House selling price prediction

Anonymous

Contents

1. Introduction

Online services such as zillow zestimates [1] provide accuarate information on how much houses sell for using gathered data, providing useful information for the realtor and the person selling the house. This notebook explores the possibilities on using stan to build regression models to predict housing prices on an zipcode level.

[1] https://www.zillow.com/zestimate/

```
library(rstan)
options(mc.cores = 4)#parallel::detectCores())
library(ggplot2)
library(matrixStats)
library(dplyr)
library(GGally)
library(corrplot)
library(reshape2)
library(ElemStatLearn)
library(glmnet)
library(plotmo)
library(Metrics)
library(bayesplot)
library(loo)
set.seed(42)
```

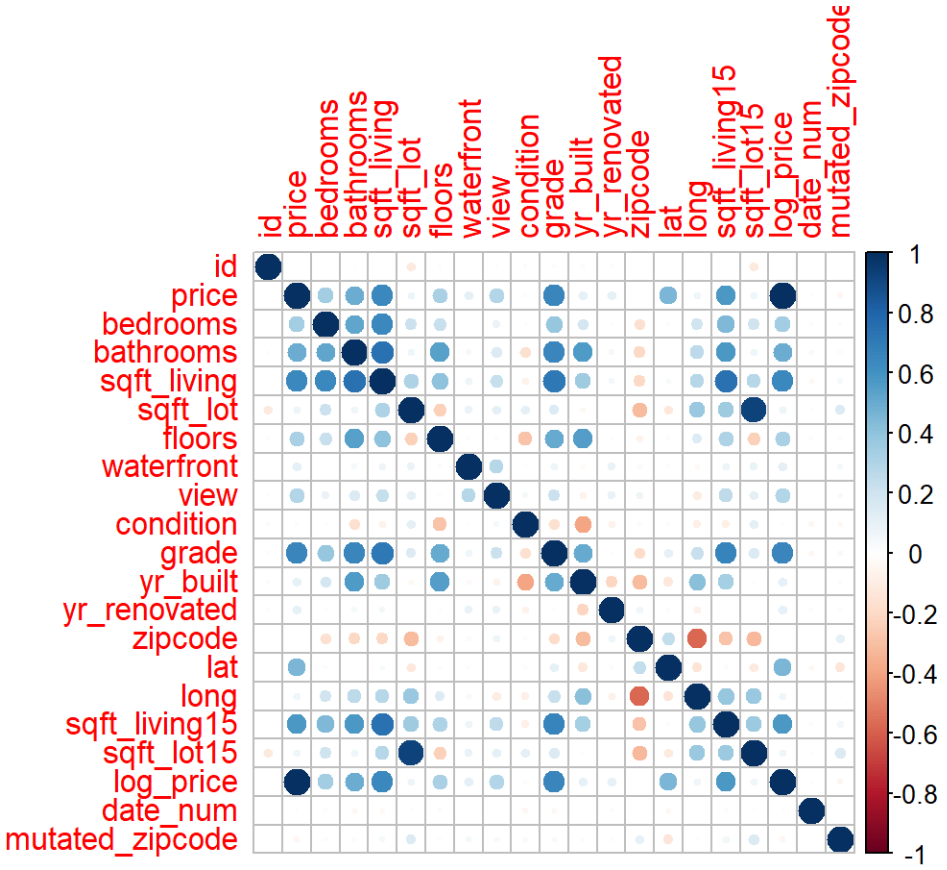
2. Dataset description

For the prediction task we have chosen House Sales in King County, USA dataset [2], which provides data for the houses sold between May 2014 / May 2015 in the area in an regression friendly form.
[2] https://www.kaggle.com/harlfoxem/housesalesprediction

```
houseprice = read.csv("data/kc_house_data.csv", header = TRUE)

#suffle rows to guarantee no row depencies
houseprice = houseprice[sample(nrow(houseprice)),]

#drop colinear columns sqft_living = sqft_above + sqft_basement
houseprice = subset(houseprice, select = -c(sqft_above, sqft_basement))

#transform the depended variable to log scale to  ensure better numerical accuracy
houseprice$log_price = log(houseprice$price)


datecol <- as.POSIXct(houseprice$date, format="%Y%m%dT%H%M%S")
houseprice$date_num = as.numeric(datecol)
unique_zips = unique(houseprice$zipcode)
houseprice$mutated_zipcode = match(houseprice$zipcode,  unique_zips)
head(houseprice)
```

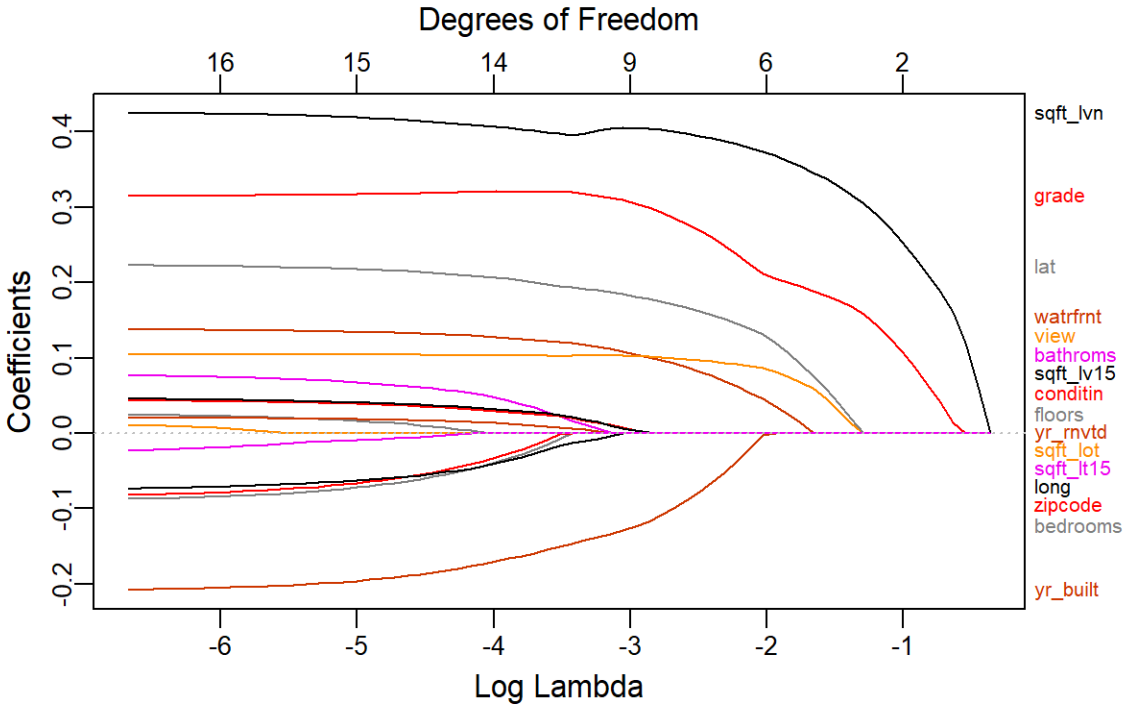
```
##               id            date   price bedrooms bathrooms sqft_living
## 19772 9523100731 20140930T000000  580000        3      2.50        1620
## 20253 8563010130 20140725T000000 1300000        3      2.50        3350
## 6184  9284801435 20141203T000000  471000        4      1.75        1760
## 17946 6672920150 20150406T000000  330000        3      2.00        1500
## 13868 7524950210 20150401T000000  910000        4      2.50        2770
## 11217 5592900105 20150213T000000  435000        4      1.75        2520
##       sqft_lot floors waterfront view condition grade yr_built
## 19772     1171      3          0    4         3     8     2008
## 20253     7752      1          0    0         3     9     2009
## 6184      5750      1          0    2         5     7     1962
## 17946    11233      1          0    0         3     7     1987
## 13868     9798      2          0    0         4     9     1986
## 11217     7200      1          0    2         5     7     1955
##       yr_renovated zipcode     lat     long sqft_living15 sqft_lot15
## 19772            0   98103 47.6681 -122.355          1620       1505
## 20253            0   98008 47.6263 -122.099          2570       7988
## 6184             0   98126 47.5521 -122.373          1860       5750
## 17946            0   98019 47.7279 -121.967          1580      14013
## 13868            0   98027 47.5620 -122.081          3040      11100
## 11217            0   98056 47.4835 -122.192          2360       7300
##       log_price   date_num mutated_zipcode
## 19772  13.27078 1412024400              1
## 20253  14.07787 1406235600              2
## 6184   13.06261 1417557600              3
## 17946  12.70685 1428267600              4
## 13868  13.72120 1427835600              5
## 11217  12.98310 1423778400              6
```

```
M <- cor(houseprice[-2], method="spearman")
corrplot(M, method = "circle")
```



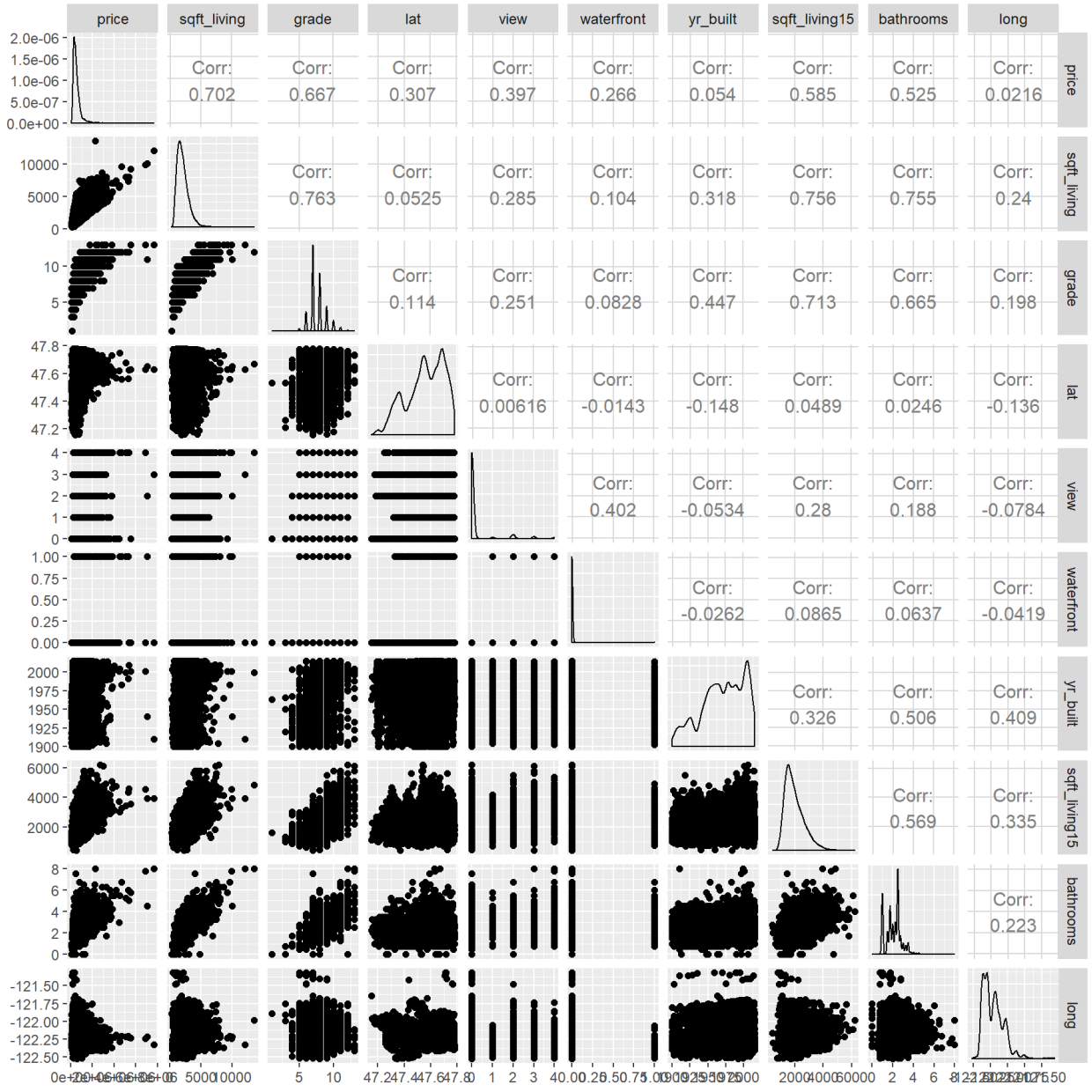
As the used dataset contains multiple predictors with linear and non-linear depencies, we use lasso regression to perform variable selection on the dataset to find an smaller subset of predictor variables to use in our model. This is neccessary for the purposes of the notebook to speed up the calculations and to better guarantee convergence. !Notice that Lasso regression estimates are calculated using an linear model so they might not be the best predictors for an non-linear model.

```
houseprice_scaled <- mutate_if(houseprice, is.numeric, list(~scale(.) %>% as.vector))

response = houseprice_scaled[3]
obs = houseprice_scaled[4:19]
ridge_regression <- glmnet(y=data.matrix(response), x=data.matrix(obs), alpha = 1)
plot_glmnet(ridge_regression, xvar = "lambda", label = TRUE)
```



From the lasso regression plot we can see that: sqft_living, grade, lat, view, waterfront, yr_built, sqft_living15, bathrooms and long are the 9 best variables for the model. When they are plotted in the matrix plot underneath, we can see that they chosen variables exhibit various linear and nonlinear effects on price. We choose all of these variables for the stan model except for the waterfront variable. Waterfront variable is not used as its binary nature causes problems in convergence in the case of polynomial model used.

```
ggpairs(houseprice  %>% select(price, sqft_living, grade, lat, view, waterfront, yr_built,
                               sqft_living15, bathrooms, long))
```



3. Model description

We fit two varying intercept regression models: an multiple linear and an multiple polynomial model. Varying intercept models are multilevel models, which make use of partial pooling

3.1 Prior choices

In [3] it is recommended to scale the parameters to unit scale and to use student-t distribution $t_\nu(0, 1)$, where $3 < \nu < 7$, as a prior for linear regression coefficients. Student-t distribution has heavier tails than a normal distribution, but less heavy tails than a cauchy distribution, making it able to predict further away values while still keeping most of the mass near the mean.

$$t_{\nu_{pdf}} = \frac{\Gamma\frac{\nu+1}{2}}{\sqrt{\nu\pi}\Gamma\frac{\nu}{2}}\left(1+\frac{x^2}{\nu}\right)^{-\frac{\nu+1}{2}}$$

[3] https://github.com/stan-dev/stan/wiki/Prior-Choice-Recommendations#prior-for-linear-regression

3.2 Stan models

We have built the models using stan radon case study [4] as a starting point to build our regression models. We have expanded on the varying intercept model of the example by adding multiple linear and polynomial terms into the model. In our model intercept parameters vary by the zipcode while the slope parameters are shared across zipcodes.

[4] https://mc-stan.org/users/documentation/case-studies/radon.html

Grouped multiple linear

$$y_i = \alpha_{j[i]} + \beta x_i + \epsilon_i$$

where $j = 1, \ldots, 70$ denotes the group of the observation. and $\epsilon_i \sim N(0, \sigma)$. The model can also be written as $y_i \sim N(\alpha_{j[i]} + \beta x_i, \sigma)$.

```
cat(readLines('models/grouped_multiple_linear.stan'), sep='\n')
```

```
## data {
##   int<lower=1> N;
##   int<lower=1> N_pred;
##   int<lower=1> N_groups;
##   int<lower=1> K;
##   vector[N] y;
##   matrix[N, K] X;
##   matrix[N_pred, K] X_pred;
##   int<lower=1> groups[N];
##   int<lower=1> groups_pred[N_pred];
## }
## parameters {
##   vector[N_groups] alpha;
##   vector[K] beta;
##   real<lower=0> sigma;
## }
## //transformed parameters {
## //  vector[N] mu;
## //  vector[N_pred] mu_pred;
## //  mu = alpha + X * beta;
## //  mu_pred = alpha + X_pred * beta;
## //}
## model {
##   real nu = 3;
##   alpha ~ student_t(nu,0,1);
##   beta ~ student_t(nu,0,1);
##   sigma ~ student_t(nu,0,1);
##   for (i in 1:N){
##     y[i] ~ normal(alpha[groups[i]] + X[i] * beta, sigma);
```

```
##   }
## }
## generated quantities {
##   vector[N_pred] y_pred;
##   vector[N] log_lik;
##   vector[N] y_rep ; // replicated data
##
##   for (i in 1:N_pred) {
##     y_pred[i] = normal_rng(alpha[groups_pred[i]] + X_pred[i] * beta, sigma);
##   }
##
##   for (i in 1:N) {
##     log_lik[i] = normal_lpdf(y[i] | alpha[groups[i]] + X[i] * beta, sigma);
##     y_rep[i] = normal_rng(alpha[groups[i]] + X[i] * beta, sigma);
##   }
## }
```

Grouped multiple polynomial

$$y_i = \alpha_{j[i]} + \beta x_i + \gamma x_i^2 + \epsilon_i$$

```
cat(readLines('models/grouped_multiple_polynomial.stan'), sep='\n')
```

```
## data {
##   int<lower=1> N;
##   int<lower=1> N_pred;
##   int<lower=1> N_groups;
##   int<lower=1> K;
##   vector[N] y;
##   matrix[N, K] X;
##   matrix[N, K] X_second;
##   matrix[N_pred, K] X_pred;
##   matrix[N_pred, K] X_pred_second;
##   int<lower=1> groups[N];
##   int<lower=1> groups_pred[N_pred];
## }
## parameters {
##   vector[N_groups] alpha;
##   vector[K] beta;
##   vector[K] beta_second;
##   real<lower=0> sigma;
## }
## //transformed parameters {
## //  vector[N] mu;
## //  vector[N_pred] mu_pred;
## //  mu = alpha + X * beta;
## //  mu_pred = alpha + X_pred * beta;
## //}
## model {
##   real nu = 3;
##   alpha ~ student_t(nu,0,1);
##   beta ~ student_t(nu,0,1);
##   beta_second ~ student_t(nu,0,1);
```

```
##   sigma ~ student_t(nu,0,1);
##   for (i in 1:N){
##     y[i] ~ normal(alpha[groups[i]] + X[i] * beta + X_second[i] * beta_second, sigma);
##   }
## }
## generated quantities {
##   vector[N_pred] y_pred;
##   vector[N] log_lik;
##   vector[N] y_rep;
##
##   for (i in 1:N_pred) {
##     y_pred[i] = normal_rng(alpha[groups_pred[i]] + X_pred[i] * beta+ X_pred_second[i] * beta_second,
##   }
##
##   for (i in 1:N) {
##     log_lik[i] = normal_lpdf(y[i] | alpha[groups[i]] + X[i] * beta + X_second[i] * beta_second, sigma
##     y_rep[i] = normal_rng( alpha[groups[i]] + X[i] * beta + X_second[i] * beta_second, sigma);
##   }
## }
##
```

3.3 Running the models

We train the models on 80%/20%-test split using 10000 first datapoints. Using all datapoints is possible, but R-will run out of memory while plotting.

```
usable_numeric_columns = c("sqft_living", "grade","view", "lat", "yr_built",
                           "sqft_living15","long", "bathrooms")
```

```
training_indices = 0:8000
testing_indices = 8001:10000

used_columns = usable_numeric_columns
target_column = c("log_price")
group_column =  c("mutated_zipcode")
original_target = houseprice[,target_column]
training_data = houseprice_scaled[training_indices,used_columns]
testing_data =  houseprice_scaled[testing_indices, used_columns]
training_target = houseprice_scaled[training_indices,target_column]
testing_target_scaled =  houseprice_scaled[testing_indices, target_column]
testing_target =  houseprice[testing_indices, target_column]

X_var = training_data
X_var_pred = testing_data
y_var = training_target
group_var = houseprice[training_indices,group_column]
group_var_pred = houseprice[testing_indices,group_column]

data_list = list(
  X = X_var,
  X_pred = X_var_pred,
  K = ncol(X_var),
  N = nrow(X_var),
```

```
  N_pred = nrow(X_var_pred),
  N_groups = length(unique_zips),
  y = y_var,
  groups = group_var,
  groups_pred  = group_var_pred
)
head(X_var)
```

```
##   sqft_living      grade       view         lat   yr_built sqft_living15
## 1  -0.5007396  0.2919089  4.9140157  0.77976752  1.2594678    -0.5348076
## 2   1.3828873  1.1426405 -0.3057524  0.47810123  1.2935122     0.8512619
## 3  -0.3483074 -0.5588228  2.3041317 -0.05739251 -0.3065744    -0.1846427
## 4  -0.6313958 -0.5588228 -0.3057524  1.21133795  0.5445355    -0.5931684
## 5   0.7513823  1.1426405 -0.3057524  0.01405477  0.5104911     1.5370016
## 6   0.4791819 -0.5588228  2.3041317 -0.55247163 -0.5448852     0.5448676
##         long  bathrooms
## 1 -1.0019545  0.5002092
## 2  0.8158614  0.5002092
## 3 -1.1297697 -0.4736105
## 4  1.7531727 -0.1490039
## 5  0.9436766  0.5002092
## 6  0.1554829 -0.4736105
```

Grouped multiple linear

```
multiple_linear_fit <- stan(file = 'models/grouped_multiple_linear.stan', data = data_list)
```

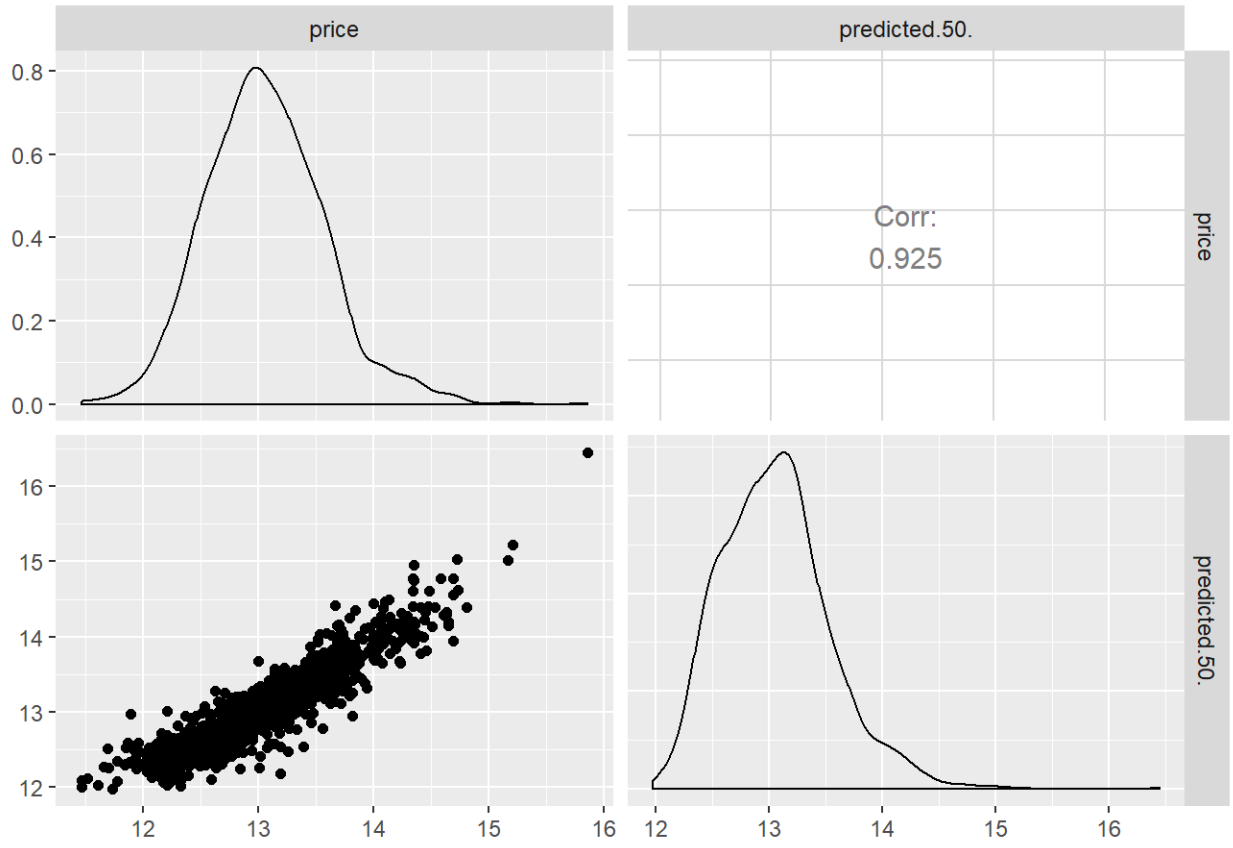
```
## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#bulk-ess
```

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

```
denormalize_results <- function(new_values, sd, mean){
  return (new_values * sd + mean)
}
orig_sd = sd(original_target)
orig_mean = mean(original_target)
```

```
predicted_draws = extract(multiple_linear_fit)$y_pred
predicted_raws = colQuantiles(predicted_draws, probs = c(0.05, 0.5, 0.95))
predicted_prices = denormalize_results(predicted_raws, orig_sd, orig_mean)
```

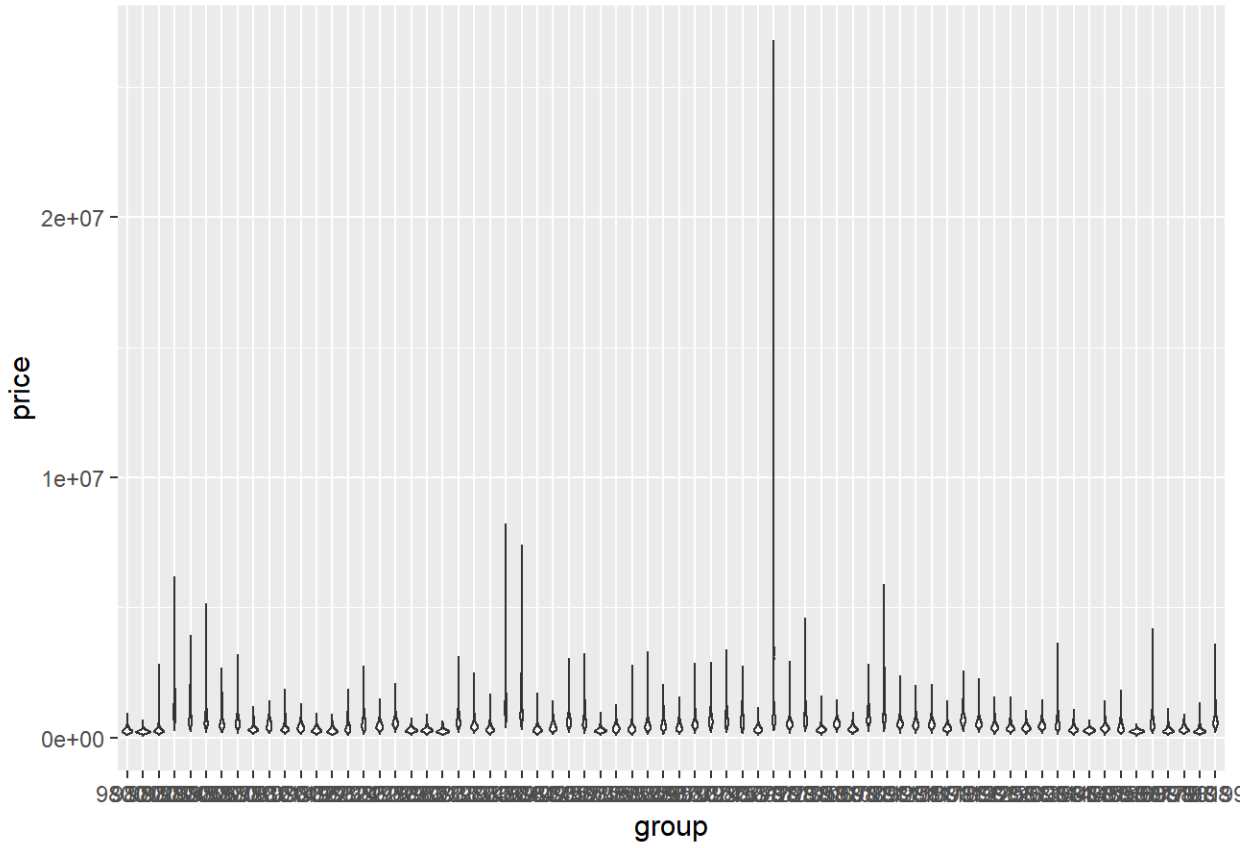
```
result_testing = data.frame(price = testing_target, predicted = predicted_prices)
ggpairs(result_testing, columns = c("price", "predicted.50."))
```




```
mae_lin = mae(exp(testing_target),exp(predicted_prices))
mae_lin
```

```
## [1] 150527.2
```

```
violin_predicted = extract(multiple_linear_fit)$y_pred
violin_predicted = exp(denormalize_results(violin_predicted, orig_sd, orig_mean))
#violin_predicted = exp(predicted_prices)
violin_groups = outer(1:nrow(violin_predicted), 1:ncol(violin_predicted),
                      FUN=function(r,c) unique_zips[group_var_pred[c]] )
violin_predicted = c(t(violin_predicted))
violin_groups = as.factor(c(t(violin_groups)))
violin_data_list_thing = data.frame(price=violin_predicted, group=violin_groups)

p <- ggplot(violin_data_list_thing, aes(x=group, y=price)) +
  geom_violin()
p
```



Grouped multiple polynomial

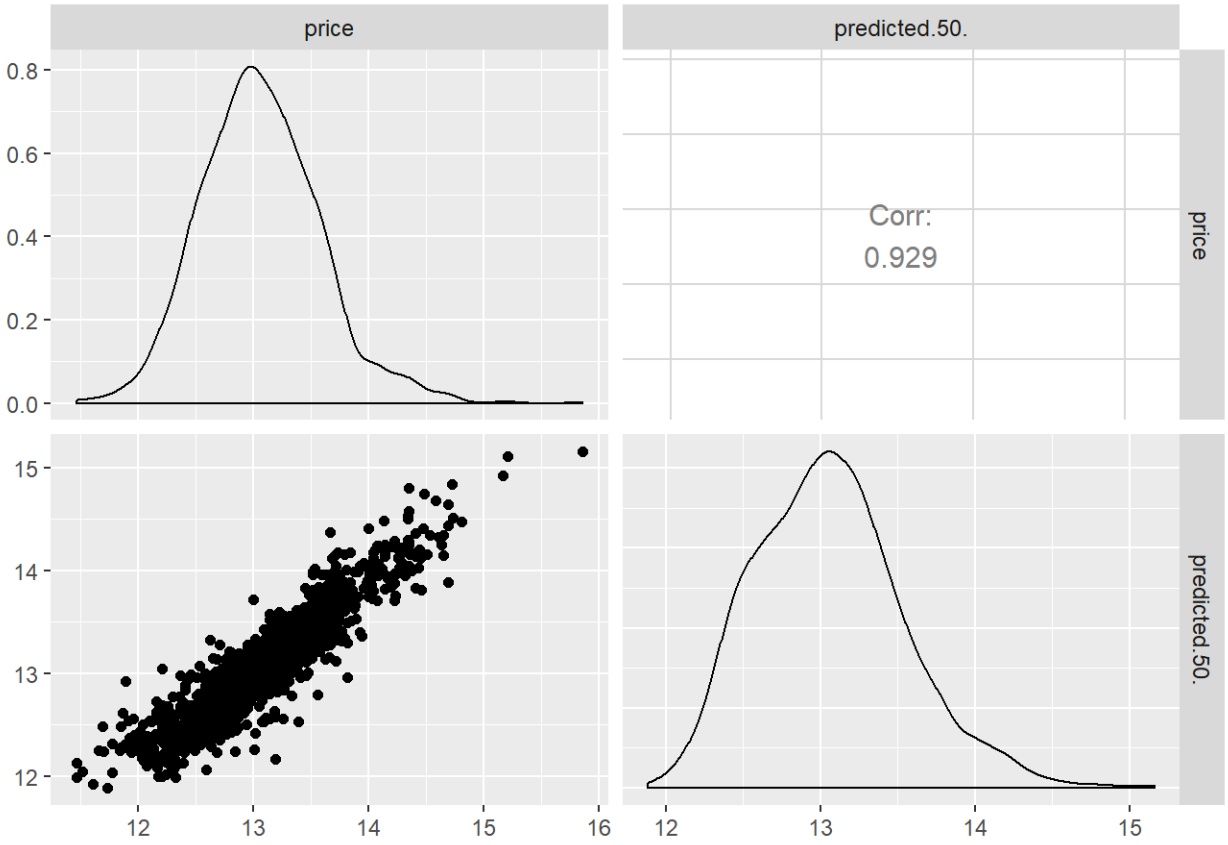
```
X_var_second = X_var^2
X_var_pred_second = X_var_pred^2

#not used (third degree polynomial model data)
X_var_third = X_var^3
X_var_pred_third = X_var_pred^3

data_list = list(
  X = X_var,
  X_second = X_var_second,
  X_third = X_var_third,
  X_pred = X_var_pred,
  X_pred_second = X_var_pred_second,
  X_pred_third = X_var_pred_third,
  K = ncol(X_var),
  N = nrow(X_var),
  N_pred = nrow(X_var_pred),
  N_groups = length(unique_zips),
  y = y_var,
  groups = group_var,
  groups_pred  = group_var_pred
)
```

```
multiple_polynomial_fit <- stan(file = 'models/grouped_multiple_polynomial.stan',
                                data = data_list)
```

```
## 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
```

```
predicted_draws = extract(multiple_polynomial_fit)$y_pred
predicted_raws = colQuantiles(predicted_draws, probs = c(0.05, 0.5, 0.95))
predicted_prices = denormalize_results(predicted_raws, orig_sd, orig_mean)
```

```
result_testing = data.frame(price = testing_target, predicted = predicted_prices)
ggpairs(result_testing, columns = c("price", "predicted.50."))
```




```
mae_pol = mae(exp(testing_target),exp(predicted_prices))
mae_pol
```

```
## [1] 144281.3
```

4. Convergence diagnostics

From the plots of the models we can see that all model parameters have converged.

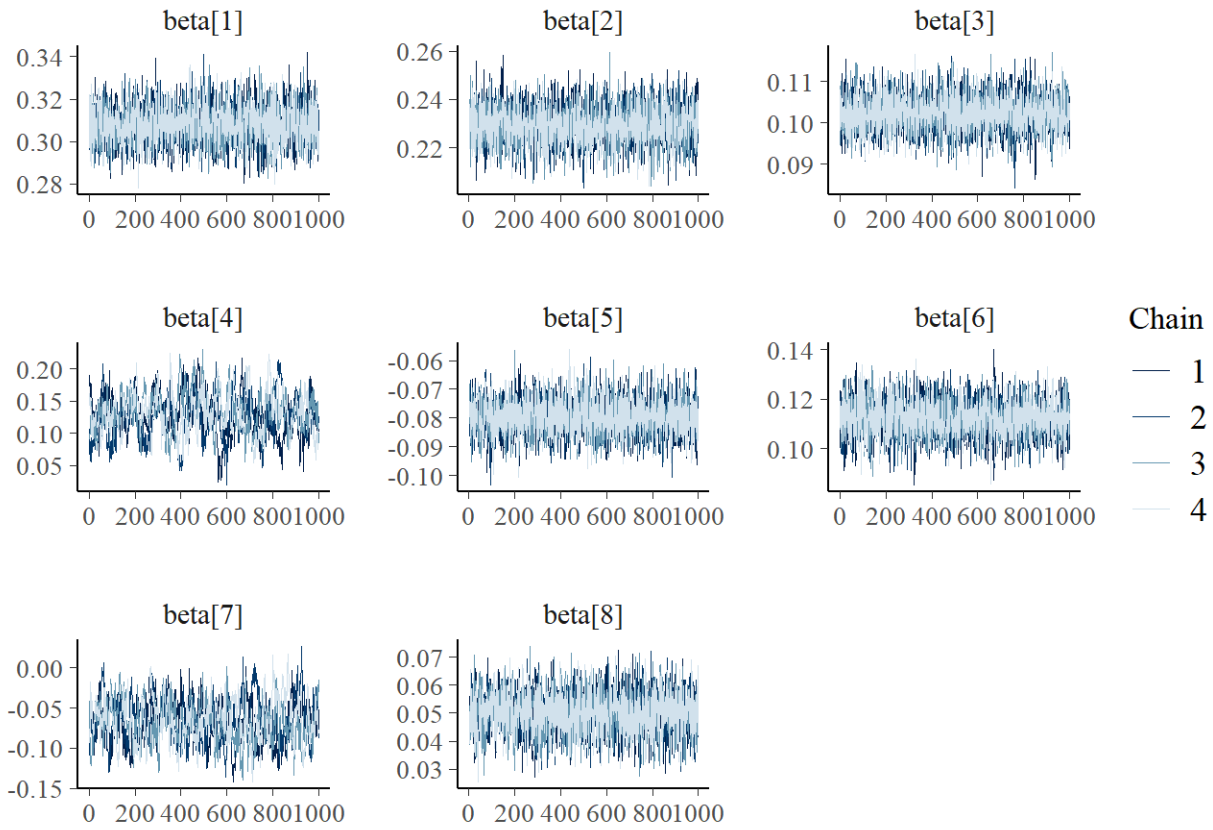
Grouped multiple linear

```
print(multiple_linear_fit, pars = c("alpha", "beta"))
```

```
## Inference for Stan model: grouped_multiple_linear.
## 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%   25%   50%   75% 97.5% n_eff Rhat
## alpha[1]   0.42    0.00 0.04  0.33  0.39  0.42  0.45  0.50   372 1.01
## alpha[2]   0.37    0.00 0.04  0.29  0.34  0.37  0.40  0.46   657 1.01
## alpha[3]   0.02    0.00 0.05 -0.07 -0.01  0.02  0.05  0.11   565 1.01
## alpha[4]  -0.25    0.00 0.08 -0.40 -0.30 -0.25 -0.19 -0.10   277 1.03
## alpha[5]   0.25    0.00 0.04  0.16  0.22  0.25  0.28  0.33   449 1.01
## alpha[6]  -0.20    0.00 0.03 -0.27 -0.22 -0.20 -0.18 -0.14  1111 1.00
## alpha[7]   0.52    0.00 0.04  0.44  0.49  0.52  0.55  0.60   373 1.02
## alpha[8]  -0.28    0.00 0.05 -0.39 -0.32 -0.28 -0.24 -0.18   297 1.02
## alpha[9]   0.29    0.00 0.03  0.24  0.28  0.29  0.31  0.35  1090 1.00
## alpha[10] -0.08    0.00 0.03 -0.14 -0.10 -0.08 -0.06 -0.02  1106 1.00
## alpha[11] -0.24    0.00 0.06 -0.35 -0.28 -0.24 -0.20 -0.12   296 1.02
## alpha[12] -0.22    0.00 0.06 -0.34 -0.26 -0.22 -0.18 -0.09   365 1.02
## alpha[13] -0.42    0.00 0.05 -0.51 -0.45 -0.42 -0.39 -0.33   461 1.01
## alpha[14]  0.88    0.00 0.04  0.79  0.85  0.88  0.91  0.97   913 1.00
## alpha[15]  0.27    0.00 0.04  0.19  0.24  0.27  0.30  0.36   294 1.03
## alpha[16]  0.65    0.00 0.05  0.56  0.62  0.65  0.68  0.74   603 1.01
## alpha[17]  0.05    0.00 0.05 -0.05  0.02  0.05  0.08  0.14   300 1.02
## alpha[18]  0.42    0.00 0.04  0.34  0.39  0.42  0.45  0.50   362 1.01
## alpha[19]  0.39    0.00 0.05  0.30  0.36  0.39  0.42  0.48   450 1.01
## alpha[20] -0.27    0.01 0.10 -0.45 -0.33 -0.27 -0.20 -0.08   269 1.02
## alpha[21] -0.09    0.00 0.04 -0.16 -0.11 -0.09 -0.07 -0.02   675 1.00
## alpha[22]  0.72    0.00 0.06  0.59  0.67  0.71  0.76  0.84  1349 1.00
## alpha[23] -0.53    0.00 0.05 -0.64 -0.57 -0.54 -0.50 -0.43   394 1.02
## alpha[24] -0.76    0.00 0.07 -0.90 -0.80 -0.76 -0.71 -0.62   222 1.03
## alpha[25] -0.28    0.00 0.06 -0.40 -0.32 -0.28 -0.24 -0.17   340 1.02
## alpha[26] -0.80    0.00 0.05 -0.89 -0.83 -0.80 -0.77 -0.71   649 1.01
## alpha[27]  0.69    0.00 0.06  0.58  0.65  0.69  0.73  0.80   706 1.00
## alpha[28] -0.42    0.00 0.04 -0.50 -0.45 -0.42 -0.40 -0.34   434 1.01
## alpha[29]  0.35    0.00 0.05  0.26  0.32  0.35  0.39  0.45   331 1.01
## alpha[30]  0.27    0.00 0.05  0.17  0.23  0.27  0.30  0.36   468 1.01
## alpha[31] -0.63    0.00 0.07 -0.77 -0.68 -0.63 -0.59 -0.50   246 1.03
## alpha[32]  0.35    0.00 0.05  0.25  0.32  0.35  0.39  0.45   392 1.01
## alpha[33]  0.50    0.00 0.05  0.40  0.47  0.50  0.54  0.60   463 1.01
## alpha[34] -0.47    0.00 0.05 -0.57 -0.50 -0.46 -0.43 -0.36   284 1.02
## alpha[35]  0.00    0.00 0.09 -0.17 -0.06  0.00  0.05  0.17   406 1.01
## alpha[36] -0.46    0.00 0.07 -0.60 -0.51 -0.46 -0.42 -0.33   270 1.02
## alpha[37] -0.70    0.00 0.06 -0.82 -0.74 -0.70 -0.66 -0.58   698 1.01
## alpha[38]  0.17    0.00 0.04  0.10  0.15  0.17  0.20  0.24  1101 1.00
## alpha[39] -0.32    0.00 0.06 -0.43 -0.36 -0.32 -0.29 -0.22   283 1.02
## alpha[40]  0.13    0.01 0.10 -0.06  0.07  0.13  0.20  0.31   348 1.01
## alpha[41] -0.04    0.00 0.05 -0.14 -0.08 -0.04 -0.01  0.05   326 1.02
## alpha[42]  0.45    0.00 0.06  0.34  0.42  0.45  0.49  0.56   538 1.01
## alpha[43]  0.19    0.00 0.05  0.10  0.16  0.19  0.22  0.28   358 1.02
## alpha[44] -0.49    0.00 0.04 -0.58 -0.52 -0.50 -0.46 -0.41   682 1.01
```

```
## alpha[45] -0.09    0.00 0.07 -0.22 -0.14 -0.09 -0.04  0.04   291 1.03
## alpha[46]  0.47    0.00 0.05  0.37  0.43  0.47  0.50  0.57  1504 1.00
## alpha[47] -0.64    0.00 0.07 -0.78 -0.69 -0.64 -0.59 -0.50   248 1.03
## alpha[48]  0.25    0.00 0.06  0.15  0.21  0.25  0.29  0.36   298 1.03
## alpha[49]  0.09    0.00 0.07 -0.06  0.04  0.09  0.14  0.23   301 1.01
## alpha[50] -0.58    0.00 0.06 -0.70 -0.62 -0.58 -0.55 -0.47   322 1.02
## alpha[51] -0.06    0.00 0.06 -0.17 -0.10 -0.06 -0.01  0.06   347 1.01
## alpha[52]  0.25    0.00 0.05  0.16  0.22  0.25  0.28  0.35   383 1.01
## alpha[53]  0.85    0.00 0.07  0.72  0.80  0.85  0.89  0.98  1226 1.00
## alpha[54] -0.15    0.00 0.05 -0.26 -0.19 -0.15 -0.12 -0.05   400 1.02
## alpha[55] -0.37    0.00 0.04 -0.45 -0.40 -0.37 -0.34 -0.28   555 1.01
## alpha[56] -0.03    0.00 0.06 -0.15 -0.08 -0.04  0.01  0.09   296 1.02
## alpha[57]  0.31    0.00 0.05  0.21  0.28  0.31  0.34  0.41  1487 1.00
## alpha[58] -0.79    0.00 0.07 -0.93 -0.84 -0.79 -0.74 -0.65   428 1.02
## alpha[59] -0.61    0.00 0.06 -0.72 -0.65 -0.61 -0.57 -0.50   302 1.03
## alpha[60] -0.50    0.00 0.08 -0.66 -0.56 -0.50 -0.45 -0.35   871 1.01
## alpha[61]  1.15    0.00 0.04  1.07  1.12  1.14  1.17  1.22  1158 1.01
## alpha[62]  0.18    0.00 0.09  0.00  0.12  0.18  0.24  0.36   536 1.01
## alpha[63]  1.28    0.00 0.09  1.10  1.22  1.28  1.34  1.45  4029 1.00
## alpha[64]  0.40    0.00 0.04  0.32  0.37  0.40  0.43  0.48  1112 1.00
## alpha[65]  0.77    0.00 0.04  0.70  0.74  0.77  0.79  0.84  5673 1.00
## alpha[66] -0.29    0.00 0.05 -0.39 -0.32 -0.29 -0.25 -0.19  1497 1.00
## alpha[67] -0.65    0.00 0.07 -0.79 -0.70 -0.65 -0.60 -0.51   279 1.03
## alpha[68] -0.60    0.00 0.04 -0.68 -0.63 -0.60 -0.58 -0.52   991 1.01
## alpha[69] -0.04    0.00 0.08 -0.21 -0.10 -0.04  0.01  0.12   368 1.01
## alpha[70] -0.06    0.00 0.09 -0.24 -0.12 -0.06  0.00  0.12   351 1.02
## beta[1]    0.31    0.00 0.01  0.29  0.30  0.31  0.31  0.33  3326 1.00
## beta[2]    0.23    0.00 0.01  0.21  0.22  0.23  0.23  0.24  4424 1.00
## beta[3]    0.10    0.00 0.00  0.09  0.10  0.10  0.11  0.11  7527 1.00
## beta[4]    0.13    0.00 0.03  0.07  0.11  0.13  0.15  0.19   200 1.04
## beta[5]   -0.08    0.00 0.01 -0.09 -0.08 -0.08 -0.08 -0.07  3537 1.00
## beta[6]    0.11    0.00 0.01  0.10  0.11  0.11  0.12  0.13  4535 1.00
## beta[7]   -0.07    0.00 0.03 -0.12 -0.08 -0.07 -0.05 -0.02   260 1.02
## beta[8]    0.05    0.00 0.01  0.03  0.05  0.05  0.06  0.06  3668 1.00
##
## Samples were drawn using NUTS(diag_e) at Sun Dec 08 21:04:32 2019.
## 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).
```

```
posterior_divergences <- as.array(multiple_linear_fit)
mcmc_trace(multiple_linear_fit, regex_pars = "beta")
```




```
#past tree depth
get_num_max_treedepth(multiple_linear_fit)
```

```
## [1] 0
```

Grouped multiple polynomial

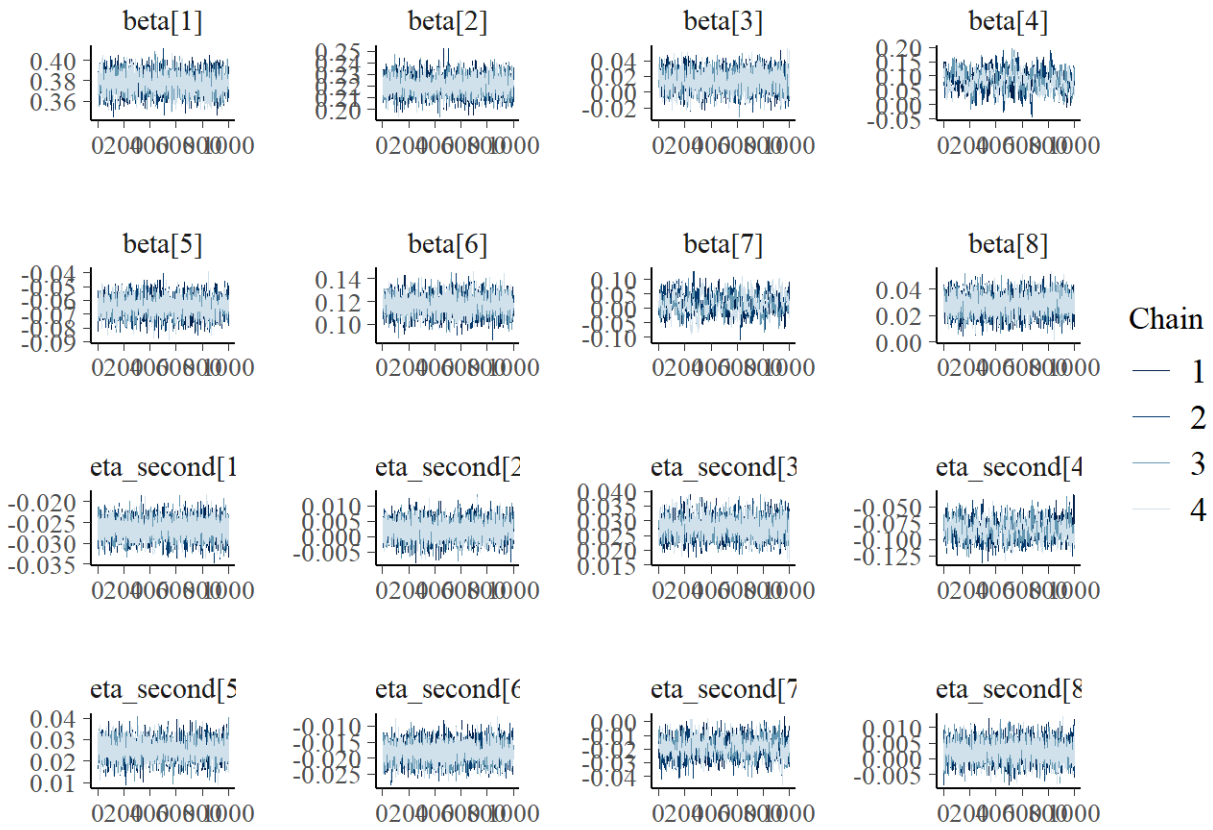
```
print(multiple_polynomial_fit,  pars = c("alpha", "beta", "beta_second"))
```

```
## Inference for Stan model: grouped_multiple_polynomial.
## 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%   25%   50%   75% 97.5% n_eff Rhat
## alpha[1]       0.58    0.00 0.05  0.48  0.55  0.58  0.61  0.68   274 1.02
## alpha[2]       0.34    0.00 0.05  0.25  0.31  0.34  0.37  0.43   868 1.00
## alpha[3]       0.15    0.00 0.06  0.04  0.11  0.15  0.19  0.27   395 1.02
## alpha[4]      -0.17    0.00 0.09 -0.34 -0.23 -0.17 -0.11 -0.01   472 1.00
## alpha[5]       0.16    0.00 0.04  0.08  0.13  0.16  0.19  0.24   456 1.01
## alpha[6]      -0.24    0.00 0.03 -0.30 -0.26 -0.24 -0.22 -0.17  1237 1.00
## alpha[7]       0.60    0.00 0.05  0.51  0.57  0.60  0.63  0.69   572 1.00
## alpha[8]       0.01    0.00 0.07 -0.12 -0.04  0.01  0.05  0.14   360 1.01
## alpha[9]       0.28    0.00 0.03  0.23  0.26  0.28  0.30  0.34  1014 1.00
```

```
## alpha[10]      -0.06    0.00 0.03 -0.12 -0.08 -0.06 -0.04  0.00   807 1.01
## alpha[11]      -0.26    0.00 0.06 -0.38 -0.30 -0.26 -0.22 -0.15   343 1.01
## alpha[12]      -0.03    0.00 0.08 -0.17 -0.08 -0.03  0.02  0.12   523 1.00
## alpha[13]      -0.31    0.00 0.06 -0.41 -0.34 -0.31 -0.27 -0.20   411 1.01
## alpha[14]       0.96    0.00 0.05  0.86  0.92  0.96  0.99  1.05   540 1.01
## alpha[15]       0.30    0.00 0.05  0.21  0.27  0.30  0.34  0.40   498 1.00
## alpha[16]       0.75    0.00 0.05  0.65  0.71  0.75  0.78  0.85   508 1.01
## alpha[17]       0.21    0.00 0.06  0.10  0.17  0.21  0.24  0.31   484 1.00
## alpha[18]       0.57    0.00 0.05  0.49  0.54  0.57  0.60  0.67   333 1.01
## alpha[19]       0.51    0.00 0.06  0.39  0.47  0.51  0.55  0.63   345 1.02
## alpha[20]       0.06    0.01 0.12 -0.18 -0.02  0.07  0.14  0.29   447 1.01
## alpha[21]      -0.15    0.00 0.04 -0.22 -0.18 -0.15 -0.13 -0.09   717 1.01
## alpha[22]       0.78    0.00 0.07  0.66  0.74  0.78  0.83  0.91   752 1.00
## alpha[23]      -0.51    0.00 0.05 -0.61 -0.55 -0.51 -0.48 -0.41   521 1.01
## alpha[24]      -0.46    0.00 0.08 -0.63 -0.52 -0.46 -0.41 -0.29   317 1.02
## alpha[25]      -0.07    0.00 0.07 -0.21 -0.12 -0.07 -0.02  0.07   504 1.00
## alpha[26]      -0.71    0.00 0.05 -0.81 -0.75 -0.71 -0.68 -0.61   766 1.01
## alpha[27]       0.82    0.00 0.06  0.70  0.77  0.82  0.86  0.94   356 1.02
## alpha[28]      -0.46    0.00 0.04 -0.54 -0.49 -0.46 -0.43 -0.38   515 1.01
## alpha[29]       0.57    0.00 0.06  0.45  0.53  0.57  0.61  0.69   245 1.02
## alpha[30]       0.39    0.00 0.06  0.27  0.35  0.39  0.43  0.51   390 1.02
## alpha[31]      -0.42    0.00 0.08 -0.57 -0.47 -0.42 -0.37 -0.27   374 1.01
## alpha[32]       0.24    0.00 0.05  0.13  0.20  0.24  0.27  0.34   442 1.01
## alpha[33]       0.70    0.00 0.06  0.58  0.66  0.70  0.74  0.82   277 1.02
## alpha[34]      -0.42    0.00 0.06 -0.53 -0.46 -0.42 -0.39 -0.32   399 1.01
## alpha[35]       0.05    0.00 0.09 -0.12 -0.01  0.05  0.10  0.21   503 1.01
## alpha[36]      -0.31    0.00 0.08 -0.46 -0.36 -0.31 -0.26 -0.17   409 1.01
## alpha[37]      -0.63    0.00 0.06 -0.75 -0.67 -0.63 -0.59 -0.51   880 1.00
## alpha[38]       0.21    0.00 0.04  0.14  0.19  0.21  0.24  0.29   446 1.02
## alpha[39]      -0.03    0.00 0.07 -0.17 -0.08 -0.03  0.02  0.11   414 1.01
## alpha[40]       0.04    0.00 0.09 -0.13 -0.02  0.04  0.10  0.22   434 1.01
## alpha[41]       0.17    0.00 0.06  0.06  0.13  0.16  0.20  0.28   384 1.01
## alpha[42]       0.64    0.00 0.07  0.52  0.60  0.64  0.69  0.77   304 1.02
## alpha[43]       0.14    0.00 0.05  0.04  0.10  0.14  0.17  0.23   519 1.01
## alpha[44]      -0.50    0.00 0.04 -0.59 -0.53 -0.50 -0.47 -0.42   942 1.00
## alpha[45]       0.05    0.00 0.08 -0.11  0.00  0.05  0.11  0.21   508 1.00
## alpha[46]       0.47    0.00 0.05  0.37  0.44  0.47  0.50  0.56  1752 1.00
## alpha[47]      -0.38    0.00 0.08 -0.53 -0.43 -0.38 -0.33 -0.22   368 1.02
## alpha[48]       0.29    0.00 0.06  0.16  0.24  0.29  0.33  0.41   474 1.00
## alpha[49]      -0.05    0.00 0.07 -0.19 -0.10 -0.05 -0.01  0.08   395 1.01
## alpha[50]      -0.53    0.00 0.06 -0.64 -0.56 -0.53 -0.49 -0.41   479 1.01
## alpha[51]       0.26    0.00 0.08  0.11  0.20  0.26  0.31  0.41   334 1.01
## alpha[52]       0.17    0.00 0.05  0.07  0.13  0.17  0.20  0.26   455 1.01
## alpha[53]       0.97    0.00 0.07  0.83  0.92  0.97  1.02  1.11   639 1.01
## alpha[54]      -0.05    0.00 0.06 -0.18 -0.09 -0.06 -0.01  0.07   451 1.01
## alpha[55]      -0.24    0.00 0.05 -0.34 -0.28 -0.24 -0.21 -0.14   402 1.02
## alpha[56]       0.13    0.00 0.08 -0.02  0.08  0.13  0.18  0.27   523 1.00
## alpha[57]       0.29    0.00 0.05  0.19  0.26  0.29  0.33  0.39  1512 1.00
## alpha[58]      -0.65    0.00 0.07 -0.79 -0.70 -0.65 -0.60 -0.51   588 1.01
## alpha[59]      -0.46    0.00 0.06 -0.58 -0.50 -0.46 -0.42 -0.35   382 1.01
## alpha[60]      -0.40    0.00 0.09 -0.57 -0.46 -0.40 -0.34 -0.23   947 1.01
## alpha[61]       1.18    0.00 0.04  1.11  1.16  1.18  1.21  1.26  1699 1.00
## alpha[62]       0.07    0.00 0.08 -0.09  0.02  0.07  0.13  0.24   629 1.01
## alpha[63]       1.51    0.00 0.09  1.34  1.45  1.51  1.57  1.68  3206 1.00
```

```
## alpha[64]       0.45    0.00 0.05  0.36  0.42  0.45  0.48  0.54   583 1.01
## alpha[65]       0.81    0.00 0.04  0.74  0.79  0.82  0.84  0.88  4353 1.00
## alpha[66]      -0.24    0.00 0.05 -0.34 -0.28 -0.24 -0.21 -0.14   753 1.01
## alpha[67]      -0.45    0.00 0.08 -0.60 -0.50 -0.45 -0.39 -0.29   442 1.01
## alpha[68]      -0.59    0.00 0.04 -0.68 -0.62 -0.59 -0.56 -0.51  1304 1.00
## alpha[69]       0.18    0.01 0.10 -0.03  0.11  0.18  0.25  0.39   389 1.01
## alpha[70]      -0.03    0.00 0.09 -0.21 -0.09 -0.03  0.03  0.15   569 1.00
## beta[1]         0.38    0.00 0.01  0.36  0.37  0.38  0.38  0.40  2874 1.00
## beta[2]         0.22    0.00 0.01  0.21  0.22  0.22  0.23  0.24  3540 1.00
## beta[3]         0.01    0.00 0.01 -0.01  0.01  0.01  0.02  0.04  2544 1.00
## beta[4]         0.09    0.00 0.03  0.02  0.06  0.09  0.11  0.15   311 1.01
## beta[5]        -0.06    0.00 0.01 -0.08 -0.07 -0.06 -0.06 -0.05  3291 1.00
## beta[6]         0.12    0.00 0.01  0.10  0.11  0.12  0.12  0.13  3732 1.00
## beta[7]         0.02    0.00 0.03 -0.05  0.00  0.02  0.04  0.08   255 1.02
## beta[8]         0.03    0.00 0.01  0.01  0.02  0.03  0.03  0.04  3683 1.00
## beta_second[1] -0.03    0.00 0.00 -0.03 -0.03 -0.03 -0.02 -0.02  5148 1.00
## beta_second[2]  0.00    0.00 0.00  0.00  0.00  0.00  0.00  0.01  6794 1.00
## beta_second[3]  0.03    0.00 0.00  0.02  0.03  0.03  0.03  0.03  2532 1.00
## beta_second[4] -0.08    0.00 0.01 -0.11 -0.09 -0.08 -0.07 -0.06   652 1.00
## beta_second[5]  0.03    0.00 0.00  0.02  0.02  0.03  0.03  0.03  3067 1.00
## beta_second[6] -0.02    0.00 0.00 -0.02 -0.02 -0.02 -0.02 -0.01  4756 1.00
## beta_second[7] -0.02    0.00 0.01 -0.03 -0.02 -0.02 -0.01  0.00   549 1.01
## beta_second[8]  0.00    0.00 0.00  0.00  0.00  0.00  0.00  0.01  5836 1.00
##
## Samples were drawn using NUTS(diag_e) at Sun Dec 08 21:15:37 2019.
## 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).
```

```
mcmc_trace(multiple_polynomial_fit, regex_pars = "beta")
```



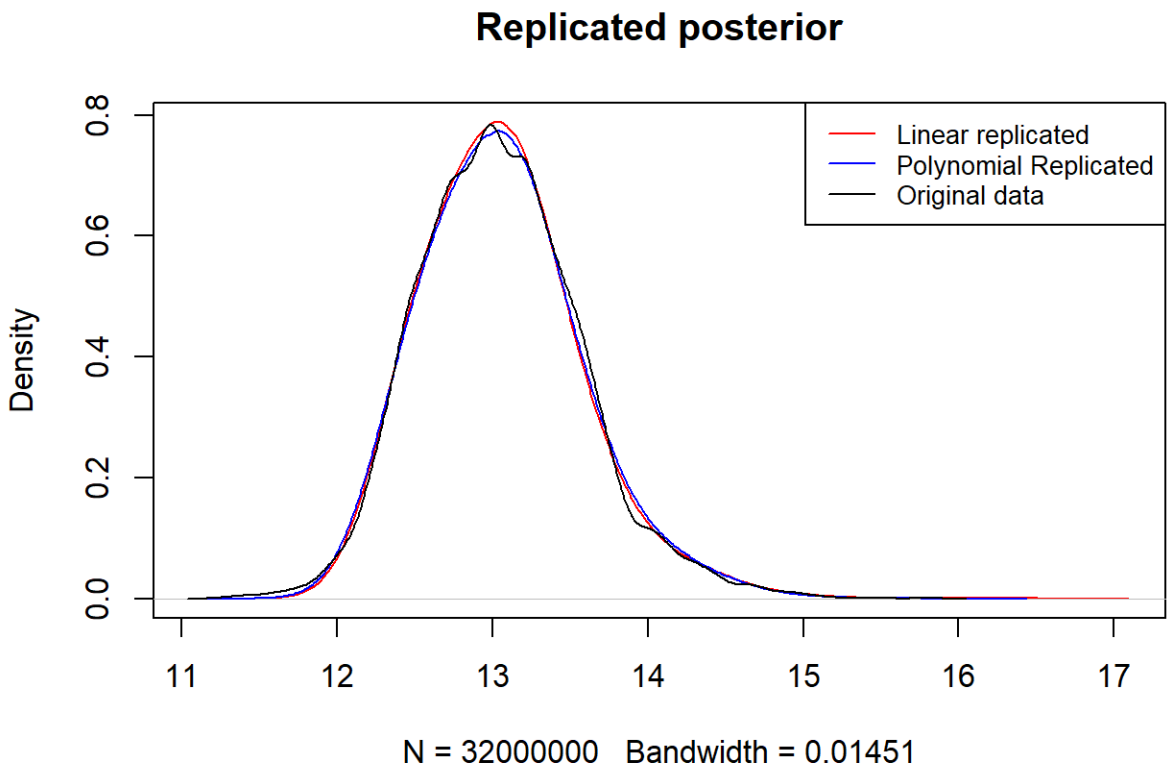

```
#past tree depth
get_num_max_treedepth(multiple_polynomial_fit)
```

```
## [1] 0
```

5. Posterior predictive checking

We can see that replicated data is indistinguishable from the target

```
replicated_data_lin = denormalize_results(extract(multiple_linear_fit)$y_rep, orig_sd, orig_mean)
replicated_data_pol = denormalize_results(extract(multiple_polynomial_fit)$y_rep, orig_sd, orig_mean)

plot(density(replicated_data_lin), col="red", main="Replicated posterior" )
lines(density(replicated_data_pol), col="blue")
lines(density(original_target), col="black")
legend(x="topright",
       legend=c("Linear replicated", "Polynomial Replicated", "Original data"),
       col=c("red", "blue", "black"), lty=1:1, cex=0.8)
```

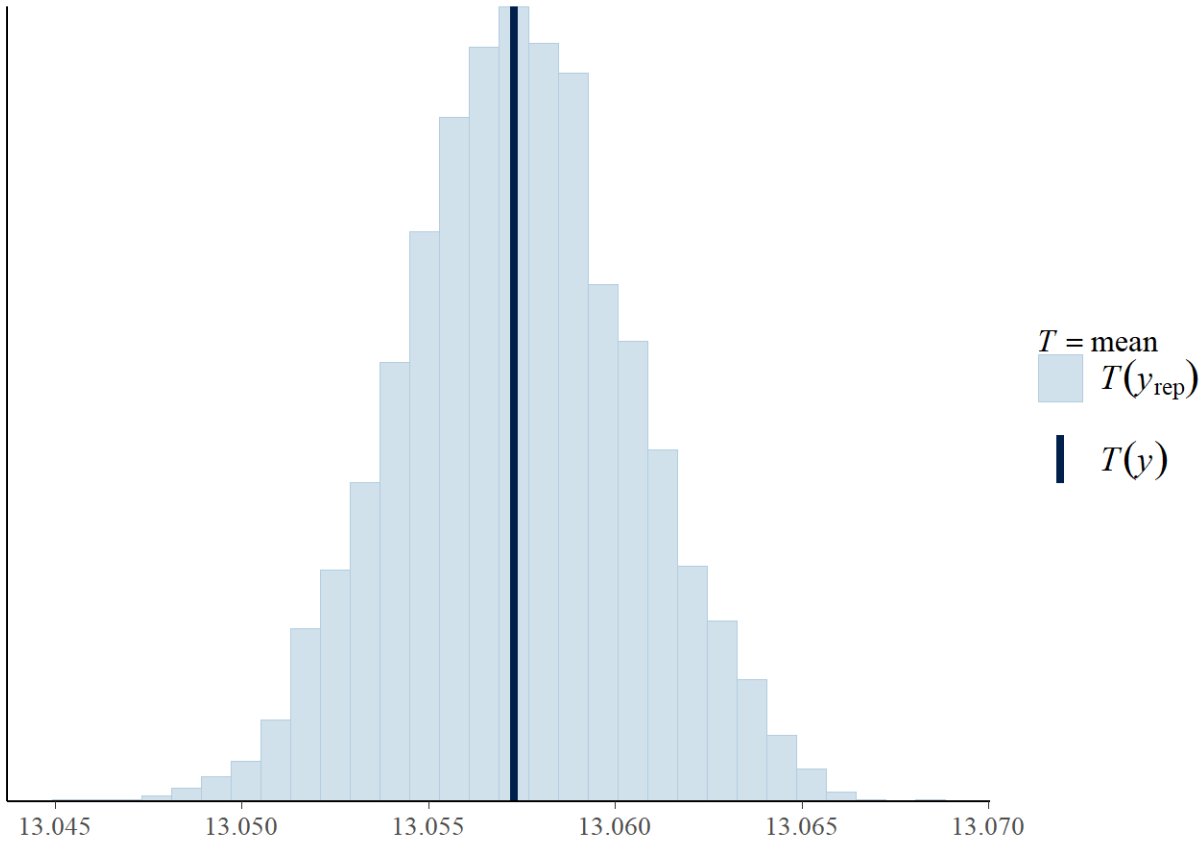



```
original_training_order = order(original_target[training_indices])
loo_lin <- loo(multiple_linear_fit, save_psis = TRUE)
```

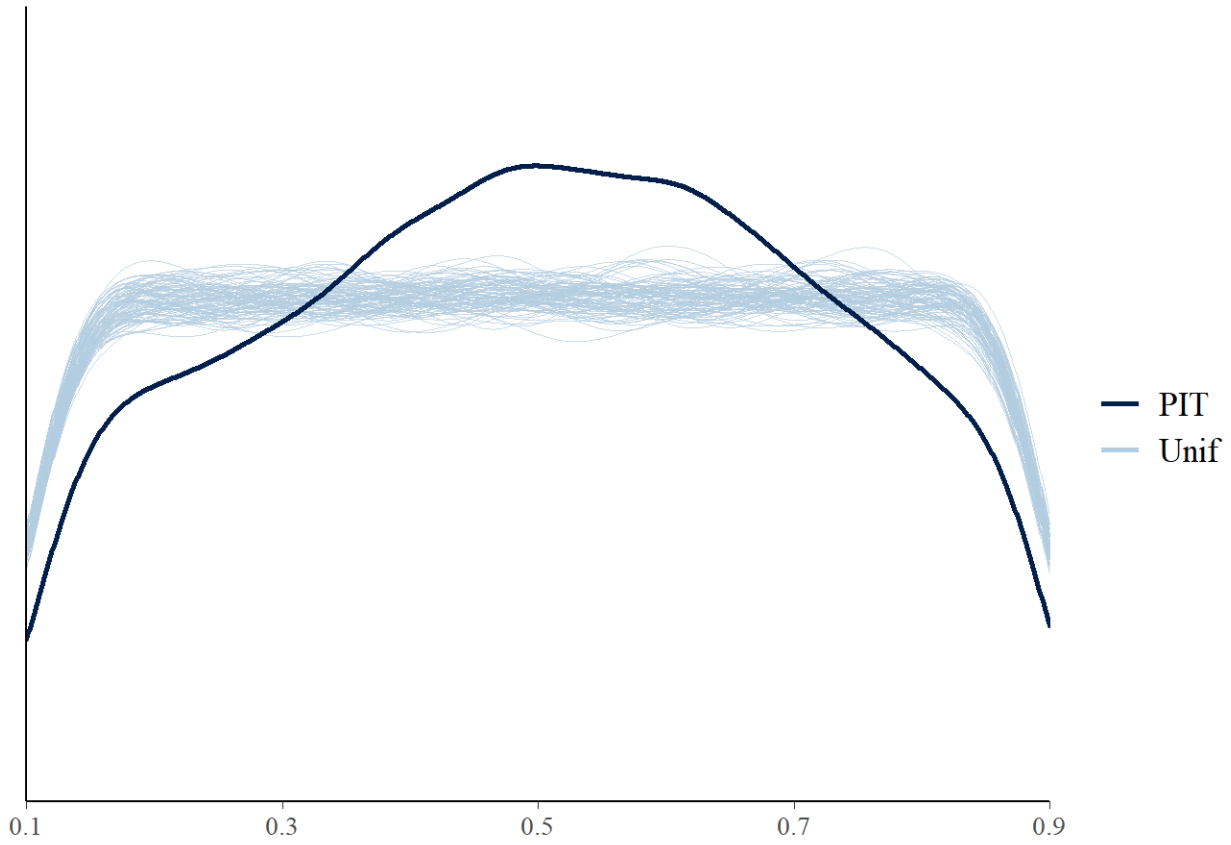
```
## Warning: Some Pareto k diagnostic values are slightly high. See help('pareto-k-diagnostic') for deta
```

```
psis_lin <- loo_lin$psis_object
lw_lin <- weights(psis_lin)
pp_check(c(original_target[training_indices]), yrep = replicated_data_lin, fun = "stat")
```

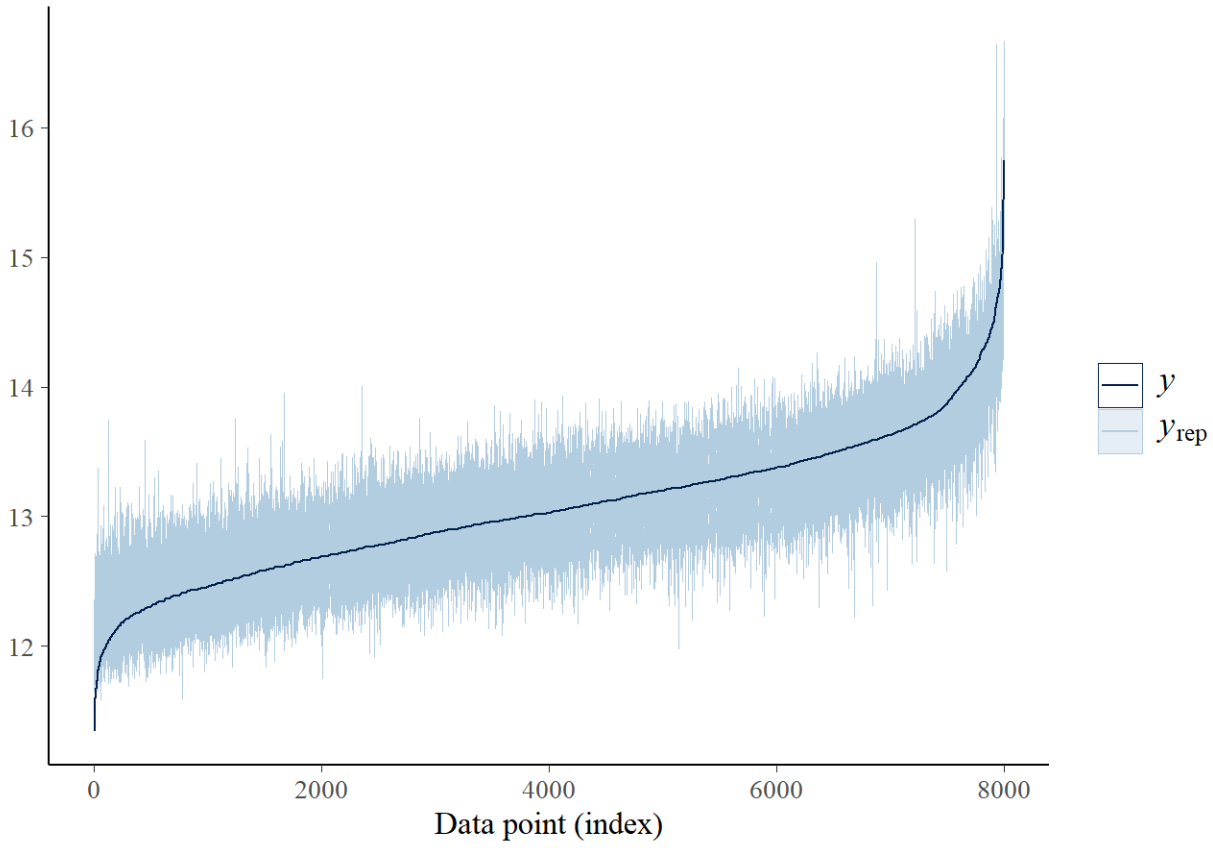
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



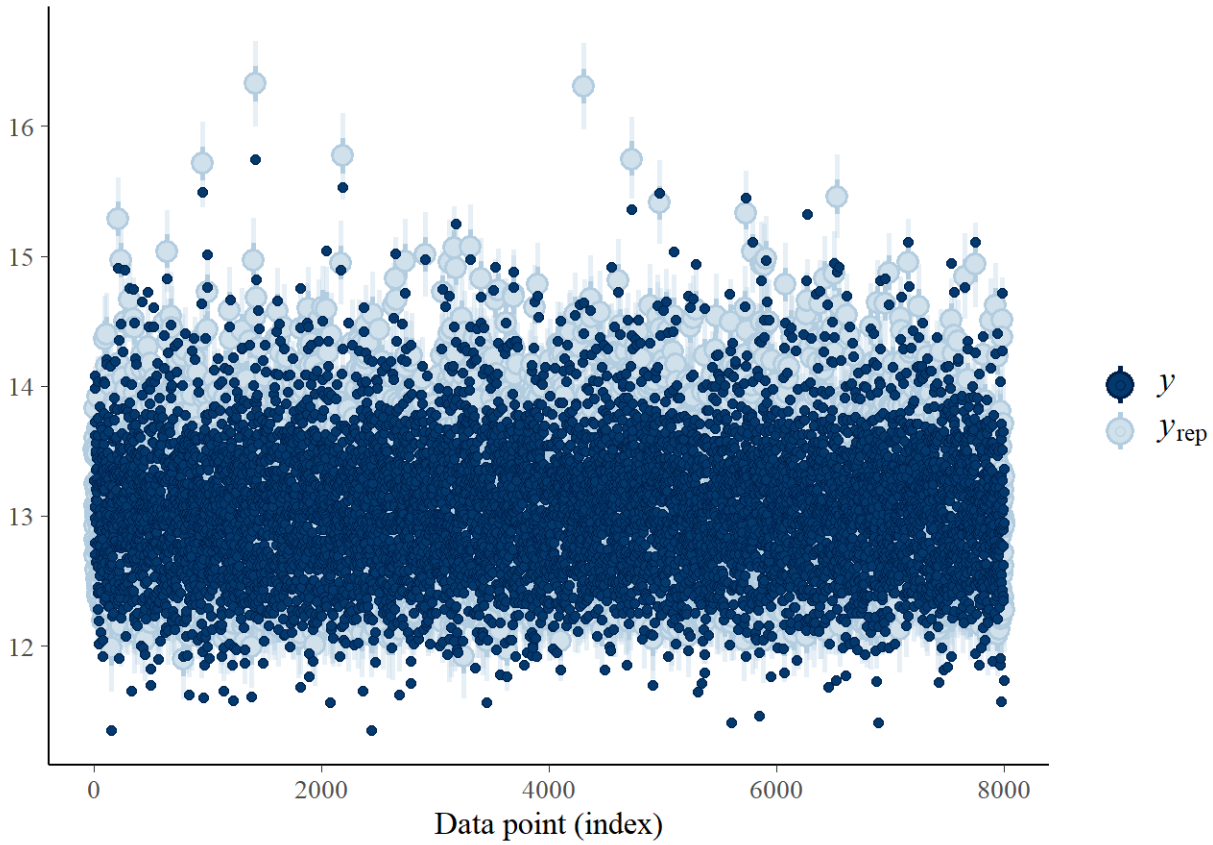

```
ppc_loo_pit_overlay(c(original_target[training_indices]), yrep = replicated_data_lin,
                    lw = lw_lin)
```




```
ppc_loo_ribbon(c(original_target[training_indices][original_training_order]),
               yrep = replicated_data_lin[,original_training_order],
               lw = lw_lin, psis_object = psis_lin)
```




```
ppc_loo_intervals(c(original_target[training_indices]),
                  yrep = replicated_data_lin, psis_object = psis_lin)
```

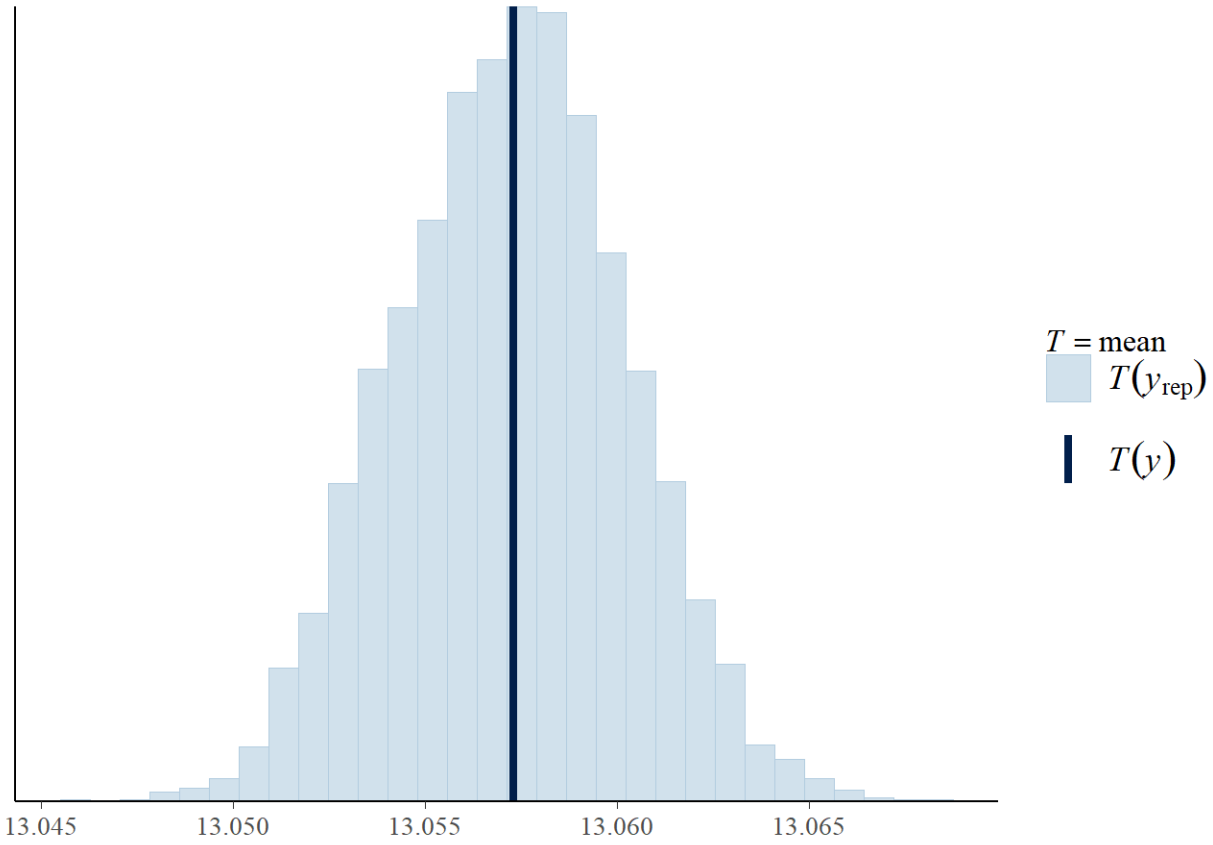



```
loo_pol <- loo(multiple_polynomial_fit, save_psis = TRUE)
```

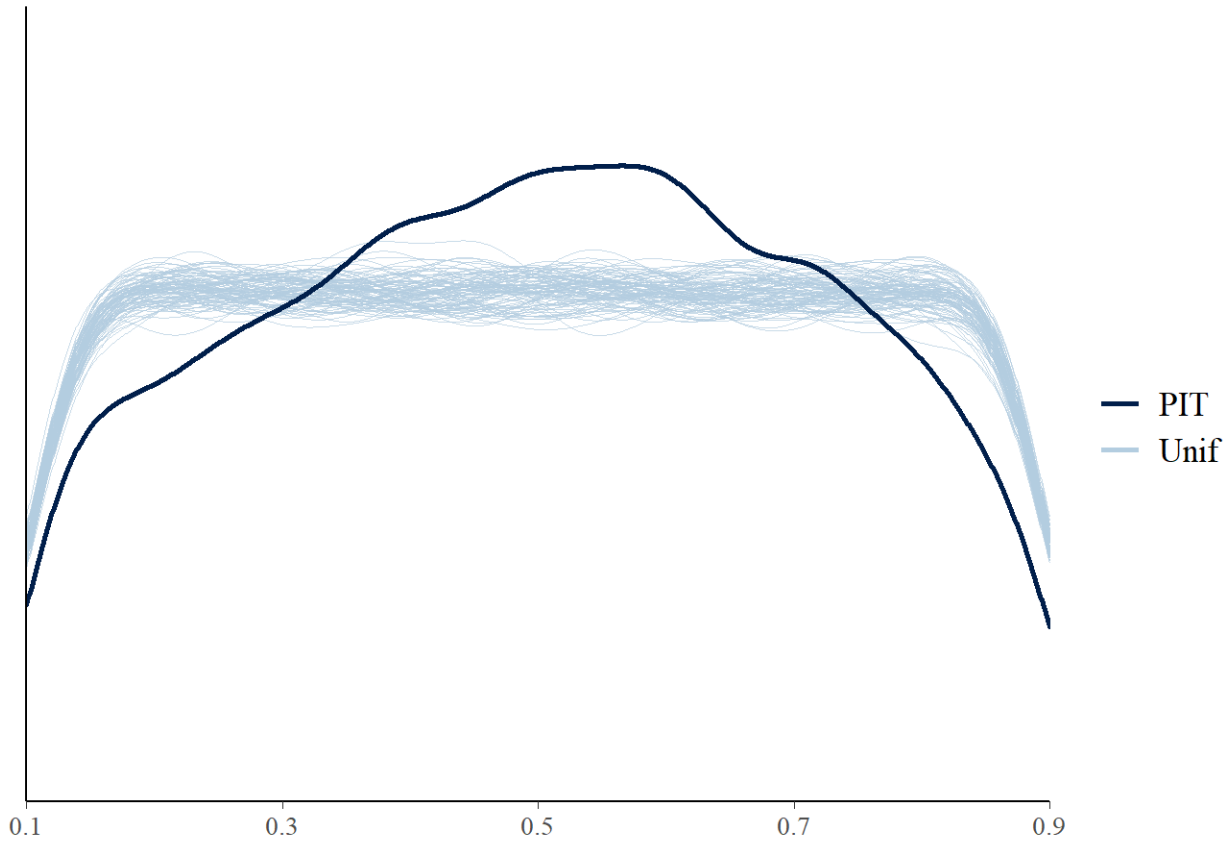
```
## Warning: Some Pareto k diagnostic values are slightly high. See help('pareto-k-diagnostic') for deta
```

```
psis_pol <- loo_pol$psis_object
lw_pol <- weights(psis_pol)
pp_check(c(original_target[training_indices]), yrep = replicated_data_pol, fun = "stat")
```

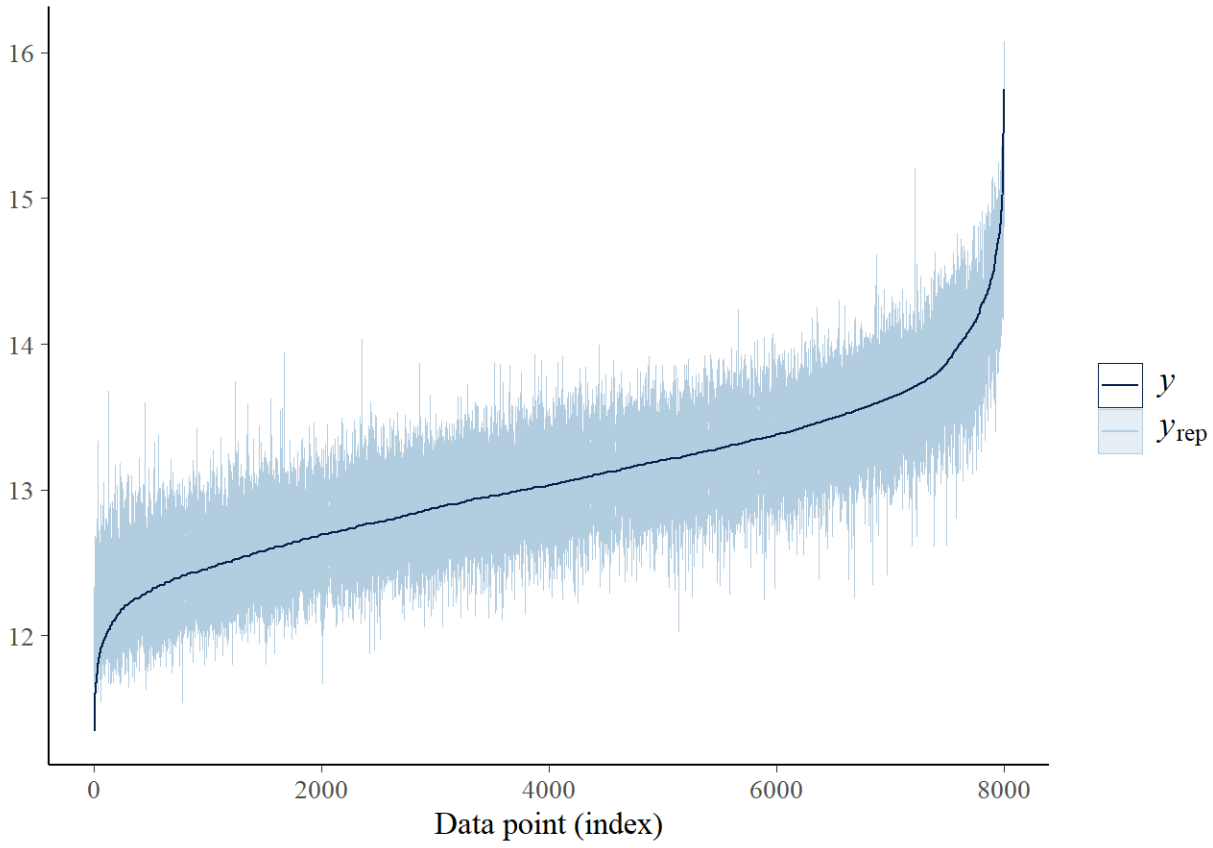
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



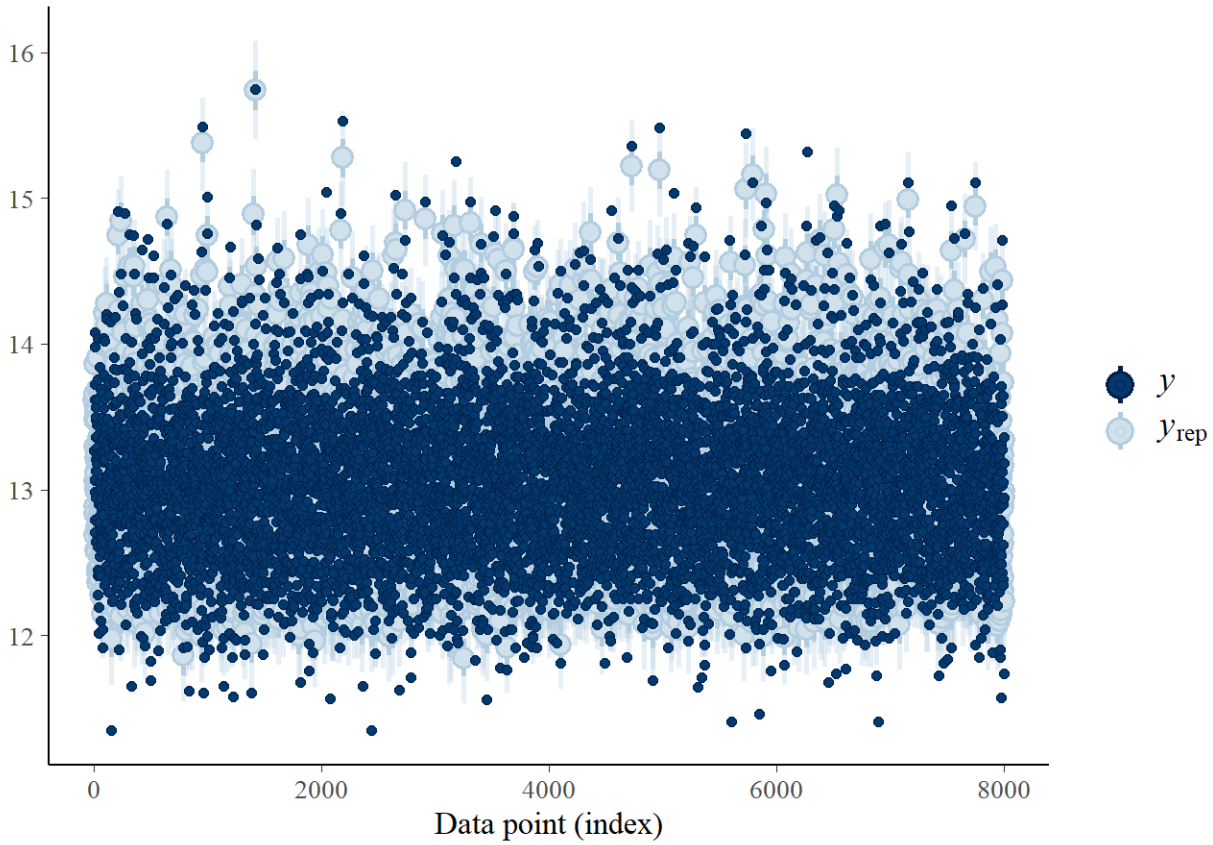

```
ppc_loo_pit_overlay(c(original_target[training_indices]), yrep = replicated_data_pol,
                    lw = lw_pol)
```




```
ppc_loo_ribbon(c(original_target[training_indices][original_training_order]),
               yrep = replicated_data_pol[,original_training_order],
               lw = lw_pol, psis_object = psis_pol)
```




```
ppc_loo_intervals(c(original_target[training_indices]),
                  yrep = replicated_data_pol, psis_object = psis_pol)
```



6. Predictive performance assesment

From the mean squared errors we can see that the polynomial model performed better on the test set

```
# compare errors
data.frame(linear = mae_lin, polynomial = mae_pol)
```

```
##     linear polynomial
## 1 150527.2   144281.3
```

PSIS-lOO

Obtained elpd information criteria values of the two models are largely the same with the polynomial model having an larger value, suggesting it is better of the two models. The k-values of the models are small suggesting the models fit the data well.

Multiple linear regression

```
# Extract log-likelihood
multiple_linear_log_lik <- extract_log_lik(multiple_linear_fit, merge_chains = FALSE)
```

```r
# PSIS-LOO elpd values
r_eff <- relative_eff(exp(multiple_linear_log_lik))
multiple_linear_loo_lin <- loo(multiple_linear_log_lik, r_eff = r_eff)
```

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

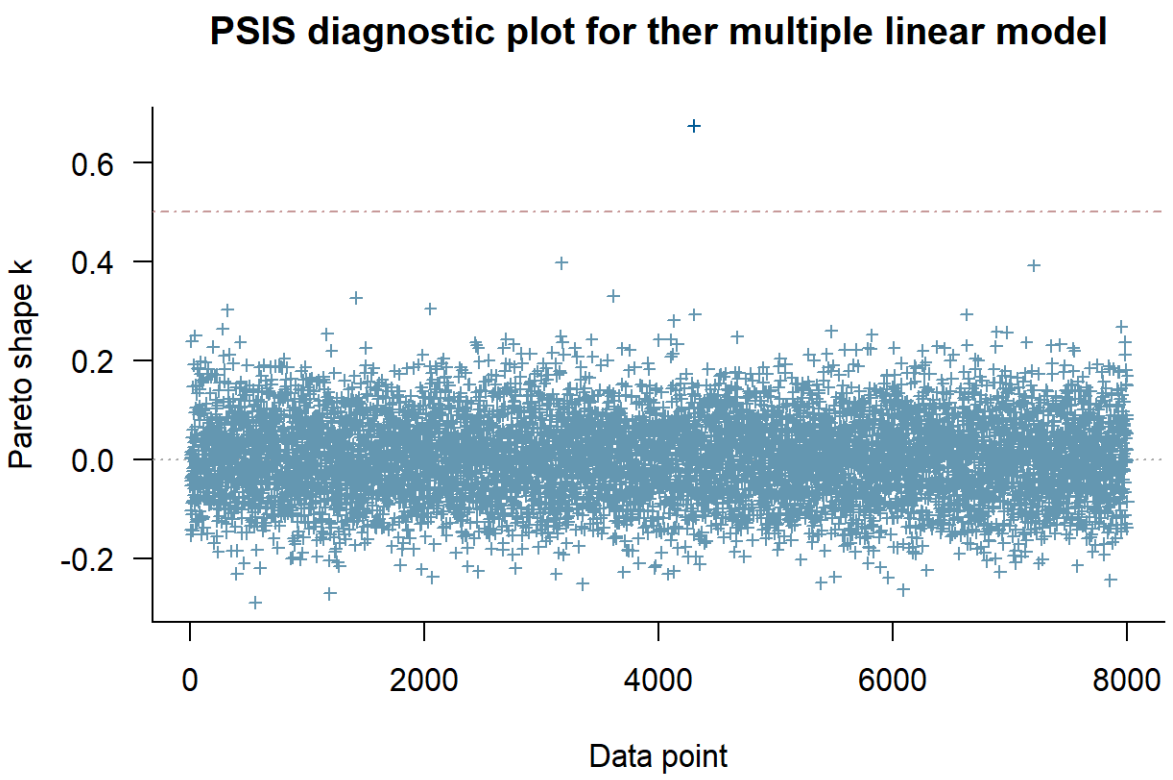
```r
#elpd loo
multiple_linear_loo_lin
```

```
##
## Computed from 4000 by 8000 log-likelihood matrix
##
##          Estimate    SE
## elpd_loo  -3427.6 105.3
## p_loo        89.9   4.4
## looic      6855.2 210.5
## ------
## Monte Carlo SE of elpd_loo is 0.2.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     7999  100.0%  210
##  (0.5, 0.7]   (ok)          1    0.0%  176
##    (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.
```

```r
pareto_k_table(multiple_linear_loo_lin)
```

```
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     7999  100.0%  210
##  (0.5, 0.7]   (ok)          1    0.0%  176
##    (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).
```

```r
plot(multiple_linear_loo_lin, diagnostic = c("k", "n_eff"), label_points = FALSE,
  main = "PSIS diagnostic plot for ther multiple linear model")
```



Multiple polynomial regression

```
# Extract log-likelihood
multiple_polynomial_log_lik <- extract_log_lik(multiple_polynomial_fit, merge_chains = FALSE)

# PSIS-LOO elpd values
r_eff <- relative_eff(exp(multiple_polynomial_log_lik))
multiple_polynomial_loo_lin <- loo(multiple_polynomial_log_lik, r_eff = r_eff)
```

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

```
#elpd loo
multiple_polynomial_loo_lin
```

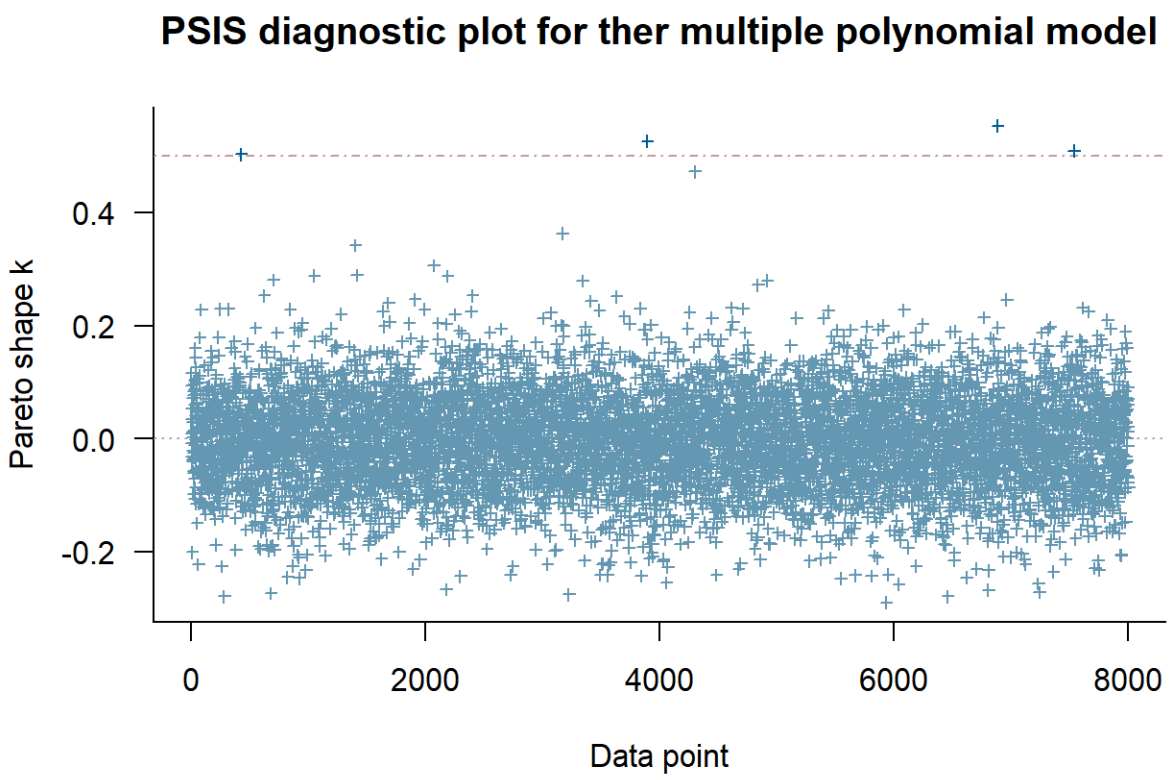
```
##
## Computed from 4000 by 8000 log-likelihood matrix
##
##          Estimate    SE
## elpd_loo  -3205.9  97.3
## p_loo        98.6   3.5
## looic      6411.8 194.5
## ------
## Monte Carlo SE of elpd_loo is 0.2.
```

```
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     7996  100.0%  591
##  (0.5, 0.7]   (ok)          4    0.0%  362
##    (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.
```

```
pareto_k_table(multiple_polynomial_loo_lin)
```

```
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     7996  100.0%  591
##  (0.5, 0.7]   (ok)          4    0.0%  362
##    (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).
```

```
plot(multiple_polynomial_loo_lin, diagnostic = c("k", "n_eff"), label_points = FALSE,
  main = "PSIS diagnostic plot for ther multiple polynomial model")
```



p_eff values

```
loo_compare(x = list(multiple_linear_loo_lin, multiple_polynomial_loo_lin))
```

```
##        elpd_diff se_diff
## model2    0.0       0.0
## model1 -221.7      45.7
```

7. Discussion

In this report we have explored linear and polynomial regression models for predicting house prices. The differences between the results from the models are small, but the polynomial model performs a bit better. The mean absolute error for both models is over hundred thousand, but considering the mean of the prices is around five hundred thousand, some error is to be expected. Overall the results follow the true data.

In the future we could consider varying slope parameter by zipcode, but this has few obvious drawbacks. There are 70 groups, so using a different beta value for each parameter for each group would increase the number of parameters of the model by 2-17 times, likely slowing the model. In addition, the number of data usable for each beta value would shrink. The dataset is large, so using most of the dataset for training, it shouldn't be a problem but with less data it could lead to overfitting. Practically it would mean that there is no relation between the effects of the parameters between different groups, e.g. the size of the building could increase price somewhere and decrease it elsewhere, which sounds counterintuitive, but could still be an avenue for future research.