. There are a lot of things going on with thin plate splines (and splines in general), but in this report we can simply consider them as one type of spline and that the thin plate spline has a basis expansion similar to the mathematical notations above.

Thus, simply put, a GAM allows us to fit a nonlinear model through linear representation of the smooth terms in some basis. In this report, we fit a simple GAM, which we dub "nonlinear" in the R code:

$$\begin{aligned}\mu &= s(x_1) + s(x_2) + s(x_3) + s(x_4) + \text{Intercept} \\ y &\sim N(\mu, \sigma)\end{aligned}$$

where:

- s is the smooth terms with the thin plate spline basis,
- Intercept is the value of the response variable when all smooth terms equal zero,
- each x_i is one of the four explanatory variables (AT, V, AP, RH),
- and σ is the standard deviation of the error (can also be interpreted as the standard deviation of the response variable).

As we shall see later, `brms` internally represents the formula as follows:

$$\begin{aligned}\mu &= \text{Intercept} + s(x_1) + s(x_2) + s(x_3) + s(x_4) \\ &= \text{Intercept} + (\beta_1 f(x_1) + \mathbf{Z_1}\gamma_1) + (\beta_2 f(x_2) + \mathbf{Z_2}\gamma_2) + (\beta_3 f(x_3) + \mathbf{Z_3}\gamma_3) + (\beta_4 f(x_4) + \mathbf{Z_4}\gamma_4) \\ &= \text{Intercept} + [f(x_1), f(x_2), f(x_3), f(x_4)]\beta + \mathbf{Z_1}\gamma_1 + \mathbf{Z_2}\gamma_2 + \mathbf{Z_3}\gamma_3 + \mathbf{Z_4}\gamma_4\end{aligned}$$

where:

- each $\mathbf{Z_i}$ is a matrix of basis functions, the columns are basis functions, the rows are data points (only 1 row in this notation here as we are considering a single data point for this explanation);
- each $\gamma_\mathbf{i}$ is a column vector of coefficients for the basis functions;
- each $f(x_i)$ is some transformation of x_i automatically calculated by `brms` based on the data. Each $f(x_i)$ is called "linear effects" in the package as it can be interpreted as the linear, perfectly smooth part of the basis of $s(x_i)$.
- and β is a column vector of coefficients for each $f(x_i)$. By default in `brms`, β has length 4 ($[f(x_1), f(x_2), f(x_3), f(x_4)]$ has 4 elements by default).

The β_i are also known as "population-level effects" in `brms`. Each $\mathbf{Z_i}$ is automatically calculated by `brms` based on the data. The notations above may not be what exactly happens in `brms`; they are just simplified descriptions.

The mathematical notation for the GAM is as follows:

$$
\begin{aligned}
y &\sim N(\mu, \sigma) \\
\mu &= \text{Intercept} + [f(x_1), f(x_2), f(x_3), f(x_4)]\beta + \mathbf{Z_1}\gamma_\mathbf{1} + \mathbf{Z_2}\gamma_\mathbf{2} + \mathbf{Z_3}\gamma_\mathbf{3} + \mathbf{Z_4}\gamma_\mathbf{4} \\
\beta &= [\beta_1, \beta_2, \beta_3, \beta_4] \\
\gamma_\mathbf{i} &= [\gamma_{i1}, \gamma_{i2}, \ldots, \gamma_{i8}]^T \\
\gamma_{ij} &\sim N(0, \sigma_i), \quad i = 1, \ldots, 4, \quad j = 1, \ldots, 8 \\
\sigma_i &\sim \text{exponential}(0.1), \quad i = 1, \ldots, 4 \\
\beta_i &\sim N(0, 100), \quad i = 1, \ldots, 4 \\
\sigma &\sim \text{exponential}(0.05) \\
\text{Intercept} &\sim N(460, 75)
\end{aligned}
$$

Where each $\gamma_\mathbf{i}$ is a vector of length 8 because there are 8 columns in each basis function matrix $\mathbf{Z_i}$ (this is the default in `brms`). The last 5 lines are priors, the last 4 of which are manually set by us.

4.2 Priors

As noted above, the γ_{ij}'s are the coefficients for the basis functions. In `brms`, they have a prespecified prior:

$$\gamma_{ij} \sim N(0, \sigma_i), \quad i = 1, \ldots, 4, \quad j = 1, \ldots, 8$$

Where σ_i is the standard deviation of the "spline coefficients" whose prior is to be set by the user. An interpretation of σ_i is that it is used to control the smoothness and the flexibility of the fitted splines. The lower the value, the less flexible the smooth function. We set the prior for σ_i as follows:

$$\sigma_i \sim \text{exponential}(0.1)$$

since we would like the smooth functions to be flexible enough but not too flexible, i.e., too "wiggly", or too inflexible. We specify this prior in `brms` syntax as follows:

```
prior(exponential(0.1), class=sds)
```

The prior for the coefficients for the linear effects are weakly informative:

$$\beta_i \sim N(0, 100)$$

We set this prior on the β_i's as we do not want to impose a narrow prior on them and we believe they could be anything in the range of $N(0, 100)$-distributed values. We specify this prior in `brms` syntax as follows:

```
prior(normal(0, 100), class=b)
```

The prior for the standard deviation σ is:

$$\sigma \sim \text{exponential}(0.05)$$

This is also a weakly informative prior and corresponds nicely to the data as the range of exponential(0.05)-distributed values is not too wide or too narrow for the data. We specify this prior in `brms` syntax as follows:

```
prior(exponential(0.05), class=sigma)
```

For our GAM, the intercept has an intuitive meaning: it is the value of the response variable PE when all predictor terms are at their means, which in this case is zero as we discover that the transformations done by `brms` involve centering the explanatory variables. Thus, we set the following prior on the intercept:

$$\text{Intercept} \sim N(460, 75)$$

because we know a priori that the nominal capacity of the CCPP is 480 MW and we believe it is very unlikely for PE to be too low (under 250) or too high (over 600) when the CCPP is working with full-load. 460 is the value which we believe to be the mean of PE when all predictor terms are at their means. This value is also close to the mean of the observed PE, which is about 454. We specify this prior in `brms` syntax as follows:

```
prior(normal(460, 75), class=Intercept)
```

4.3 Model fitting and Stan code

The full code for the prior of the GAM is:

```
nonlinear.prior <-
  prior(normal(0, 100), class=b) +
  prior(exponential(0.1), class=sds) +
  prior(exponential(0.05), class=sigma) +
  prior(normal(460, 75), class=Intercept)
```

To fit the model with `brms`, we use the following code:

```
nonlinear.formula <- bf(PE ~ s(AT) + s(V) + s(AP) + s(RH))
nonlinear.fit <- nonlinear.formula %>%
  brm(data=train, prior=nonlinear.prior, file="models/nonlinear",
      seed=123, refresh=0, control=list(adapt_delta=0.95)) %>%
  add_criterion("loo", file="models/nonlinear")
```

We will walk through the code and provide explanations as needed. Similar to the linear model, we omitted most of the arguments and let `brm()` use the default values. Among the arguments of notice are `chains = 4`, `iter = 2000`, `warmup = floor(iter/2)`. This means 4 Markov chains were used, and each chain was run for 2000 iterations with 1000 iterations of warmup. Generally, we did not change the default values for the arguments much as these default values worked well for us.

`bf()` is the formula of the model; it tells `brm()` the relationship between the response variable and explanatory variables. As we described above, `PE` is a sum of univariate smooth functions of the explanatory variables `AT`, `V`, `AP`, and `RH`. The smooth functions have thin plate spline basis (the default).

Here, the arguments that we provided are explained above in the linear model section. We saved the model in a file specified by the argument `file`. The value for `adapt_delta` is also lower than for the linear model as we found this value worked well. A high `adapt_delta` is to avoid false-positive divergences. Like what we did for the linear model, `add_criterion("loo")` is the command for `brms` to perform PSIS-LOO cross-validation for the model.

We did not set `sample_prior=T` (the default is `F`) as it took about 1.5 hours just to run the model alone. If we needed to check the priors for this model, we could run another model with the option `sample_prior="only"` to only sample from the priors (and ignore the likelihood).

The Stan code of the model is generated by `brms` with the following command:

```
make_stancode(nonlinear.formula, prior=nonlinear.prior, data=train)
```

We now walk through the generated Stan code part by part.

```
// generated with brms 2.14.4
functions {
}
data {
  int<lower=1> N;  // total number of observations
```

```
  vector[N] Y;  // response variable
  // data for splines
  int Ks;  // number of linear effects
  matrix[N, Ks] Xs;  // design matrix for the linear effects
```

The number `Ks` is essentially the number of perfectly smooth parts of the four splines. Since we used the default options, the basis of each $s(x_i)$ will have one linear function. The value of `Ks` is 4. The matrix `Xs` is the matrix that concatenates these linear functions as columns.

```
  // data for spline s(AT)
  int nb_1;  // number of bases
  int knots_1[nb_1];  // number of knots
  // basis function matrices
  matrix[N, knots_1[1]] Zs_1_1;
```

The values for `nb_i` and `knots_i[nb_i]` are the same for all four splines. Since we used the default options, `nb_i=1` and `knots_i[1]=8`. `Zs_i_1` is the basis function matrix $\mathbf{Z_i}$ described above. Below is the rest of the `data` block (newlines were inserted for readability).

```
  // data for spline s(V)
  int nb_2;  // number of bases
  int knots_2[nb_2];  // number of knots
  // basis function matrices
  matrix[N, knots_2[1]] Zs_2_1;

  // data for spline s(AP)
  int nb_3;  // number of bases
  int knots_3[nb_3];  // number of knots
  // basis function matrices
  matrix[N, knots_3[1]] Zs_3_1;

  // data for spline s(RH)
  int nb_4;  // number of bases
  int knots_4[nb_4];  // number of knots
  // basis function matrices
  matrix[N, knots_4[1]] Zs_4_1;

  int prior_only;  // should the likelihood be ignored?
}
transformed data {
}
```

`prior_only` is an indicator of whether to only sample from the prior (equals 1 if `sample_prior="only"`). Next, we consider the `parameters` block.

```
parameters {
  real Intercept;  // temporary intercept for centered predictors
  vector[Ks] bs;  // spline coefficients
```

`Intercept` is the intercept term described above. `bs` is the β vector described above.

```
  // parameters for spline s(AT)
  // standarized spline coefficients
  vector[knots_1[1]] zs_1_1;
  real<lower=0> sds_1_1;  // standard deviations of spline coefficients
```

The parameters for the four splines are similar since we used default options. Each `zs_i_1` corresponds to the column vector of standardized coefficients for the basis functions (each column in `Zs_i_1`). The actual coefficients are in the `transformed parameters` block which will be explained right below. Each `sds_i_1` corresponds to the σ_i, the standard deviation of the spline coefficients $\gamma_{\mathbf{i}}$. Below is the remainder of the `parameters` block.

```
  // parameters for spline s(V)
  // standarized spline coefficients
  vector[knots_2[1]] zs_2_1;
  real<lower=0> sds_2_1;  // standard deviations of spline coefficients

  // parameters for spline s(AP)
  // standarized spline coefficients
  vector[knots_3[1]] zs_3_1;
  real<lower=0> sds_3_1;  // standard deviations of spline coefficients

  // parameters for spline s(RH)
  // standarized spline coefficients
  vector[knots_4[1]] zs_4_1;
  real<lower=0> sds_4_1;  // standard deviations of spline coefficients

  real<lower=0> sigma;  // residual SD
}
```

`sigma` corresponds to σ, the standard deviation of the error terms. Next, we inspect the `transformed parameters` block.

```
transformed parameters {
  // actual spline coefficients
  vector[knots_1[1]] s_1_1;
  // actual spline coefficients
  vector[knots_2[1]] s_2_1;
  // actual spline coefficients
  vector[knots_3[1]] s_3_1;
  // actual spline coefficients
  vector[knots_4[1]] s_4_1;
  // compute actual spline coefficients
  s_1_1 = sds_1_1 * zs_1_1;
  // compute actual spline coefficients
  s_2_1 = sds_2_1 * zs_2_1;
  // compute actual spline coefficients
  s_3_1 = sds_3_1 * zs_3_1;
  // compute actual spline coefficients
  s_4_1 = sds_4_1 * zs_4_1;
}
```

Each `s_i_1` corresponds to the column vector of coefficients for the basis functions (each column in `Zs_i_1`) and is computed by `s_i_1 = sds_i_1 * zs_i_1`. This essentially means that instead of using $\gamma_{ij} \sim N(0, \sigma_i)$, $\sigma_i \sim$ exponential(0.1) (the centered parameterization), `brms` uses the equivalent non-centered parameterization:

$$
\begin{aligned}
\gamma_{ij} &= zs_{ij} \times \sigma_i \\
zs_{ij} &\sim N(0,1), \quad i = 1, \ldots, 4, \quad j = 1, \ldots, 8 \\
\sigma_i &\sim \text{exponential}(0.1)
\end{aligned}
$$

Next, we look at the `model` block.

```
model {
  // likelihood including all constants
  if (!prior_only) {
    // initialize linear predictor term
    vector[N] mu = Intercept + rep_vector(0.0, N) +
                   Xs * bs + Zs_1_1 * s_1_1 + Zs_2_1 * s_2_1 +
                   Zs_3_1 * s_3_1 + Zs_4_1 * s_4_1;
    target += normal_lpdf(Y | mu, sigma);
  }
```

The likelihood is $\mathbf{Y} \sim N(\mu, \sigma)$. Instead of computing one data point at a time, this Stan code computes the likelihood once with all data points. `Y` and `mu` are column vectors of length `N`, the number of data points, and each matrix in the expression of `mu` has `N` rows. Finally, we look at the priors and the rest of the Stan code for the model:

```
  // priors including all constants
  target += normal_lpdf(Intercept | 460, 75);
  target += normal_lpdf(bs | 0, 100)
    - 4 * normal_lccdf(0 | 0, 100);
  target += exponential_lpdf(sds_1_1 | 0.1);
  target += std_normal_lpdf(zs_1_1);
  target += exponential_lpdf(sds_2_1 | 0.1);
  target += std_normal_lpdf(zs_2_1);
  target += exponential_lpdf(sds_3_1 | 0.1);
  target += std_normal_lpdf(zs_3_1);
  target += exponential_lpdf(sds_4_1 | 0.1);
  target += std_normal_lpdf(zs_4_1);
  target += exponential_lpdf(sigma | 0.05);
}
generated quantities {
  // actual population-level intercept
  real b_Intercept = Intercept;
}
```

we see that the priors that we specified in `brms` syntax did end up here. As mentioned above, the `zs_i_1`'s are standardized spline coefficients and have standard normal distribution as their prior. In the `generated quantities` block, because the data was already centered before running the Stan code, the "actual population-level intercept" is just the `Intercept` that was sampled by Stan.

4.4 Convergence diagnostics

After fitting the GAM, we validate the fit. First, we print the full summary of the fitting by using the information from the fitted object:

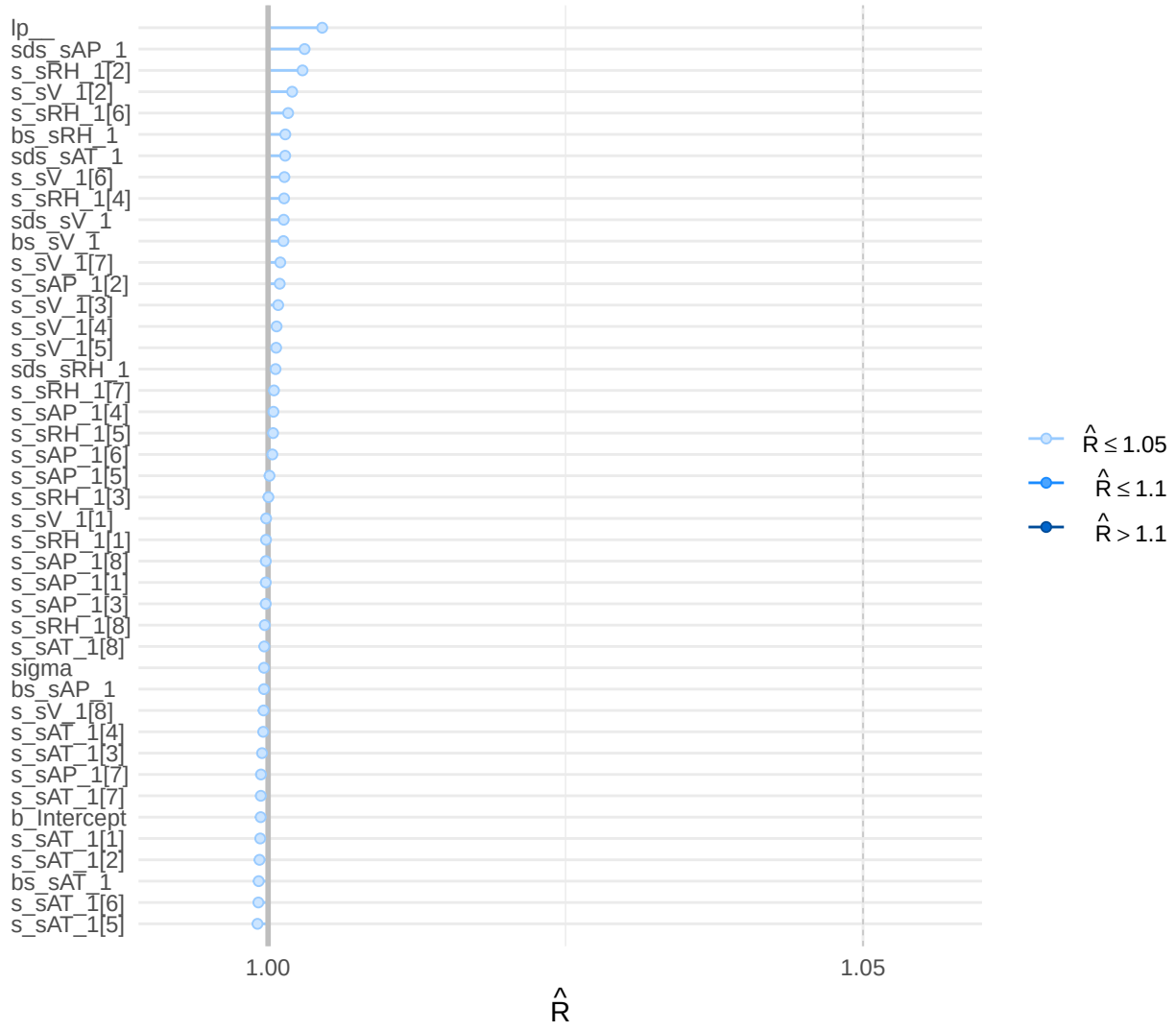
```
print(nonlinear.fit$fit, probs=c(0.025, 0.975))
```

```
## Inference for Stan model: 032c3b7aef4b3341754cd55e3528c216.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##                  mean se_mean    sd      2.5%     97.5% n_eff Rhat
## b_Intercept    454.45    0.00  0.05    454.36    454.54  7690    1
## bs_sAT_1       -63.24    0.19 11.03    -83.87    -40.53  3516    1
## bs_sV_1         -4.68    0.17  9.96    -23.79     15.20  3297    1
## bs_sAP_1       -11.13    0.15  8.04    -26.79      4.56  2700    1
```

```
## bs_sRH_1        -1.22    0.20  8.26    -15.10     16.83  1731    1
## sds_sAT_1       14.36    0.11  4.14      8.38     24.45  1476    1
## sds_sV_1        19.79    0.14  5.21     11.93     32.27  1454    1
## sds_sAP_1        9.04    0.08  3.10      4.77     16.61  1451    1
## sds_sRH_1        6.19    0.08  3.08      1.92     13.51  1461    1
## sigma            4.10    0.00  0.03      4.03      4.17  7532    1
## s_sAT_1[1]      -6.90    0.12  7.94    -22.80      8.33  4455    1
## s_sAT_1[2]      -1.08    0.06  4.38     -9.55      7.49  5095    1
## s_sAT_1[3]     -17.55    0.04  2.55    -22.65    -12.73  3639    1
## s_sAT_1[4]       9.86    0.02  1.68      6.62     13.19  4926    1
## s_sAT_1[5]      20.77    0.05  3.54     13.80     27.93  4173    1
## s_sAT_1[6]     -18.61    0.06  4.03    -26.61    -10.94  5160    1
## s_sAT_1[7]      -0.22    0.09  5.67    -11.42     11.11  4039    1
## s_sAT_1[8]       8.70    0.08  5.94     -3.10     20.29  5680    1
## s_sV_1[1]      -46.25    0.12  7.26    -59.98    -31.93  3508    1
## s_sV_1[2]        7.32    0.07  3.97     -0.25     14.98  3439    1
## s_sV_1[3]       -7.00    0.04  2.43    -11.93     -2.37  3393    1
## s_sV_1[4]       -4.27    0.02  1.42     -6.96     -1.50  4055    1
## s_sV_1[5]      -19.88    0.05  2.80    -25.33    -14.46  3542    1
## s_sV_1[6]      -11.79    0.05  3.20    -18.05     -5.71  4359    1
## s_sV_1[7]      -11.91    0.07  4.35    -20.72     -3.36  4030    1
## s_sV_1[8]       -4.28    0.08  5.45    -14.95      6.38  4588    1
## s_sAP_1[1]      10.15    0.08  5.13      0.94     20.50  3827    1
## s_sAP_1[2]       5.77    0.06  3.75     -1.34     13.46  3561    1
## s_sAP_1[3]       6.12    0.04  2.12      2.01     10.19  2902    1
## s_sAP_1[4]       8.68    0.02  1.19      6.35     11.00  4208    1
## s_sAP_1[5]      -6.06    0.05  2.71    -11.26     -0.81  3584    1
## s_sAP_1[6]       1.37    0.06  3.48     -5.73      8.04  3751    1
## s_sAP_1[7]       7.49    0.08  4.30     -0.48     16.21  3282    1
## s_sAP_1[8]      -9.74    0.10  6.46    -23.33      2.14  3853    1
## s_sRH_1[1]      -4.68    0.11  5.90    -18.56      4.44  2680    1
## s_sRH_1[2]       9.72    0.09  3.78      3.26     17.88  1927    1
## s_sRH_1[3]       0.71    0.03  1.55     -2.27      3.96  3414    1
## s_sRH_1[4]      -2.11    0.03  1.39     -4.88      0.52  2817    1
## s_sRH_1[5]      -1.27    0.05  2.61     -6.52      3.54  3230    1
## s_sRH_1[6]       3.52    0.07  3.73     -2.84     11.75  2900    1
## s_sRH_1[7]      -4.29    0.09  4.60    -14.25      3.41  2876    1
## s_sRH_1[8]       3.07    0.06  4.51     -6.04     12.36  6082    1
## lp__        -21933.80    0.22  6.29 -21946.65 -21922.57   857    1
##
## Samples were drawn using NUTS(diag_e) at Wed Dec 02 10:22:25 2020.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

Note that the printout above reveals that the names of the parameters maybe a little bit different from the Stan code shown above. This may be because of some internal workings of `brms`.

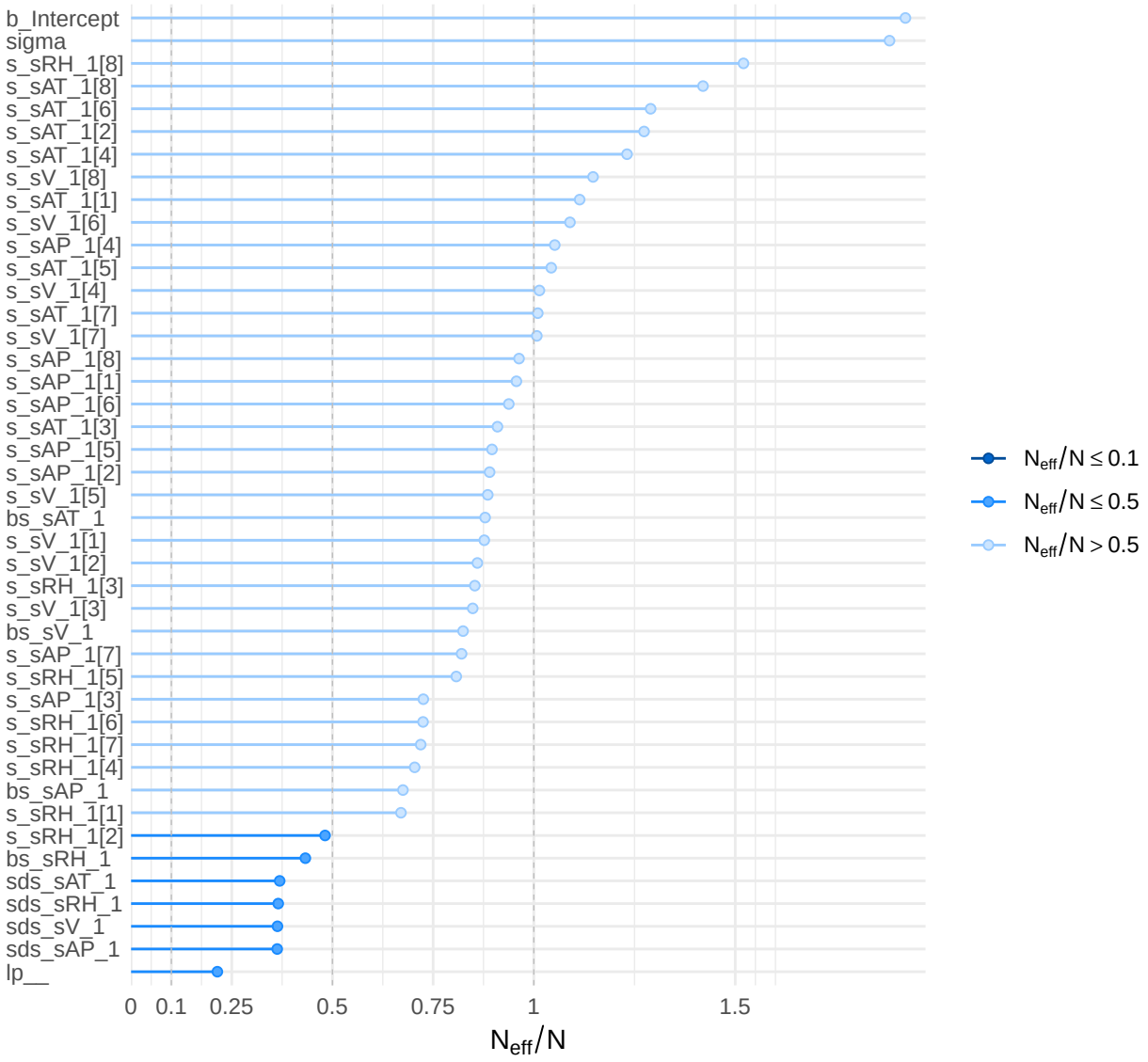
Next, we check $\hat{R}$. $\hat{R}$ is a potential scale reduction factor. $\hat{R}$ measures the ratio of the mean within the variance of the chains to the variance of the pooled draws across chains. These are equal if all chains are at equilibrium, and $\hat{R} \sim 1.00$. Empirically, if $\hat{R}$ is larger than 1.05, it usually means that the sequences have yet to converge, and then we should run more iterations to reach convergence. For the GAM, by looking at the printout above, it is clear that the chains have mixed well and reached convergence since the $\hat{R}$ values are all ≈ 1.00. We could also visualize the $\hat{R}$ values.



As shown in the plot, all $\hat{R}$ values are approximately 1.00. Note that some values are smaller than 1 because of underflows in numerical computation. The GAM has converged well. We could also check convergence graphically with traceplots like above; however, this is not really necessary here as there are a lot of Stan parameters and we could always monitor convergence with $\hat{R}$ instead.

Next, we check the effective sample size n_{eff}. In Stan context, the drawn samples are usually autocorrelated within each chain; n_{eff} measures the number of explanatory samples with the same estimation quality as the total sample size. The higher the n_{eff}'s are, the better the samples are. By looking at the summary above, we see that the n_{eff} values seem high enough. We could also inspect the ratio n_{eff}/N, where N is the total sample size:

```
mcmc_neff(neff_ratio(nonlinear.fit))+yaxis_text(hjust=0)#replace neff_ratio by rhat gets plot above
```



One heuristic is that ratios smaller than 0.1 indicate that something went wrong and we should check the model again. In the plot, we see that the n_{eff}'s for the standard deviations of the spline coefficients `sds_s...` are not as good compared to other n_{eff}'s, but they are still good enough.

Now, we examine HMC-specific convergence diagnostics. We first look for divergence transitions, which mean that the HMC algorithm has problems sufficiently exploring the neighborhoods of the posterior distribution around those transitions.

```
rstan::check_divergences(nonlinear.fit$fit)
```

```
## 0 of 4000 iterations ended with a divergence.
```

We did not encounter any divergent transitions, which means that the HMC algorithm worked well. Finally, we check the tree depth. Stan sets an upper limit on the depth of the trees that the NUTS algorithm, a type of HMC used by Stan, evaluates in each iteration. An iteration that saturated the maximum tree depth means that the algorithm stopped early to avoid long running times (Stan Development Team, 2020).

```
rstan::check_treedepth(nonlinear.fit$fit)
```

```
## 3 of 4000 iterations saturated the maximum tree depth of 10 (0.075%).
```

```
## Try increasing 'max_treedepth' to avoid saturation.
```

We got very few iterations that saturated the maximum tree depth, which can be considered as good enough. Below, we also provide a more general summary of the fit.

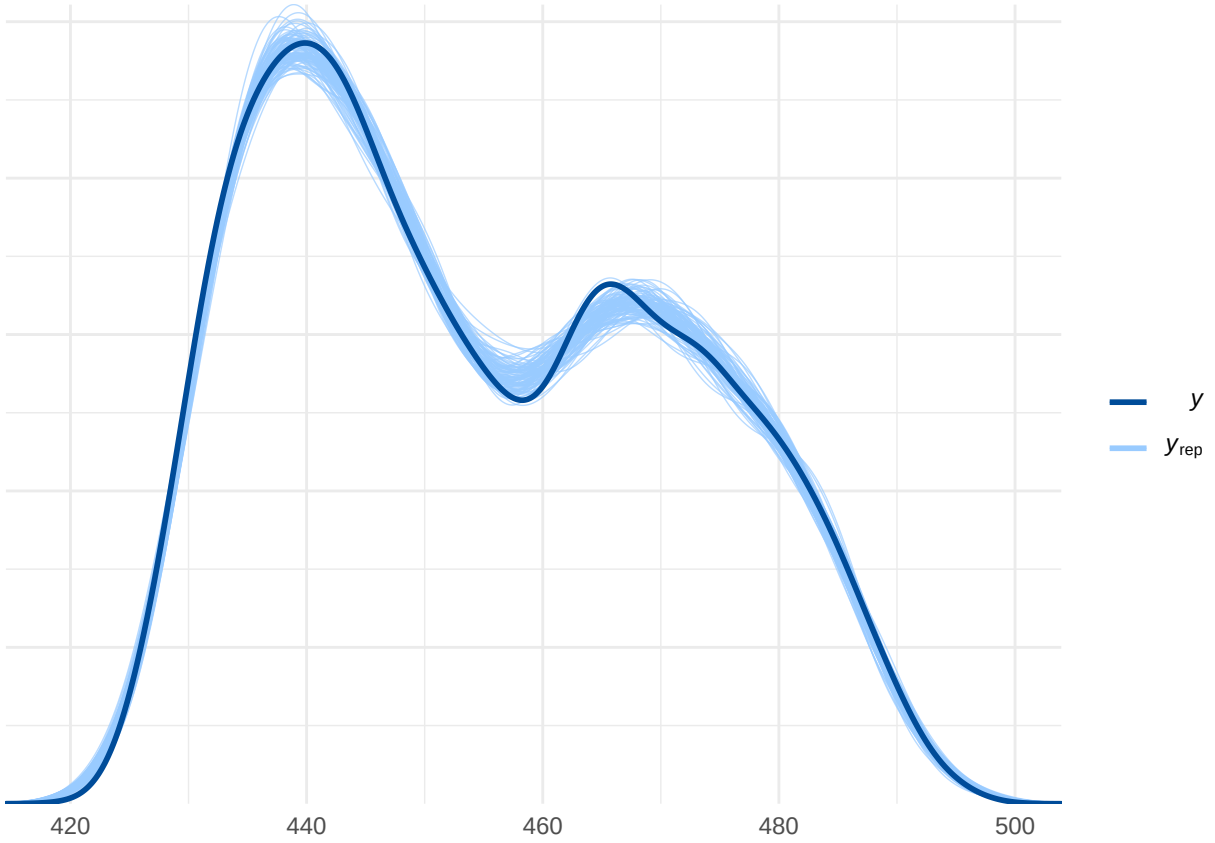
```
summary(nonlinear.fit, priors=T)
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: PE ~ s(AT) + s(V) + s(AP) + s(RH)
##    Data: train (Number of observations: 7724)
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 4000
##
## Priors:
## b ~ normal(0, 100)
## Intercept ~ normal(460, 75)
## sds ~ exponential(0.1)
## sigma ~ exponential(0.05)
##
## Smooth Terms:
##            Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sds(sAT_1)    14.36      4.14     8.38    24.45 1.00     1402     2044
## sds(sV_1)     19.79      5.21    11.93    32.27 1.00     1380     2046
## sds(sAP_1)     9.04      3.10     4.77    16.61 1.00     1386     2259
## sds(sRH_1)     6.19      3.08     1.92    13.51 1.00     1366     2126
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept   454.45      0.05   454.36   454.54 1.00     7461     2437
## sAT_1       -63.24     11.03   -83.87   -40.53 1.00     3462     2775
## sV_1         -4.68      9.96   -23.79    15.20 1.00     3319     2351
## sAP_1       -11.13      8.04   -26.79     4.56 1.00     2715     2393
## sRH_1        -1.22      8.26   -15.10    16.83 1.00     1682     2249
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     4.10      0.03     4.03     4.17 1.00     7378     2712
##
## Samples were drawn using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

4.5 Posterior predictive checks

We conduct graphical posterior predictive checks to check the goodness of fit of the GAM.

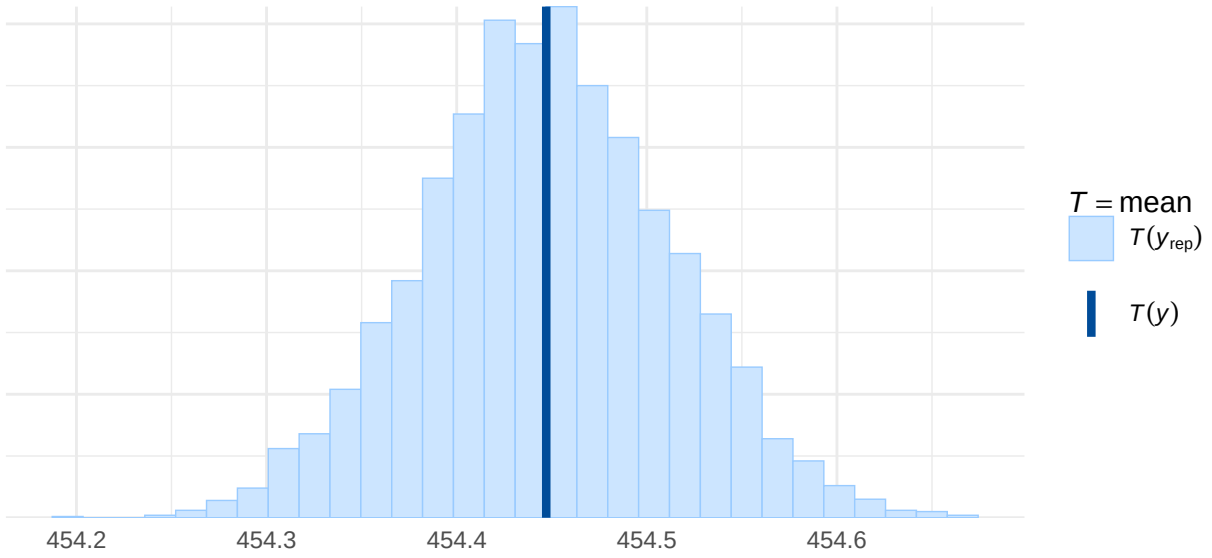
```
pp_check(nonlinear.fit, type="dens_overlay", nsamples=100)
```



From the plot above, it is evident that the GAM can simulate data that looks very similar to the observed PE values. Upon closer inspection, we see that the simulated curves did not quite capture the shape of the observed curve around the peak on the right. This is understandable because the observed curve has a complicated shape around that region; the fit of the GAM can be considered satisfactory concerning this plot.

One posterior test statistic of interest is the mean.

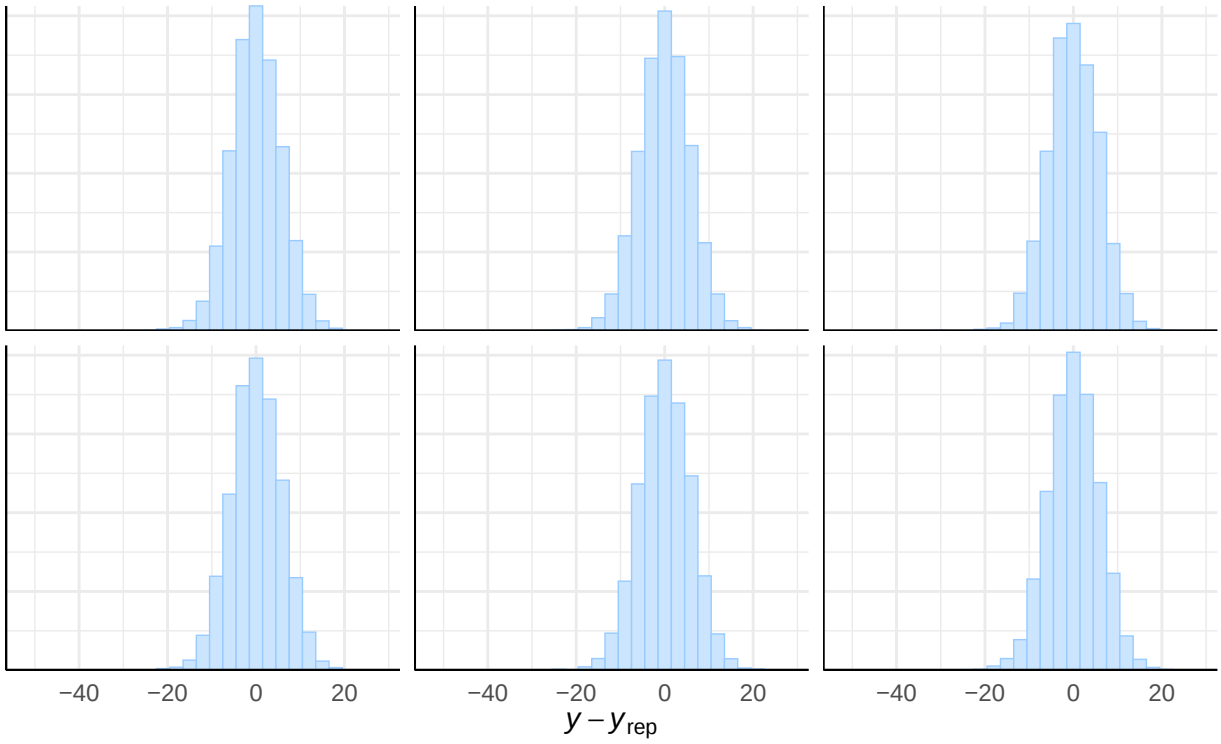
```
pp_check(nonlinear.fit, type='stat', stat='mean', nsamples=4000)
```



The distribution of the replicated means looks like a normal distribution and the range is very narrow. This could indicate a good model fit.

Next, we look at the distributions of the predictive errors (`y-yrep`). We use six `yrep` samples.

```
pp_check(nonlinear.fit, type="error_hist", nsamples=6, binwidth=3)
```

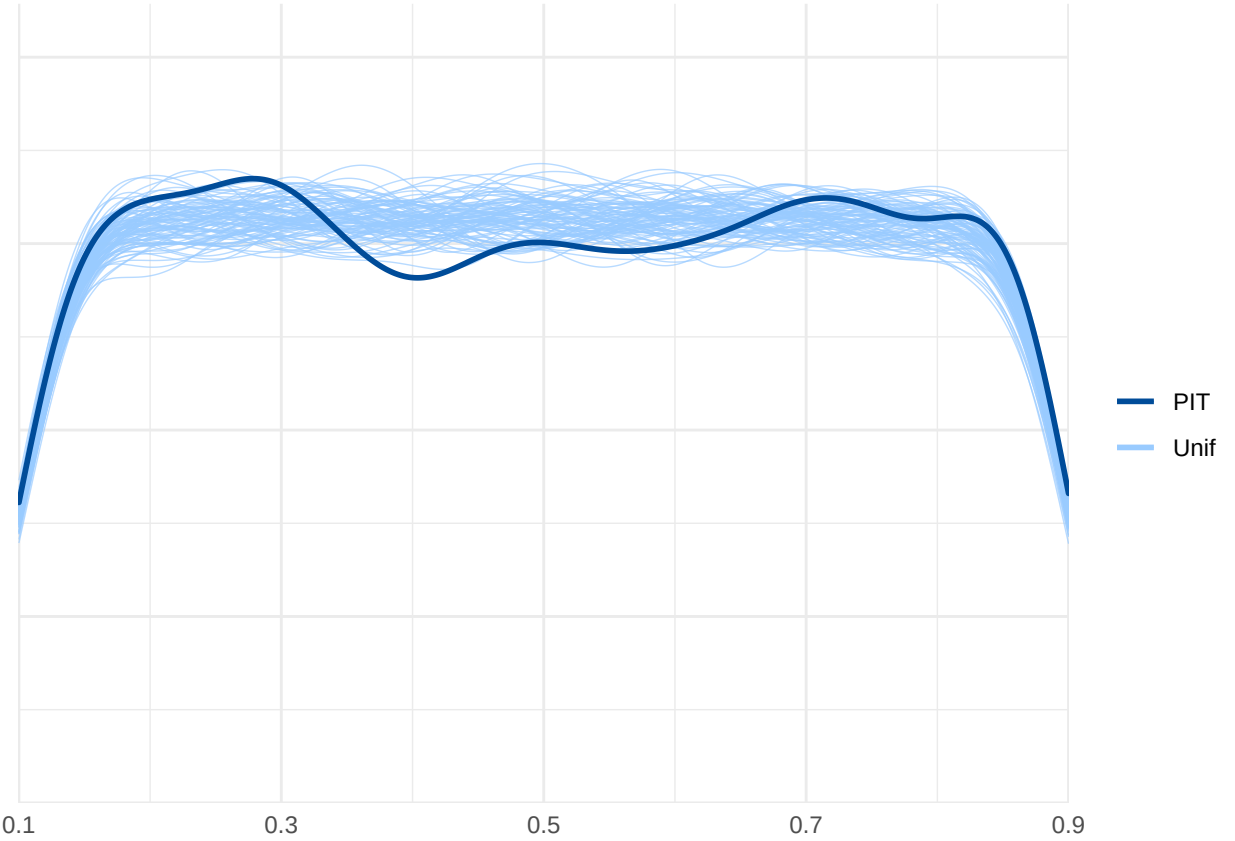


These distributions look approximately normal even though there are very small proportions of errors less than -25, indicating that the GAM occasionally predicts larger `PE`s than were observed. Despite that, most of the errors are distributed around zero. We could also look at the scatterplot of the average predictive errors.

```
pp_check(nonlinear.fit, type='error_scatter_avg', nsamples=4000)
```

Finally, we do LOO-PIT check.

```
pp_check(nonlinear.fit, type="loo_pit_overlay", nsamples = 4000)
```



Since the thick curve has a similar shape to the thin curves, the GAM is calibrated.

Compared to the posterior predictive plots shown in the linear model section, it is evident that the GAM

fits the data better than the linear model.

4.6 Leave-one-out cross-validation

See the linear model section for an interpretation of PSIS-LOO cross-validation.

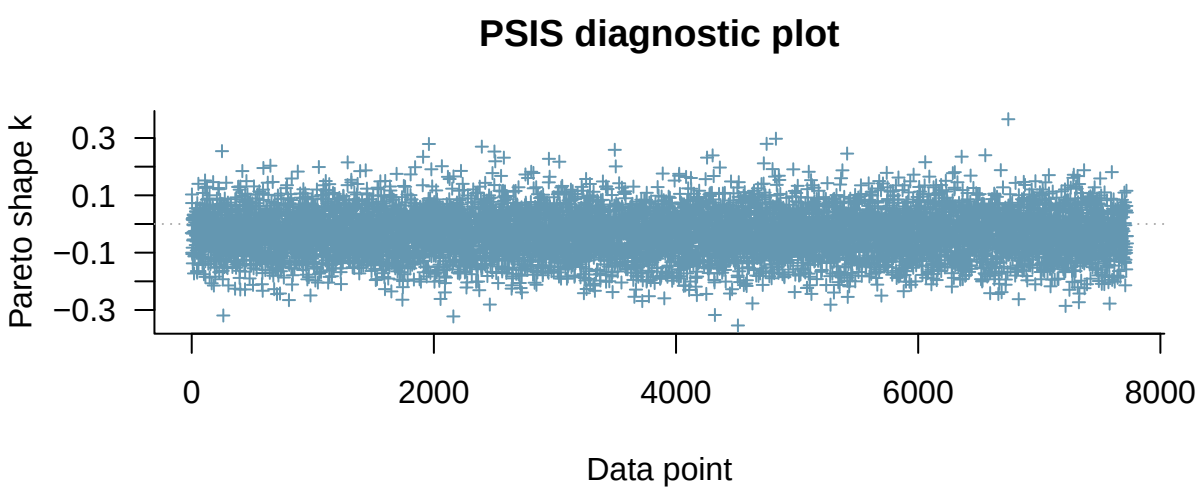
```
nonlinear.loo <- loo(nonlinear.fit)
knitr::kable(nonlinear.loo$estimates, digits=1, caption="LOO estimates of GAM")
```

Table 10: LOO estimates of GAM

	Estimate	SE
elpd_loo	-21873.9	105.0
p_loo	34.4	2.1
looic	43747.8	210.1

`p_loo` is quite large compared to the linear model because we can see that the GAM has a lot more parameters than the linear model. Next, we check the Pareto `k` estimates.

```
plot(nonlinear.loo)
```



Since all `k` estimates are below 0.5, the `elpd_loo` estimate is accurate and reliable.

4.7 Predictive performance

```
pp_rmse(nonlinear.fit, test)
```

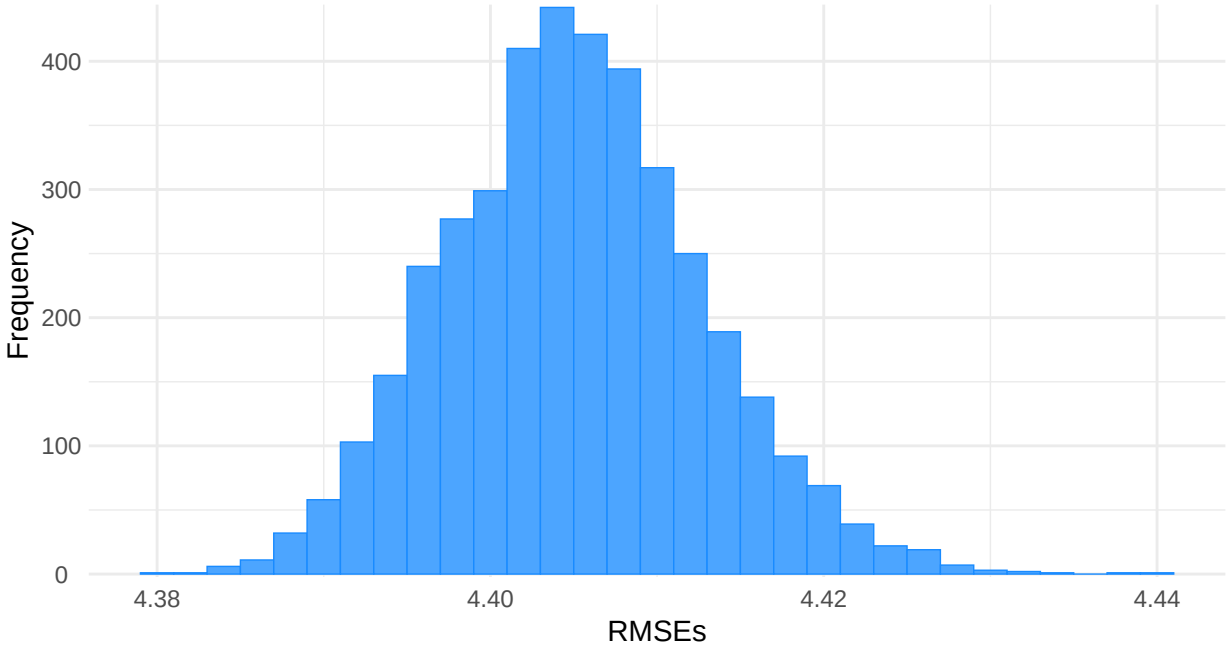


Table 11: Estimates for RMSE

	Estimate	Est.Error	Q2.5	Q97.5
RMSEs	4.405	0.008	4.391	4.421

```
bayes_r2(nonlinear.fit, test)
```

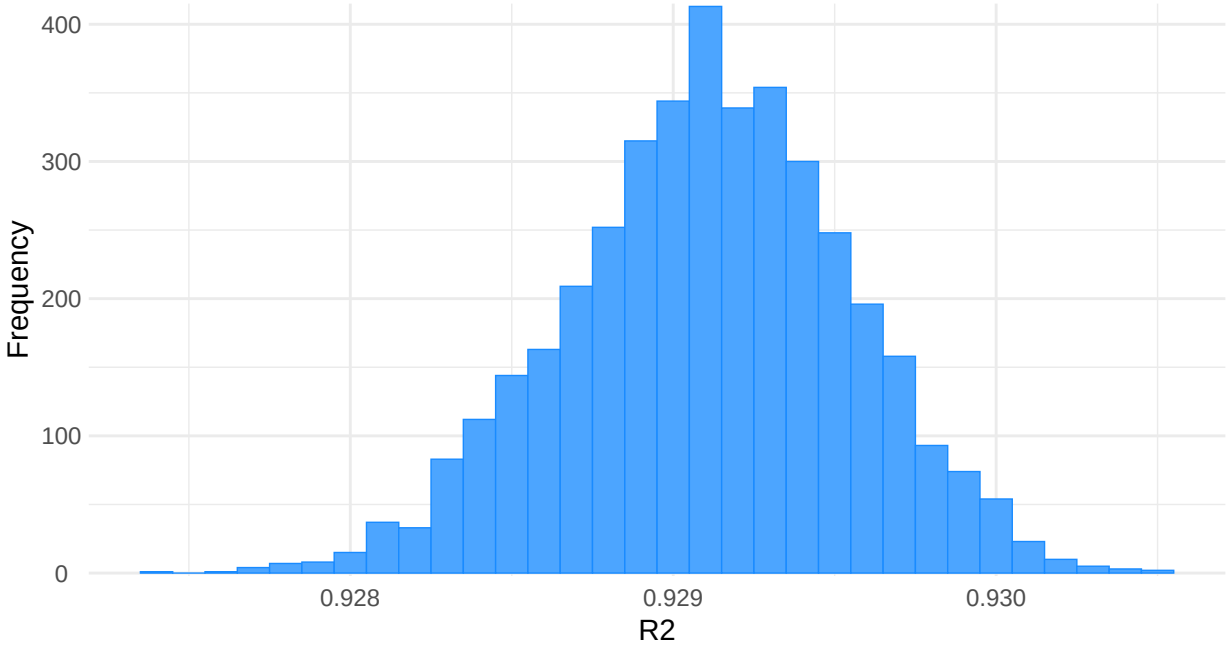


Table 12: Estimates for R2

	Estimate	Est.Error	Q2.5	Q97.5
R2	0.92911	0.00044	0.92823	0.92995

The practical usefulness of these statistics is described in the linear model section.

4.8 Sensitivity analysis

To perform sensitivity analysis on the GAM, we test the following prior:

```
nonlinear.prior2 <-
  prior(normal(0, 100), class=b) +
  prior(exponential(0.001), class=sds) +
  prior(exponential(0.05), class=sigma) +
  prior(normal(460, 75), class=Intercept)
```

That is, we only change the prior for the standard deviations of the spline coefficients (σ_i). We let the prior become wider to allow for more "wiggly" splines. The mean of the exponential distribution with rate parameter 0.001 is $1/0.001 = 1000$. We consider the GAM fitted above `nonlinear.fit` to be the base model.

```
nonlinear.fit2 <- nonlinear.formula %>%
  brm(data=train, prior=nonlinear.prior2, file="models/nonlinear2",
      seed=123, refresh=0, control=list(adapt_delta=0.95)) %>%
  add_criterion("loo", file="models/nonlinear2")
rstan::check_hmc_diagnostics(nonlinear.fit2$fit)
```

```
##
## Divergences:
```

```
## 0 of 4000 iterations ended with a divergence.
```

```
##
## Tree depth:
```

```
## 148 of 4000 iterations saturated the maximum tree depth of 10 (3.7%).
## Try increasing 'max_treedepth' to avoid saturation.
```

```
##
## Energy:
```

```
## E-BFMI indicated no pathological behavior.
```

We got a considerable number of transitions that saturated the maximum treedepth compared to the base GAM `nonlinear.fit`.

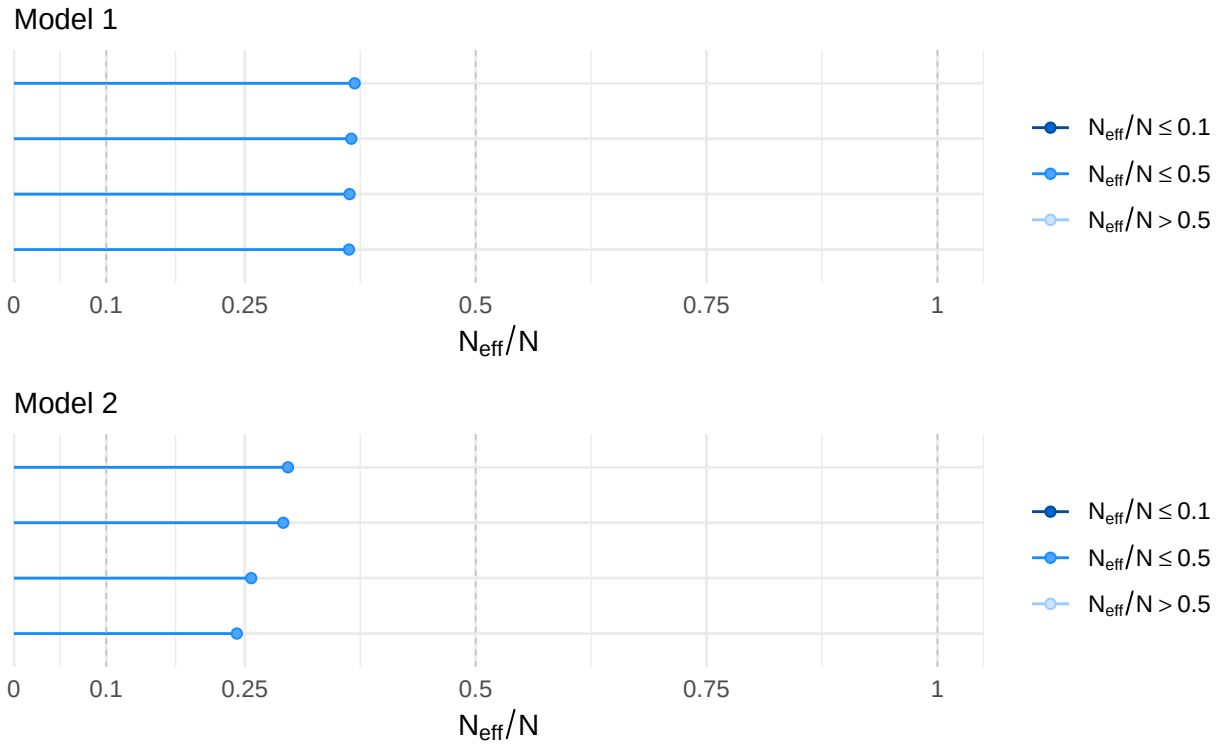
```
summary(nonlinear.fit2)
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: PE ~ s(AT) + s(V) + s(AP) + s(RH)
##    Data: train (Number of observations: 7724)
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 4000
##
## Smooth Terms:
##            Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sds(sAT_1)    16.36      5.38     9.13    29.89 1.00      975     1931
```

```
## sds(sV_1)     23.38      7.53    13.19    41.27 1.00     1054     1687
## sds(sAP_1)    10.14      4.12     5.08    20.36 1.00     1050     2007
## sds(sRH_1)     7.13      3.67     2.15    16.37 1.01      889     1939
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept   454.45      0.05   454.36   454.54 1.00     6011     3112
## sAT_1       -61.90     11.34   -82.92   -39.70 1.00     2211     2722
## sV_1         -5.90     10.03   -25.91    13.35 1.00     2868     2299
## sAP_1       -11.07      8.19   -26.77     5.39 1.00     2166     2644
## sRH_1         0.30      9.00   -15.02    19.77 1.00     1109     2152
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     4.10      0.03     4.04     4.16 1.00     7688     2381
##
## Samples were drawn using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

We see that the estimates for the standard deviations of the spline coefficients σ_i (`sds`) have increased, indicating that our wide prior was reflected in the posterior. However, the ESS values seem lower than those of the base model:

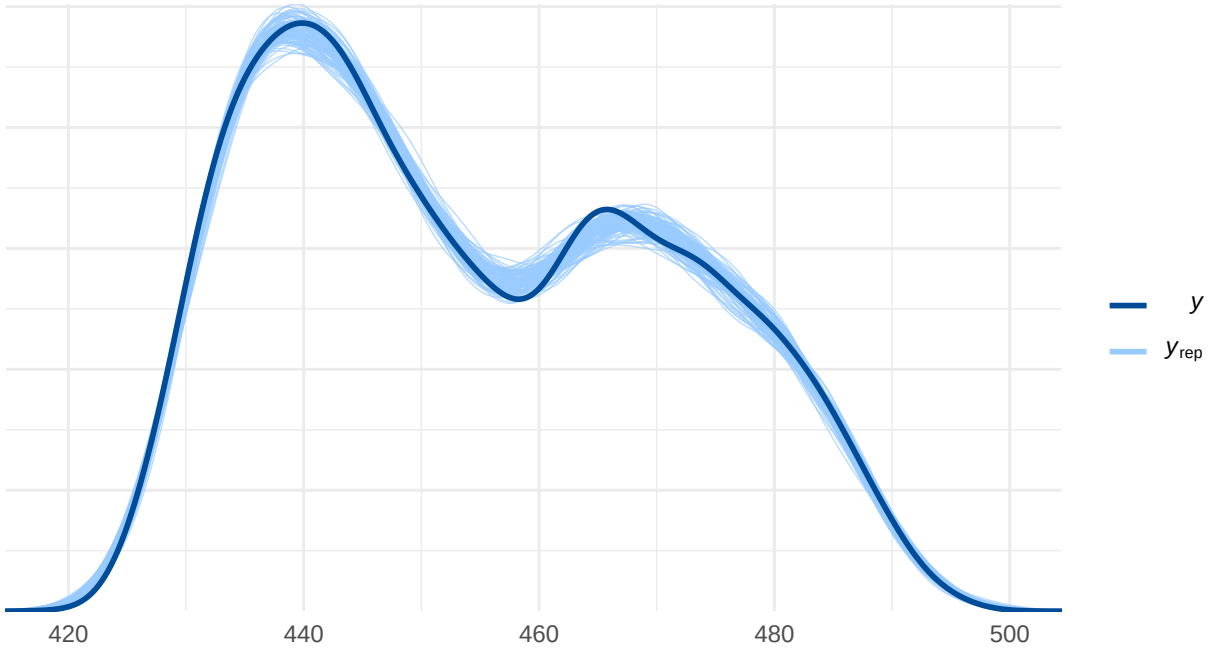
```
pars = c("sds_sAT_1", "sds_sAP_1", "sds_sV_1", "sds_sRH_1")
neff_fit <- neff_ratio(nonlinear.fit, pars = pars)
neff_fit2 <- neff_ratio(nonlinear.fit2, pars = pars)
compare_neff(mcmc_neff(neff_fit), mcmc_neff(neff_fit2), ncol = 1)
```



Therefore, the draws for the `sds` values of `nonlinear.fit2` are a bit worse than those for the `sds` values of

`nonlinear.fit`. Next, we look at the kernel density estimates for the replicated datasets of `nonlinear.fit2`.

```
pp_check(nonlinear.fit2, nsamples=100)
```



This plot looks essentially the same as the plot for the base model above. We conclude that allowing for more freedom for the σ_i's will result in a slightly worse model because of the lower ESS for `sds` terms.

One other alternative prior that we will test is:

```
nonlinear.prior3 <-
  prior(normal(0, 1), class=b) +
  prior(exponential(1), class=sds) +
  prior(exponential(0.05), class=sigma) +
  prior(normal(460, 75), class=Intercept)
```

The idea here is that we would like to restrict the wiggliness of the splines. The "linear effects" `b` have very narrow priors, and so are the σ_i.

```
nonlinear.fit3 <- nonlinear.formula %>%
  brm(data=train, prior=nonlinear.prior3, file="models/nonlinear3",
      seed=123, refresh=0, control=list(adapt_delta=0.95)) %>%
  add_criterion("loo", file="models/nonlinear3")
rstan::check_hmc_diagnostics(nonlinear.fit3$fit)
```

```
##
## Divergences:

## 0 of 4000 iterations ended with a divergence.

##
## Tree depth:

## 0 of 4000 iterations saturated the maximum tree depth of 10.

##
## Energy:

## E-BFMI indicated no pathological behavior.
```

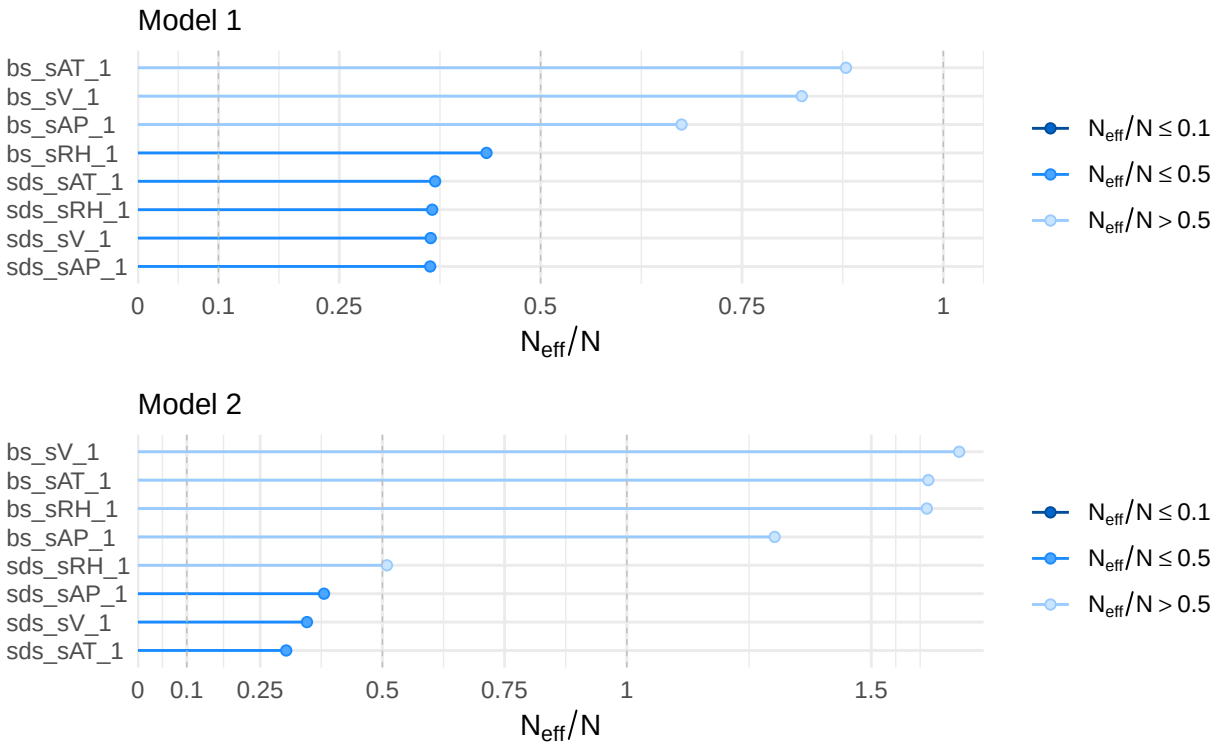
There were no warnings this time, which is a bit surprising.

```
summary(nonlinear.fit3)
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: PE ~ s(AT) + s(V) + s(AP) + s(RH)
##    Data: train (Number of observations: 7724)
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 4000
##
## Smooth Terms:
##            Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sds(sAT_1)    15.20      2.10    11.51    19.60 1.00     1185     1614
## sds(sV_1)     11.47      1.94     8.22    15.81 1.00     1371     1939
## sds(sAP_1)     5.82      1.47     3.50     9.20 1.00     1433     1910
## sds(sRH_1)     4.04      1.10     2.37     6.62 1.00     2006     2231
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept   454.45      0.05   454.36   454.53 1.00     6258     2438
## sAT_1        -0.55      1.00    -2.50     1.34 1.00     6574     3060
## sV_1         -0.00      0.97    -1.91     1.85 1.00     6864     3137
## sAP_1        -0.32      0.98    -2.21     1.60 1.00     5265     2989
## sRH_1        -0.11      0.98    -2.02     1.81 1.00     6667     3244
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     4.11      0.03     4.04     4.17 1.00     7308     3069
##
## Samples were drawn using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

Compared to `nonlinear.fit2` and `nonlinear.fit1`, the estimates for the parameters shown in the printout above have decreased, especially the "population-level effects" coefficients since we enforced a narrow prior on them. The ESS estimates for the "smooth terms" have increased (compared to `nonlinear.fit2`), indicating that less wiggly splines seem to be better.

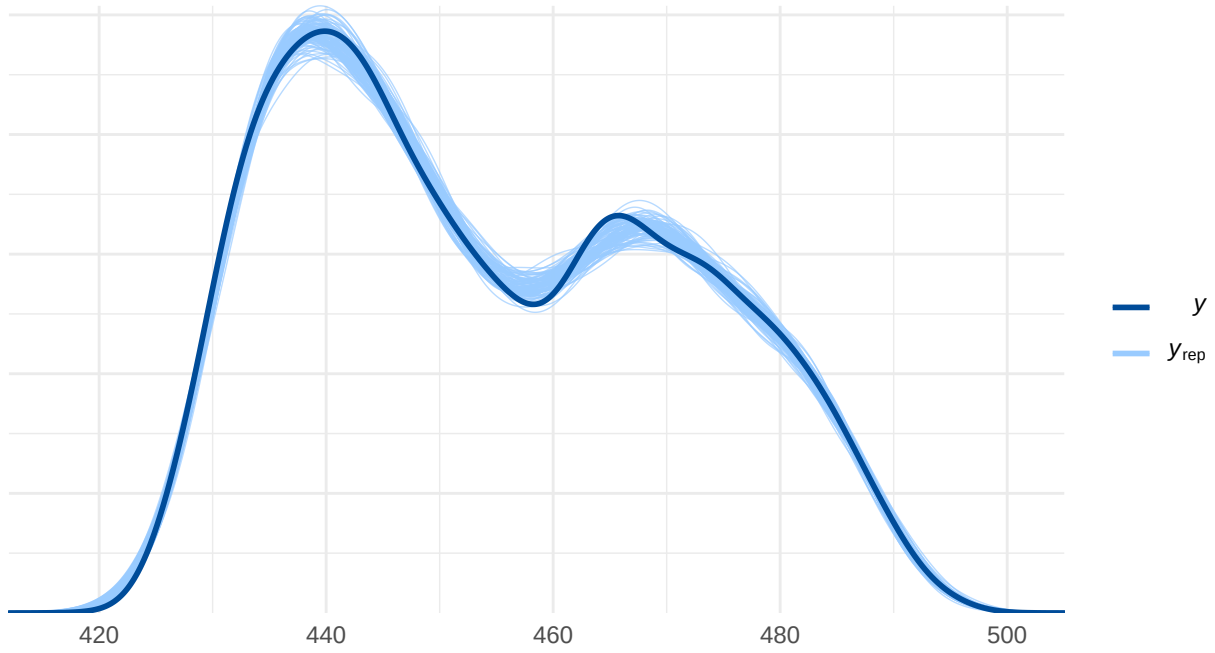
```
pars <- append(pars, c("bs_sAT_1", "bs_sV_1", "bs_sAP_1", "bs_sRH_1"))
neff_fit <- neff_ratio(nonlinear.fit, pars = pars)
neff_fit3 <- neff_ratio(nonlinear.fit3, pars = pars)
compare_neff(mcmc_neff(neff_fit)+ yaxis_text(hjust = 0),
             mcmc_neff(neff_fit3) + yaxis_text(hjust = 0), ncol = 1)
```



Compared to `nonlinear.fit`, `nonlinear.fit3` has approximately the same ratio `N_eff/N` for the σ_i. However, `N_eff/N` ratios for the linear effects have increased, indicating that a narrower prior on linear effects seems to be better.

Next, we look at the kernel density estimates for the replicated datasets of `nonlinear.fit3`.

```
pp_check(nonlinear.fit3, nsamples=100)
```



Again, there is no noticeable visual change in the kernel density estimates. It seems that changing priors for

the spline coefficients and standard deviations have little effect on the posterior predictive distribution.

Finally, we compare all three GAMs with PSIS-LOO CV:

```
knitr::kable(loo_compare(nonlinear.fit, nonlinear.fit2, nonlinear.fit3),
             digits=1, caption="PSIS-LOO comparison")
```

Table 13: PSIS-LOO comparison

	elpd_diff	se_diff	elpd_loo	se_elpd_loo	p_loo	se_p_loo	looic	se_looic
nonlinear.fit2	0.0	0.0	-21873.6	105.1	35.0	2.1	43747.1	210.1
nonlinear.fit	-0.3	0.4	-21873.9	105.0	34.4	2.1	43747.8	210.1
nonlinear.fit3	-10.1	6.3	-21883.7	104.3	29.3	1.7	43767.3	208.6

The higher the `elpd_loo` estimate for a model, the more accurate in prediction it is. Although there is barely any difference between `nonlinear.fit` and `nonlinear.fit3`, `nonlinear.fit3` seems to be the worst model. This is probably because we set very narrow priors on the linear effects of `nonlinear.fit3`. We also believe that the prior on the `sds` parameters for `nonlinear.fit3` is also a little bit narrow.

Having conducted sensitivity analysis with regard to priors on parameters for the splines $s(x_i)$, we conclude that the σ_i should have a prior that is not too narrow or too wide (Exponential(1) was a bit too narrow), and so should the β_i parameters ($N(0, 100)$ was a bit too wide). Also, changing spline parameters does not seem to have a strong effect on the posterior predictive distribution.

5 Model comparison

Having evaluated PSIS-LOO cross-validation on each model separately, we now compare the GAM with the better linear model `fit2`. We compare models based on the `elpd_loo` estimate computed in previous sections.

```
cmp <- loo_compare(loo(nonlinear.fit), loo2)
knitr::kable(cmp, digits=1, caption="Comparison between GAM and linear model")
```

Table 14: Comparison between GAM and linear model

	elpd_diff	se_diff	elpd_loo	se_elpd_loo	p_loo	se_p_loo	looic	se_looic
nonlinear.fit	0.0	0.0	-21873.9	105.0	34.4	2.1	43747.8	210.1
fit2	-733.9	42.2	-22607.7	88.5	7.1	0.7	45215.5	176.9

It seems that there is a significant difference between the two models with regard to their PSIS-LOO values. The model with the highest PSIS-LOO estimate is the GAM (the first row of the model). Thus, we conclude that GAM is the best model. Note that it can also be seen from the posterior predictive checks above that the GAM fits the data better than the linear model.

6 Conclusion

6.1 Conclusion of the data analysis

The data analysis provided an alternative method to predict the full-load electrical power output of the combined cycle power plant (CCPP) with the ambient conditions in the plant. Instead of modeling the CCPP as a thermodynamic system and solving systems of equations, and instead of frequentist methods such as machine learning (Tüfekci, 2014), we applied Bayesian modeling and fitted two Bayesian models that

incorporated our prior belief. The results obtained from these Bayesian models reflected our uncertainty of the posterior distribution and the estimates of parameters and quantities of interest.

We also aimed to determine which of the two models, the linear model, or the generalized additive model (GAM) was the better one. The result was that the GAM, dubbed "nonlinear model" in our code, was the better model because it captured the seemingly nonlinear relationship between the response variable and the explanatory variables better than did the linear model. We also compared the two models with regard to posterior predictive checking and PSIS-LOO cross-validation and found that the GAM performed better at posterior predictive checking, indicated by the satisfactory fit of the kernel density estimates replicated data against the observed density of PE; in addition, the PSIS-LOO estimate for the GAM was higher than that for the linear model, meaning that the GAM could be more accurate in predictive power than the liner model.

We conclude that in a Bayesian regression context, simple linear models such as the one we fitted are rarely useful unless one knows that the true relationship between the explanatory variables and the response variable is a linear one. In addition, a GAM could be a decent solution to common regression tasks since it can capture common nonlinear relationships pretty well, especially when one does not know the true relationship.

6.2 Discussion

Despite both of us having a data science background, the application of Bayesian workflow to a data analysis project has been an interesting experience. Instead of just creating some neural networks that can magically resolve the larger part of the data preparation and feature engineering processes, we have to delve deeper into the data. It is to find suitable priors based on the nature of some variables and to discover a function that can reasonably represent the nature of the interactions between the response and explanatory variables. This took a large portion of our time working on this project. Thus the Bayesian workflow requires more domain knowledge of the subject which is somewhat akin to a traditional machine learning workflow.

Among the issues that we have encountered is the choice of units of variables. When we first looked at the data, a few choices regarding data scaling were presented. We could have either scaled or normalized the data since that would have a positive impact on our models' performance. We instead went with changing the units of the variable themselves to ensure that all variables have a similar range and the results have the same interpretability as if they were not changed. This was, in effect, a way to scale the data. Another issue was that because we did not deeply understand splines and generalized linear models (since we had little time to study the subject), the explanation for the GAM model above was quite vague and unclear. Given more time, we could study the subject more thoroughly and come up with models that have great priors and are good fits to the data.

We experimented with two ideas that were not presented in this report, variable selection and building a hierarchical model with groups determined by an external clustering method. we tried running variable selection with our linear models. However, due to hours of running time and inconclusive results, we decided against including variable selection in the report. The result was inconclusive since there are only 4 explanatory variables and removing any of them would have a large impact on the root mean square error. We also tried building a hierarchical model with groups identified by a clustering algorithm (kmeans, hierarchical clustering); however, we faced several issues. Our attempts to create a working hierarchical model ended up with large `Rhat` warnings, `maximum_treedepth` saturation warnings, and divergent transitions. The attempted model did not converge. A possible explanation was that we specified wrong priors. This is a possible area of improvement. Given more time, we could experiment more with hierarchical modeling and create a working model with a hierarchical structure.

7 References

Gabry, J., Simpson, D., Vehtari, A., Betancourt, M., & Gelman, A. (2019). Visualization in Bayesian workflow. Journal of the Royal Statistical Society: Series A (Statistics in Society), 182(2), 389-402.

Gelman, A., Goodrich, B., Gabry, J., & Vehtari, A. (2019). R-squared for Bayesian regression models. The American Statistician, 73(3), 307-309.

Kaya, H., Tüfekci, P., & Gürgen, F. S. (2012, March). Local and global learning methods for predicting power of a combined gas & steam turbine. In Proceedings of the International Conference on Emerging Trends in Computer and Electronics Engineering ICETCEE (pp. 13-18).

Liu, C., Wang, H., Ding, J., & Zhen, C. (2012). An overview of modelling and simulation of thermal power plant. International Journal of Advanced Mechatronic Systems, 4(2), 76-85.

Smrekar, J., Pandit, D., Fast, M., Assadi, M., & De, S. (2010). Prediction of power output of a coal-fired power plant by artificial neural network. Neural Computing and Applications, 19(5), 725-740.

Tüfekci, P. (2014). Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods. International Journal of Electrical Power & Energy Systems, 60, 126-140.

Vehtari, A., Gelman, A., & Gabry, J. (2017). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. Statistics and computing, 27(5), 1413-1432.

Stan Development Team (2020). Prior Choice Recommendations. https://github.com/stan-dev/stan/wiki/Prior-Choice-Recommendations

Stan Development Team (2020). Brief Guide to Stan's Warnings. https://mc-stan.org/misc/warnings.html