Contents


```
FALSE Loading required package: StanHeaders

FALSE Loading required package: ggplot2

FALSE rstan (Version 2.19.2, GitRev: 2e1f913d3ca3)

FALSE For execution on a local, multicore CPU with excess RAM we recommend calling
FALSE options(mc.cores = parallel::detectCores()).
FALSE To avoid recompilation of unchanged Stan programs, we recommend calling
FALSE rstan_options(auto_write = TRUE)

FALSE For improved execution time, we recommend calling
FALSE Sys.setenv(LOCAL_CPPFLAGS = '-march=native')
FALSE although this causes Stan to throw an error on a few processors.

FALSE This is loo version 2.1.0.
FALSE **NOTE: As of version 2.0.0 loo defaults to 1 core but we recommend using as many as possible. Us

FALSE **NOTE for Windows 10 users: loo may be very slow if 'mc.cores' is set in your .Rprofile file (se

FALSE
FALSE Attaching package: 'loo'

FALSE The following object is masked from 'package:rstan':
FALSE
FALSE     loo

FALSE Loading required package: Rcpp

FALSE rstanarm (Version 2.19.2, packaged: 2019-10-01 20:20:33 UTC)

FALSE - Do not expect the default priors to remain the same in future rstanarm versions.

FALSE Thus, R scripts should specify priors explicitly, even if they are just the defaults.

FALSE - For execution on a local, multicore CPU with excess RAM we recommend calling
```

```
FALSE options(mc.cores = parallel::detectCores())

FALSE - bayesplot theme set to bayesplot::theme_default()

FALSE    * Does _not_ affect other ggplot2 plots

FALSE    * See ?bayesplot_theme_set for details on theme setting

FALSE
FALSE Attaching package: 'rstanarm'

FALSE The following object is masked from 'package:rstan':
FALSE
FALSE     loo

FALSE This is bayesplot version 1.7.0

FALSE - Online documentation and vignettes at mc-stan.org/bayesplot

FALSE - bayesplot theme set to bayesplot::theme_default()

FALSE    * Does _not_ affect other ggplot2 plots

FALSE    * See ?bayesplot_theme_set for details on theme setting

FALSE Parsed with column specification:
FALSE cols(
FALSE   year = col_double(),
FALSE   state = col_character(),
FALSE   month = col_character(),
FALSE   number = col_double(),
FALSE   date = col_date(format = "")
FALSE )
```

1 Abstract

The aim of this project is to analyze the number of fires in the states of Brazil and build a Bayesian model in order to make predictions about the frequency of forest fires in a time series can help to take action to prevent them.

2 Introduction

The dataset on which the analysis are made is taken from "Kaggle", an online community of data scientists and machine learners where many data sets are available for the users. The "amazon" dataset reports the number of forest fires in Brazil divided by states. The series comprises the period of approximately 10 years (1998 to 2017). The data were obtained from the official website of the Brazilian government. Brazil has the largest rainforest on the planet that is the Amazon rainforest. Forest fires are a serious problem for the preservation of the Tropical Forests. Understanding the frequency of forest fires in a time series can help to take action to prevent them.

3 Explorative Analysis

In the original dataset "Amazon", the following features are present:

- year (1998 - 2017);
- state (23);
- month;
- number;

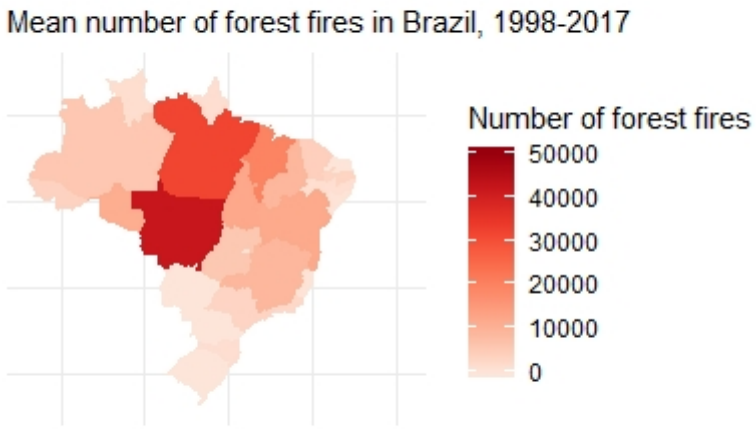


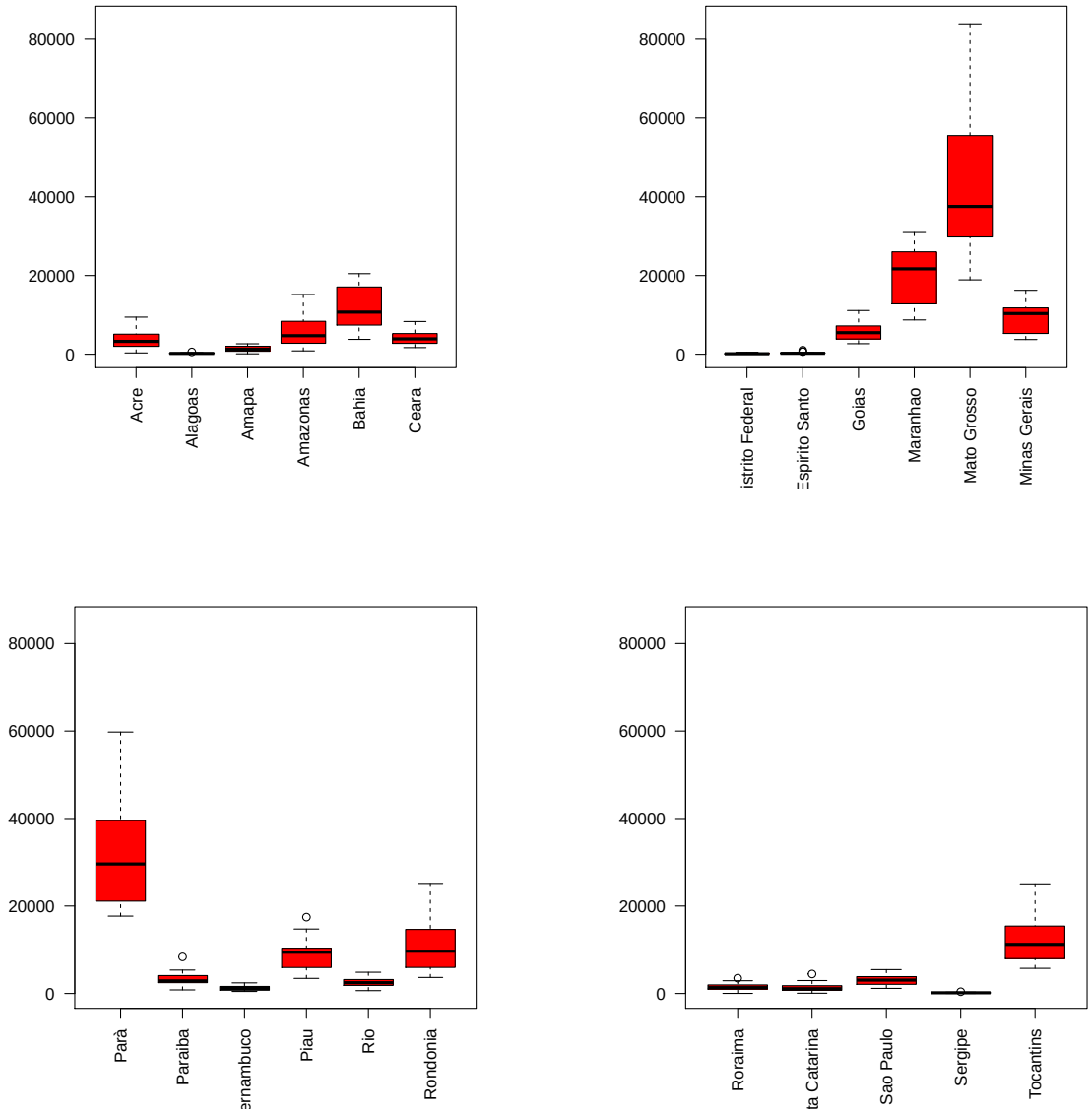
Figure 1: Number of fires - Brazil

- date;

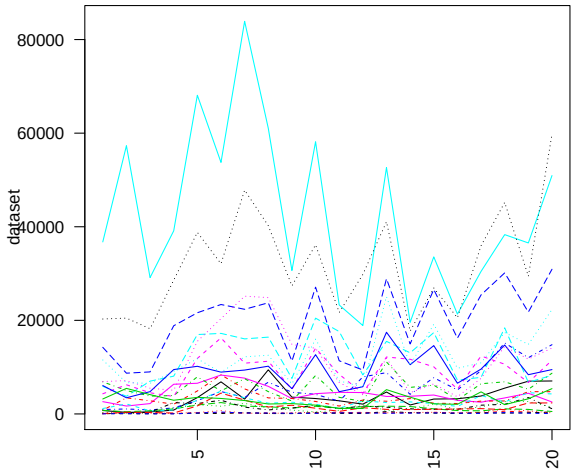
All the following analysis are made on a subset of the initial dataset. For each state and for each year, the sum of the total fires per month is calculated. Say also x and y in order to apply the models.

The dataset can be visualized in the below map. The intensity of the colour is proportional to the mean number of fires among the years 1998 - 2017, as displayed in the legend on the right.

In this dataset there is a big eterogeneity between the states. In the following boxplots, the mean and the variability between the number of fires in each state are displayed.



Moreover, in the following matplot every line corresponds to a different state. The y-axes represents the number of fires for that state while the x-axes represent the years (from 1998 to 2017). It can be seen that there is not a linear increasing of the number of fires among the years.



4 Methods

4.1 Choice of models

Because of the hierarchical structure of the data, building a hierarchical model was a natural choice. Since there are massive differences in variance and mean between the states, a pooled model can not perform very well in analyzing a single state. Because of this, we have focused on comparing separate and hierarchical models.

First, a normal model is introduced, as it is a good basis to build further models on and it provides a good baseline for further results. Because of the aforementioned qualities of the dataset (mean and variance) the next model we built was a negative binomial model, which should fit the dataset better. A separate and hierarchical versions of this model proved to model our data better than their normal counterparts. The last model we included was a regression model that utilizes the negative binomial distribution and predicts the amount of fires for the year 2018 in each of the states.

We also experimented with additional models that we decided to leave out because of their poor performance or the lack of sensible results. These models include a poisson model, pooled versions of the used models, and regression models that use a different distribution or alternative parametrizations that stan provides.

4.2 Choice of priors

We wanted to capture the large variance in the dataset with our choice of priors. For the normal models we used a prior distribution μ_0 $N(\mu, \sigma)$, where μ is the mean of the dataset and σ is the variance of the dataset.

For the negative binomial model, as well as the regression model, we used the priors αu and $\beta = 1.2$

4.3 Methods for comparing the models

5 Experiments and Results

5.1 Normal Model

5.1.1 Separate Normal Model

$$y_{ji} \sim N(\mu_j, \sigma_j)$$

```
# DEFINITION OF THE SEPARATE MODEL IN STAN

separate_code = "

data {
  int<lower=0> N;              // number of data points
  int<lower=0> K;              // number of groups
  int<lower=1,upper=K> x[N];  // group indicator
  vector[N] y;
}

parameters {
  vector[K] mu;                // group means
  vector<lower=0>[K] sigma;    // group stds
}

model {
  y ~ normal(mu[x], sigma[x]);
}

generated quantities {
  vector[K] y_state;
  vector[N] log_lik;

  for (i in 1:N)
    log_lik[i] = normal_lpdf(y[i] | mu[x[i]], sigma[x[i]]);

  for (i in 1:K)
    y_state[i]=normal_rng(mu[i], sigma[i]);
}

"
```

5.1.2 Pooled Normal Model

$$y_i \sim N(\mu, \sigma)$$

```
# DEFINITION OF THE POOLED MODEL IN STAN

pooled_code = "

data {
  int<lower=0> N;      // number of data points
  vector[N] y;         //
}
```

```
parameters {
  real mu;              // common mean
  real<lower=0> sigma; // common std
}

model {
  y ~ normal(mu, sigma);
}


generated quantities {
  real ypred;
  vector[N] log_lik;
  ypred = normal_rng(mu,sigma);
  for (i in 1:N)
    log_lik[i] = normal_lpdf(y[i] | mu, sigma);
}
"
```

5.1.3 Hierarchical Normal Model

$$y_{ji} \sim N(\overline{\mu} + \mu_j, \sigma)$$

```
# DEFINITION OF HIERARCHICAL MODEL IN STAN

hierarchical_code = "

data {
  int<lower=0> N;           // number of data points
  int<lower=0> K;           // number of groups
  int<lower=1,upper=K> x[N]; // group indicator
  vector[N] y;
}

parameters {
  real mu0;                 // prior mean
  real<lower=0> sigma0;     // prior std
  vector[K] mu;             // group means
  real<lower=0> sigma;      // common std
}

model {

  mu0 ~ normal(7933,25325);  // weakly informative prior
  sigma0 ~ cauchy(0,4);      // weakly informative prior
  mu ~ normal(mu0, sigma0);  // population prior with unknown parameters
  sigma ~ cauchy(0,4);       // weakly informative prior
  y ~ normal(mu[x], sigma);
}

generated quantities {
  real ypred;
  real mupred;
```

```
  vector[K] y_state;
  vector[N] log_lik;

  mupred = normal_rng(mu0,sigma0);
  ypred = normal_rng(mupred, sigma);

  for (i in 1:N)
    log_lik[i] = normal_lpdf(y[i] | mu[x[i]], sigma);

  for (i in 1:K)
    y_state[i]=normal_rng(mu[i], sigma);

}
"
```

```
## Warning: There were 34 divergent transitions after warmup. Increasing adapt_delta above 0.8 may help
## http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
```

```
## Warning: There were 263 transitions after warmup that exceeded the maximum treedepth. Increase max_t
## http://mc-stan.org/misc/warnings.html#maximum-treedepth-exceeded
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
## Warning: The largest R-hat is 1.58, indicating chains have not mixed.
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#r-hat
```

```
## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#bulk-ess
```

```
## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail quant
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#tail-ess
```

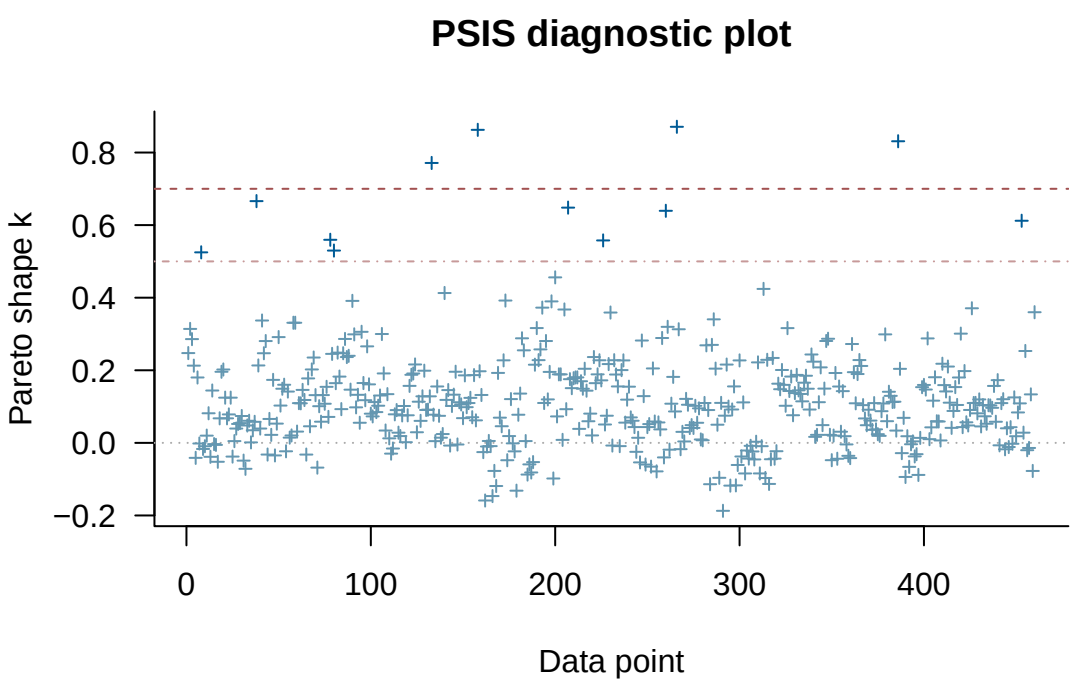
5.1.4 Results of the Normal Models

Here we have computed the PSIS-LOO elpd values and the k-values for each of the three normal models introduced in the last section as well as the effective number of parameters Peff for each of the three models.

```
# SEPARATE MODEL
loo_separate=loo(fit_separate)
```

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

```
plot(loo_separate)
```



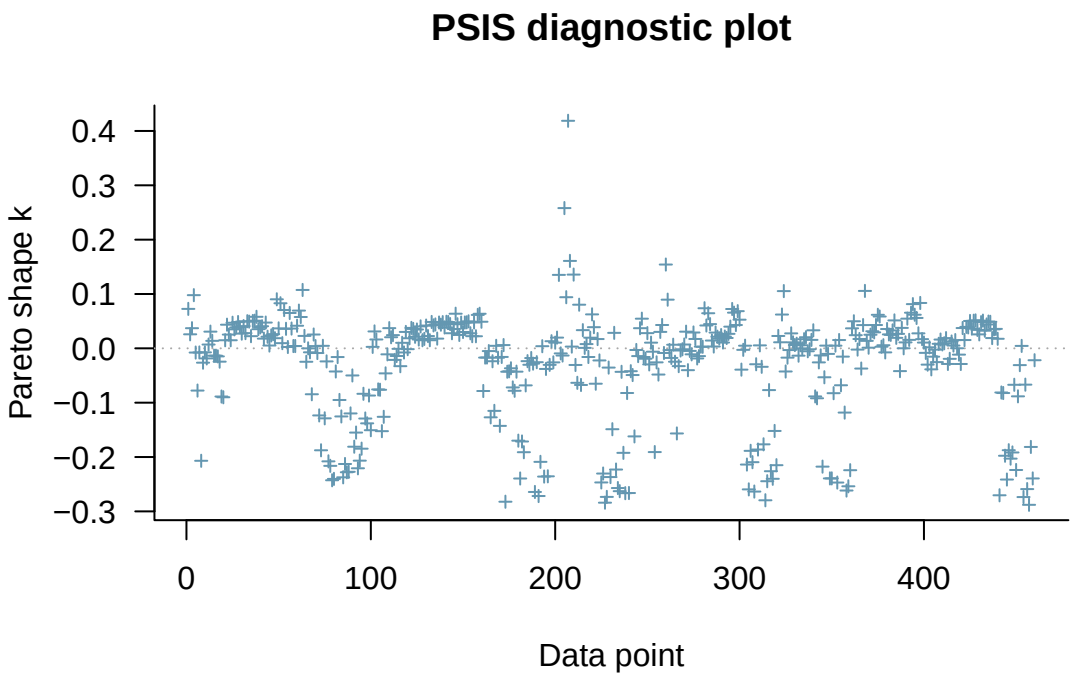

```
log_lik_s <- extract_log_lik(fit_separate, merge_chains = FALSE)
r_eff_s <- relative_eff(exp(log_lik_s))
loo_s <- loo(log_lik_s, r_eff = r_eff_s, save_psis=TRUE, cores=2 )
```

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

```
print(loo_s)
```

```
##
## Computed from 4000 by 460 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo  -4088.3 34.9
## p_loo        44.9  5.3
## looic      8176.5 69.8
## ------
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     448   97.4%   545
##  (0.5, 0.7]   (ok)         8    1.7%   75
##    (0.7, 1]   (bad)        4    0.9%   30
##    (1, Inf)   (very bad)   0    0.0%   <NA>
## See help('pareto-k-diagnostic') for details.
```

```
# POOLED MODEL
loo_pooled=loo(fit_pooled)
plot(loo_pooled)
```




```
log_lik_p <- extract_log_lik(fit_pooled, merge_chains = FALSE)
r_eff_p <- relative_eff(exp(log_lik_p))
loo_p <- loo(log_lik_p, r_eff = r_eff_p, save_psis=TRUE, cores=2 )
print(loo_p)
```

```
##
## Computed from 4000 by 460 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo  -4962.2 37.0
## p_loo         6.4  2.2
## looic      9924.4 74.0
## ------
## Monte Carlo SE of elpd_loo is 0.1.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

```
# HIERARCHICAL MODEL - normal
loo_hierarchical=loo(fit_hierarchical)
```

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

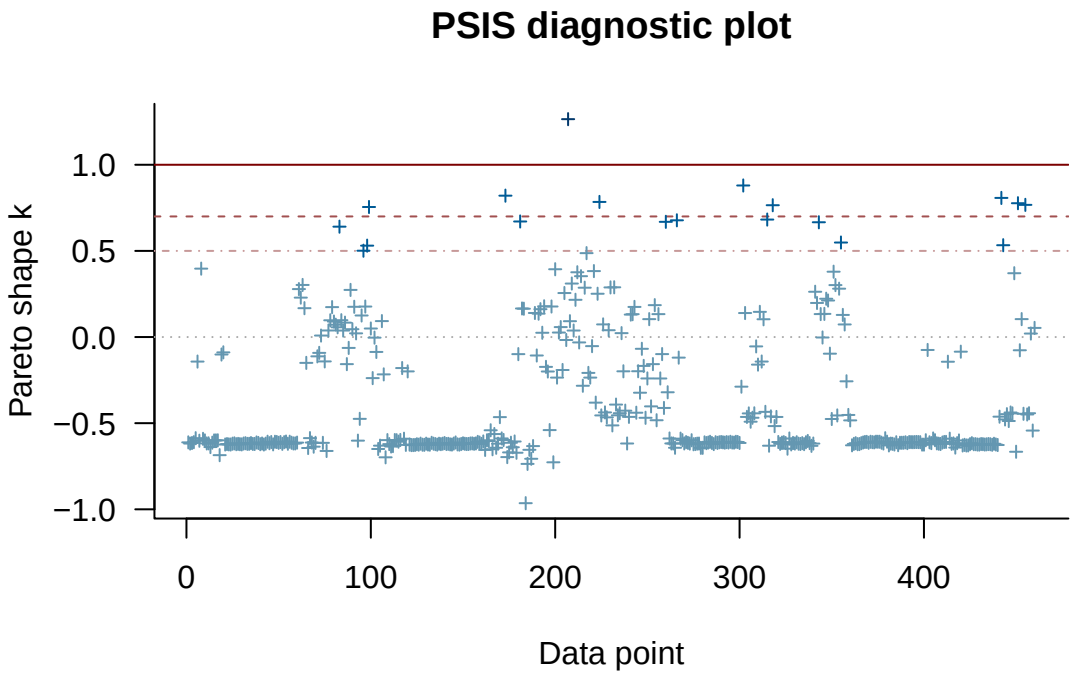
```
plot(loo_hierarchical)

log_lik_h <- extract_log_lik(fit_hierarchical, merge_chains = FALSE)
r_eff_h <- relative_eff(exp(log_lik_h))
loo_h <- loo(log_lik_h, r_eff = r_eff_h, save_psis=TRUE, cores=2 )
```

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

```
print(loo_h)
```

```
##
## Computed from 4000 by 460 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo  -4781.8 47.6
## p_loo       152.4 19.3
## looic      9563.7 95.3
## ------
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     441   95.9%   0
##  (0.5, 0.7]   (ok)        10    2.2%   0
##    (0.7, 1]   (bad)        8    1.7%   5
##    (1, Inf)   (very bad)   1    0.2%   0
## See help('pareto-k-diagnostic') for details.
```

```
plot(loo_h)
```




```
# ESTIMATION OF PSIS-LOO values

loo_s$estimates[1] #separate
```

```
## [1] -4088.252
```

```
loo_p$estimates[1] #pooled
```

```
## [1] -4962.186
```

```
loo_h$estimates[1] #hierarchical - normal
```

```
## [1] -4781.836
```

Next, we have computed the effective number of parameters Peff for each of the three models.

Answer: Equation (7.15) in the book:

$$p_{loo-cv} = lppd - lppd_{loo-cv}$$

where

$$lppd_{loo-cv}$$

is the PSIS-LOO value (sum of the LOO log densities) (calculated above in point 2)

and

$$lppd = \sum_{i=1}^{n} \log(\frac{1}{S} \sum_{s=1}^{S} p(y_i|\theta^s))$$

The lppd of the observed data y is an overestimate of the elppd for future data.

```
S=4000
n=20*23

# ESTIMATION OF lppd

# SEPARATE
vector_s=rep(0,n)
for(i in 1:n)
 vector_s[i]=log(1/S*(sum(exp(samples_s$log_lik[,i]))))

# POOLED
vector_p=rep(0,n)
for(i in 1:n)
 vector_p[i]=log(1/S*(sum(exp(samples_p$log_lik[,i]))))


# HIERARCHICAL - normal
vector_h=rep(0,n)
for(i in 1:n)
 vector_h[i]=log(1/S*(sum(exp(samples_h$log_lik[,i]))))
```

```
#RESULTING VALUES FOR peff

peff_s = sum(vector_s) - loo_s$estimates[1]
peff_p = sum(vector_p) -loo_p$estimates[1]
peff_h = sum(vector_h) - loo_h$estimates[1]

peff_s #separate
```

```
## [1] 44.91247
```

```
peff_p #pooled
```

```
## [1] 6.430996
```

```
peff_h #hierarchical normal
```

```
## [1] 152.356
```

5.2.1 Separate Negative Binomial Model

As discussed in the methods section of this document, the data has a higher variance than mean. Because of this we applied a negative binomial model to the data.

```
separate_negative_bin = "

data {
  int<lower=0> N;           // number of data points
  int<lower=0> K;           // number of groups
  int<lower=1,upper=K> x[N]; // group indicator
  int<lower=0> y[N];
}

parameters {
  real<lower=0> alpha[K];
  real<lower=0> beta[K];
}

model {
  alpha ~ exponential(0.0006303); //change this
  beta ~ exponential(1.2);
  y ~ neg_binomial(alpha[x], beta[x]);
}

generated quantities {
  int<lower=0> y_rep[K];
  vector[N] log_lik;

  for (i in 1:N)
    log_lik[i] = neg_binomial_lpmf(y[i] | alpha[x[i]], beta[x[i]]);

  for (i in 1:K)
    y_rep[i] = neg_binomial_rng(alpha[i], beta[i]);
}
"
```

5.2.2 Hierarchical Negative Binomial Model

```
 # DEFINITION OF THE HIERARCHICAL MODEL IN STAN

hierarchical_negative_bin = "

data {
  int<lower=0> N;           // number of data points
  int<lower=0> K;           // number of groups
  int<lower=1,upper=K> x[N]; // group indicator
  int<lower=0> y[N];
}
```

```
parameters {
  real alpha;
  real<lower=0> beta[K];
}

model {
  alpha ~ exponential(0.0006303);
  beta ~ exponential(1.2);
  y ~ neg_binomial(alpha, beta[x]);
}

generated quantities {
  int<lower=0> y_rep[K];
  vector[N] log_lik;

  for (i in 1:N)
    log_lik[i] = neg_binomial_lpmf(y[i] | alpha, beta[x[i]]);

  for (i in 1:K)
    y_rep[i] = neg_binomial_rng(alpha, beta[i]);
}
"
```

5.2.3 Results of the Negative Binomial Models

```
# Separate Model
samples_s = extract(object=separate_neg_binomial_fit, permuted = TRUE, inc_warmup = FALSE, include = TRU

log_lik_s <- extract_log_lik(separate_neg_binomial_fit, merge_chains = FALSE)
r_eff_s <- relative_eff(exp(log_lik_s))
loo_model_s <- loo(log_lik_s, r_eff = r_eff_s, save_psis=TRUE, cores=2 )
```

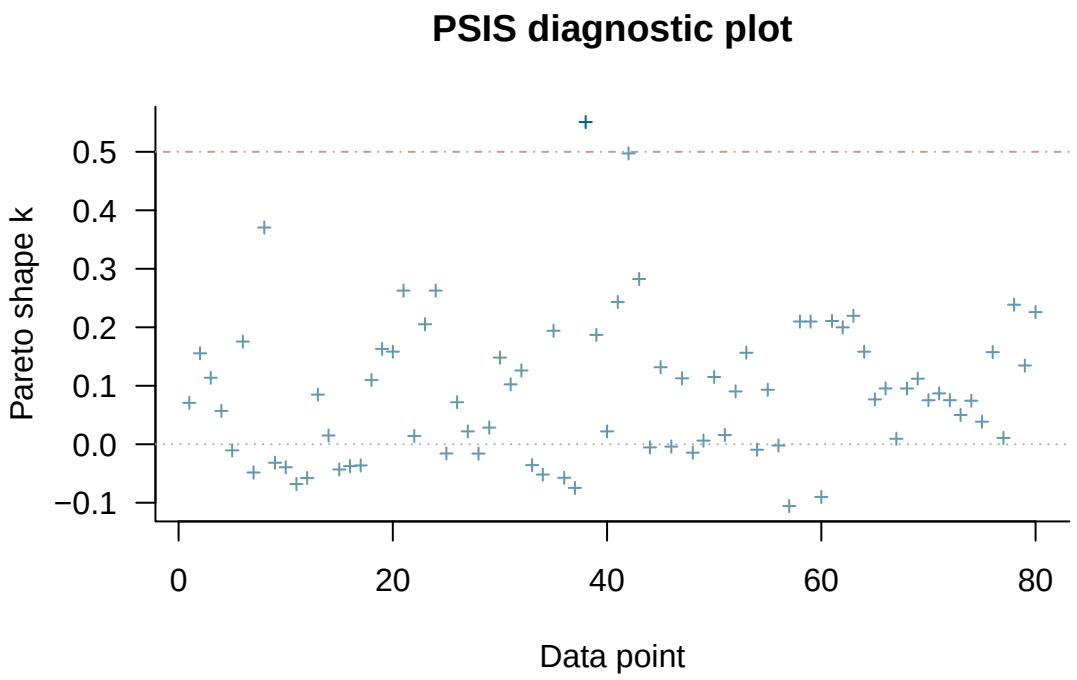
```
## Warning: Some Pareto k diagnostic values are slightly high. See help('pareto-k-diagnostic') for deta
```

```
print(loo_model_s)
```

```
##
## Computed from 4000 by 80 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -665.8 14.4
## p_loo         8.7  1.7
## looic      1331.7 28.8
## ------
## Monte Carlo SE of elpd_loo is 0.1.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     79    98.8%   441
##  (0.5, 0.7]   (ok)        1     1.2%   278
##    (0.7, 1]   (bad)       0     0.0%   <NA>
##    (1, Inf)   (very bad)  0     0.0%   <NA>
##
## All Pareto k estimates are ok (k < 0.7).
```

```
## See help('pareto-k-diagnostic') for details.
```

```
plot(loo_model_s)
```



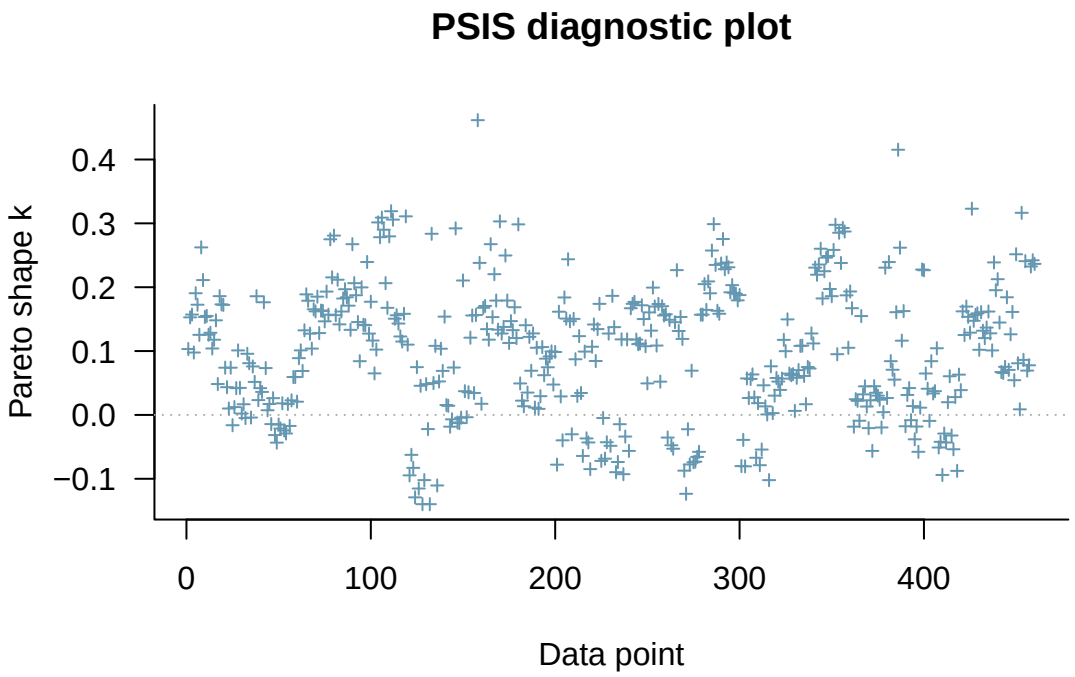

```
# PSIS-LOO values
loo_model_s$estimates[1]
```

```
## [1] -665.832
```

```
# Hierarchical Model
samples_h = extract(object=hierarchical_neg_binomial_fit, permuted = TRUE, inc_warmup = FALSE, include =

log_lik_h <- extract_log_lik(hierarchical_neg_binomial_fit, merge_chains = FALSE)
r_eff_h <- relative_eff(exp(log_lik_h))
loo_model_h <- loo(log_lik_h, r_eff = r_eff_h, save_psis=TRUE, cores=2 )
print(loo_model_h)
```

```
##
## Computed from 4000 by 460 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo  -4062.9 36.3
## p_loo        24.3  2.1
## looic      8125.8 72.7
## ------
## Monte Carlo SE of elpd_loo is 0.1.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

```r
plot(loo_model_h)
```




```r
# PSIS-LOO values
loo_model_h$estimates[1]
```

```
## [1] -4062.895
```

5.3.1 Hierarchical Regression Model

```r
# DEFINITION OF THE MODEL IN STAN

hierarchical_regression = "

data {
  int<lower=0> N;      //number of datapoints
  int<lower=0> K;      //number of groups
  vector[N] x;         //predictor (year)
  int<lower=0> y[N];   //response (n of fires)
  real xpred;          //regression predictor
}
parameters {
  real<lower=0> alpha;
  real<lower=0> beta;
  real<lower=0> phi;
}
model {
  phi ~ exponential(1.2);
  alpha ~ exponential(0.0006303);
  beta ~ exponential(0.0006303);
```

```
  y ~ neg_binomial(alpha + beta * x, phi);
}

generated quantities {
  int<lower=0> ypred[K];
  vector[N] log_lik;

  for (i in 1:K)
    ypred[i] = neg_binomial_rng(alpha + beta * xpred, phi);

  for (i in 1:N)
    log_lik[i] = neg_binomial_lpmf(y[i] | alpha, beta);
}
"
```

Table 1: Comparison between the two best fitted models

	ME	ME_ratio	MSE
LINEAR - reduced	72.036	24.36812	501.0531
MIXED - reduced	68.743	23.25400	535.1980

Reference

- https://www.kaggle.com/gustavomodelli/forest-fires-in-brazil
- https://datascienceplus.com/bayesian-regression-with-stan-beyond-normality/
- https://mc-stan.org/docs/2_20/functions-reference/nbalt.html
- https://mc-stan.org/loo/reference/loo-glossary.html
- **https://github.com/avehtari/BDA_R_demos/blob/master/demos_rstan/ppc/poisson-simple.stan**