analysis

December 16, 2020

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pystan
import arviz as az
from pathlib import Path
from matplotlib.patches import Patch
from matplotlib.lines import Line2D

verbose=False
```

1 2. Some Elementary statistics

[2]:
```
model_path = '../Stan'
data_file = '../Data/data.txt'
accident_data = np.loadtxt(data_file)
print(accident_data.shape)
```

```
(6, 15)
```

[3]:
```
# # uk population from 2005 to 2019
# total_population=np.array([60413300,60827100,61319100,61823800,62260500,
#                   62759500,63285100,63705000,64105700,64596800,
#                   65110000,65648100,66040200,66435600,66796800])
# population_factor = total_population/np.max(total_population)
# population_factor
```

[4]:
```
# accident_data = accident_data / population_factor.reshape(-1,␣
↪total_population.size)
```

[3]:
```
# mean value approximately 25 cases per 10,000 people
mean_value = np.mean(accident_data)
mean_value
```

[3]:
```
24.953333333333337
```

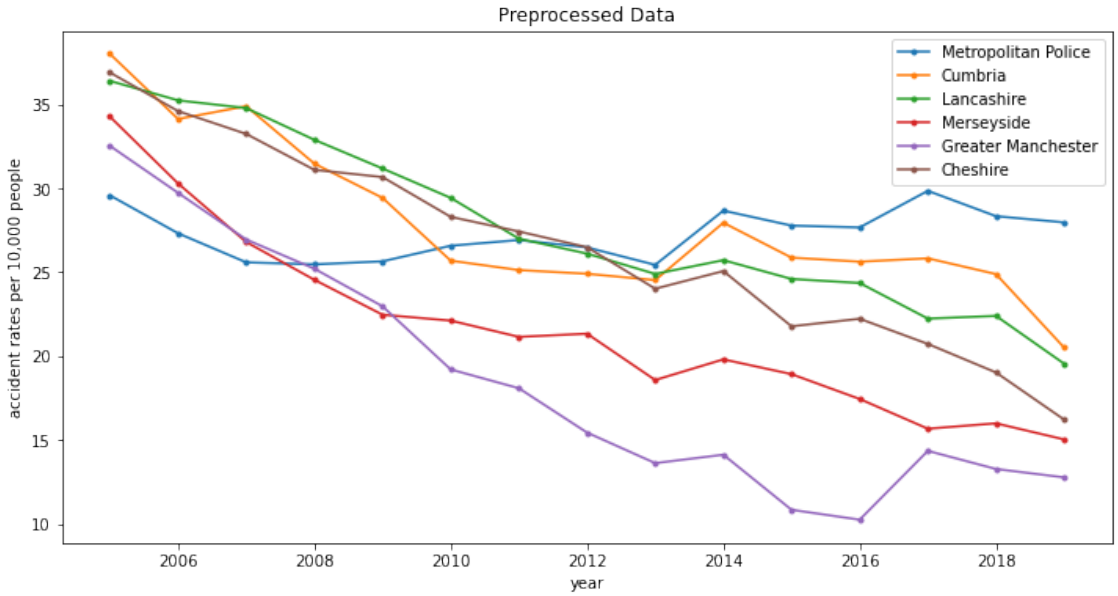
[4]:
```
area_names = ["Metropolitan Police", 'Cumbria','Lancashire',
              'Merseyside','Greater Manchester','Cheshire']
years = np.arange(2005, 2020, 1).astype(np.int)
df = pd.DataFrame(accident_data, index=area_names, columns=years)
df
# print(df.to_markdown())
```

[4]:

	2005	2006	2007	2008	2009	2010	2011	2012	\
Metropolitan Police	29.57	27.31	25.58	25.46	25.64	26.57	26.91	26.47	
Cumbria	38.00	34.12	34.88	31.46	29.42	25.68	25.12	24.90	
Lancashire	36.39	35.23	34.77	32.88	31.17	29.42	26.99	26.10	
Merseyside	34.26	30.28	26.79	24.54	22.46	22.11	21.14	21.33	
Greater Manchester	32.52	29.72	26.94	25.20	22.96	19.20	18.08	15.44	
Cheshire	36.89	34.59	33.23	31.09	30.66	28.29	27.42	26.48	

	2013	2014	2015	2016	2017	2018	2019
Metropolitan Police	25.43	28.66	27.77	27.66	29.84	28.33	27.96
Cumbria	24.52	27.94	25.86	25.62	25.82	24.88	20.50
Lancashire	24.89	25.71	24.59	24.35	22.23	22.39	19.54
Merseyside	18.57	19.80	18.92	17.44	15.67	15.99	15.03
Greater Manchester	13.62	14.12	10.84	10.25	14.35	13.27	12.77
Cheshire	24.01	25.06	21.77	22.22	20.72	19.02	16.21

[5]:
```
plt.figure(figsize=(12, 6))
for i in range(6):
    plt.plot(years,accident_data[i],marker='.', label=area_names[i])
plt.legend()
plt.title('Preprocessed Data')
plt.xlabel('year')
plt.ylabel('accident rates per 10,000 people')
plt.show()
```

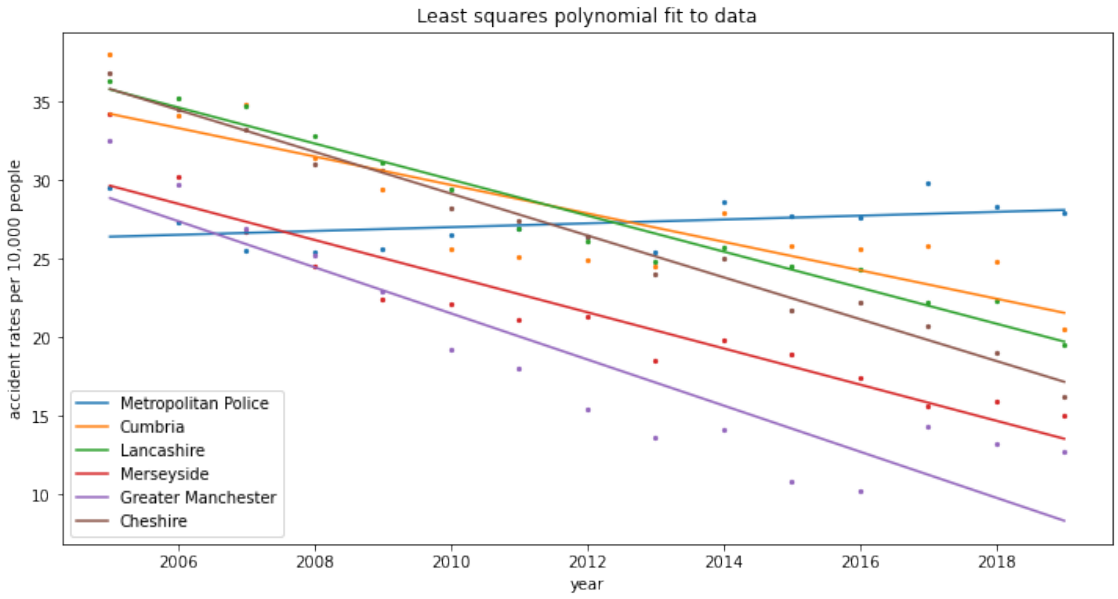


[6]:
```
print(df.T.describe().round(2).to_markdown())
```

```
|       |   Metropolitan Police |   Cumbria |   Lancashire |   Merseyside |
Greater Manchester |   Cheshire |
|:------|----------------------:|----------:|-------------:|-------------:|-----
---------------:|-----------:|
| count |                    15 |        15 |           15 |           15 |
15    |         15 |
| mean  |                 27.28 |     27.91 |        27.78 |        21.62 |
18.62 |      26.51 |
| std   |                  1.45 |      4.74 |         5.24 |         5.47 |
7.15 |       6.01 |
| min   |                 25.43 |     20.5  |        19.54 |        15.03 |
10.25 |      16.21 |
| 25%   |                 26.06 |     25.01 |        24.47 |        18.01 |
13.44 |      21.99 |
| 50%   |                 27.31 |     25.82 |        26.1  |        21.14 |
15.44 |      26.48 |
| 75%   |                 28.14 |     30.44 |        32.03 |        23.5  |
24.08 |      30.88 |
| max   |                 29.84 |     38    |        36.39 |        34.26 |
32.52 |      36.89 |
```

[7]:
```
plt.figure(figsize=(12, 6))
for i in range(6):
    plt.scatter(years, accident_data[i, :], marker='.', s=20)
    fit = np.polyfit(years, accident_data[i, :], 1)
```

```
    fitted_values = np.polyval(fit, years)
    plt.plot(years, fitted_values, label=area_names[i])
plt.legend()
plt.title('Least squares polynomial fit to data')
plt.xlabel('year')
plt.ylabel('accident rates per 10,000 people')
plt.show()
```



2 3. Probability Models

2.1 3.1 Separate Model

In a separate model, we treat each district as an individual entity, and assign independent parameters to them. Specifically, we assign individual parameters α_i and β_i to the ith area, and make the mean vary linearly with respect to years. But each district will have a constant variance across all 15 years. The mathematical expression for the separate model can be specified with the following equations:

$$\alpha_i \sim Normal(30, 20)$$
$$\beta_i \sim Normal(0, 4.85)$$
$$\sigma_j \sim uniform$$
$$\mu_{i,j} = \alpha_i + \beta_i * year[j]$$
$$accident[i, j] \sim Normal(\mu_{i,j}, \sigma_j)$$

[10]:
```
separate_model_name = 'accident_separate.stan'
separate_stan_model = pystan.StanModel(file=model_path + '/' +␣
 ↪separate_model_name)
print(separate_stan_model.model_code)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL
anon_model_7118a747b64b48d22305b3d729749de8 NOW.
```

```
//
// This Stan program defines a simple model, with a
// vector of values 'y' modeled as normally distributed
// with mean 'mu' and standard deviation 'sigma'.
//
// Learn more about model development with Stan at:
//
//    http://mc-stan.org/users/interfaces/rstan.html
//    https://github.com/stan-dev/rstan/wiki/RStan-Getting-Started
//

// The input data is a vector 'y' of length 'N'.
data {
  int<lower=0> N; // the number of police force
  int<lower=0> Y; // the number of years has been studied, year 2005 corresponds
to 1
  matrix[N,Y] accidentData;//accident data
  int prior_choice; // choose different setup for prior distribution
  int xpred; // year of prediction (actual year)
}

// The parameters accepted by the model. Our model
// accepts two parameters 'mu' and 'sigma'.
parameters {
  vector[N] alpha;
  vector[N] beta;
  vector<lower=0>[N] sigma;
}

transformed parameters{
  matrix[N,Y]mu;
  for(i in 1:N)
    for(j in 1:Y)
      mu[i,j]=alpha[i]+beta[i]*j;
}

// The model to be estimated. We model the output
// 'y' to be normally distributed with mean 'mu'
// and standard deviation 'sigma'.
model {
```

```
  // loop over police offices
  if (prior_choice==3){
    for(i in 1:N){
      alpha[i]~normal(0,100);
      beta[i]~normal(0,10);
    }
  } else if (prior_choice==2){
    // uniform prior
  } else {
    // default prior
    for(i in 1:N){
      alpha[i]~normal(30,20);
      beta[i]~normal(0,4.85);
    }
  }

  //for each police force
  for(i in 1:N){
    //for each observed year
    for(j in 1:Y){
     accidentData[i,j]~normal(mu[i,j],sigma[i]);
    }
  }
}


generated quantities{
  //log likelihood
  matrix[N,Y] log_lik;
  matrix[N,Y] yrep;
  //accident prediction in 2020 in different police force
  vector[N] pred;

  for(i in 1:N){
    // 2005 -> 1, 2006 -> 2, …, 2020 -> 16
    pred[i]=normal_rng(alpha[i]+beta[i]*(xpred-2004),sigma[i]);
  }

  for(i in 1:N){
    for(j in 1:Y){
      // do posterior sampling and try to reproduce the original data
      yrep[i,j]=normal_rng(mu[i,j],sigma[i]);
      // prepare log likelihood for PSIS-LOO
      log_lik[i,j]=normal_lpdf(accidentData[i,j]|mu[i,j],sigma[i]);
    }
  }

}
```

[8]:
```
def test_stan_model(stan_model, data, verbose = False, cov_coeff=0):
    data_for_stan = dict(
        N = data.shape[0],
        Y = data.shape[1],
        accidentData = data,
        years = np.arange(1, data.shape[1]+1), # stan index starts from 1
        xpred=2020,
        prior_choice=1,
        cov_coeff=0,
    )
    stan_results = stan_model.sampling(data=data_for_stan)
    if verbose:
        print(stan_results)
    else:
        print(stan_results.stansummary(pars=["alpha", "beta", "sigma"]))
    return stan_results
```

[ ]:
```
```

[12]:
```
separate_results = test_stan_model(separate_stan_model, accident_data,␣
↪verbose=verbose)
```

```
Inference for Stan model: anon_model_7118a747b64b48d22305b3d729749de8.
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]  26.31    0.02   0.87   24.6  25.77  26.31  26.85  28.05   2155    1.0
alpha[2]  35.03    0.03   1.57  31.79  34.01  35.06  36.03  38.12   2820    1.0
alpha[3]  36.91    0.01   0.67  35.56  36.48  36.91  37.35  38.23   2761    1.0
alpha[4]  30.81    0.02   1.16  28.51  30.03   30.8  31.54  33.13   2868    1.0
alpha[5]  30.33    0.03    1.8  26.66  29.19  30.32   31.5  33.85   2810    1.0
alpha[6]  37.16    0.01   0.53  36.12  36.82  37.16  37.49  38.24   2631    1.0
beta[1]    0.12  2.0e-3    0.1  -0.07   0.06   0.12   0.18    0.3   2185    1.0
beta[2]   -0.89  3.2e-3   0.17  -1.22   -1.0  -0.89  -0.78  -0.54   2895    1.0
beta[3]   -1.14  1.4e-3   0.07  -1.29  -1.19  -1.14   -1.1  -0.99   2574    1.0
beta[4]   -1.15  2.3e-3   0.13  -1.41  -1.23  -1.15  -1.06   -0.9   2979    1.0
beta[5]   -1.46  3.8e-3    0.2  -1.86  -1.59  -1.46  -1.33  -1.06   2853    1.0
beta[6]   -1.33  1.1e-3   0.06  -1.45  -1.37  -1.33   -1.3  -1.22   2693    1.0
sigma[1]   1.56  6.4e-3   0.37   1.03    1.3   1.49   1.75   2.46   3256    1.0
sigma[2]   2.86    0.01   0.66   1.92    2.4   2.76   3.19   4.47   2682    1.0
sigma[3]   1.25  5.2e-3   0.29   0.83   1.04    1.2   1.39   1.93   3084    1.0
sigma[4]   2.14  8.0e-3   0.46   1.46   1.81   2.07   2.39   3.22   3362    1.0
sigma[5]   3.28    0.01   0.74   2.21   2.77   3.16   3.66   5.05   3299    1.0
```

```
sigma[6]    0.93  3.8e-3    0.21    0.62    0.79     0.9    1.04    1.44    3009     1.0

Samples were drawn using NUTS at Mon Dec 14 15:57:49 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).
```

2.2 3.2 Pooled Model

$$\alpha \sim Normal(30, 20)$$
$$\beta \sim Normal(0, 4.85)$$
$$\sigma_j \sim uniform$$
$$\mu_j = \alpha + \beta * year[j]$$
$$accident[:, j] \sim Normal(\mu_j, \sigma_j)$$

[13]:
```
pooled_model_name = 'accident_pooled.stan'
pooled_stan_model = pystan.StanModel(file=model_path + '/' + pooled_model_name)
print(pooled_stan_model.model_code)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL
anon_model_78401062c54be5038724f93ddb7d812c NOW.
```

```
//
// This Stan program defines a simple model, with a
// vector of values 'y' modeled as normally distributed
// with mean 'mu' and standard deviation 'sigma'.
//
// Learn more about model development with Stan at:
//
//    http://mc-stan.org/users/interfaces/rstan.html
//    https://github.com/stan-dev/rstan/wiki/RStan-Getting-Started
//

data {
  int<lower=0> N; // the number of police force
  int<lower=0> Y; // the number of years has been studied, year 2005 corresponds
to 1
  matrix[N,Y] accidentData;//accident data
  int prior_choice; // choose different setup for prior distribution
  int xpred; // year of prediction (actual year)
}



parameters {
  real alpha;
  real beta;
```

```
  real<lower=0> sigma;
}

transformed parameters{
  vector[Y]mu;
  //linear model
  for(j in 1:Y)
    mu[j]=alpha+beta*j;
}


model {
  //prior
  if (prior_choice==3){
    // weaker prior
    alpha~normal(0,100);
    beta~normal(0,10);
  } else if (prior_choice==2) {
    // uniform prior
  } else {
    // default prior
    alpha~normal(30,20);
    beta~normal(0,4.85);
  }

  //for each year, different police force share the same model
  for(j in 1:Y){
    accidentData[,j]~normal(mu[j],sigma);
  }
}

generated quantities{
  //log likelihood
  matrix[N,Y] log_lik;
  matrix[N,Y] yrep;
  //accident prediction in 2020 in different police force
  vector[N] pred;
  for(i in 1:N){
    // 2005 -> 1, 2006 -> 2, …, 2020 -> 16
    pred[i]=normal_rng(alpha+beta*(xpred-2004),sigma);
  }

  for(i in 1:N){
    for(j in 1:Y){
      // do posterior sampling and try to reproduce the original data
      yrep[i,j]=normal_rng(mu[j],sigma);
      // prepare log likelihood for PSIS-LOO
      log_lik[i,j]=normal_lpdf(accidentData[i,j]|mu[j],sigma);
```

```
    }
  }

}
```

[14]:
```
pooled_results = test_stan_model(pooled_stan_model, accident_data,␣
↪verbose=verbose)
```

```
Inference for Stan model: anon_model_78401062c54be5038724f93ddb7d812c.
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   32.8    0.03   1.03  30.77   32.1   32.8  33.46  34.88   1649    1.0
beta   -0.98  2.8e-3   0.11   -1.2  -1.05  -0.98  -0.91  -0.76   1588    1.0
sigma   4.72  8.1e-3   0.37   4.06   4.46   4.68   4.94   5.53   2084    1.0

Samples were drawn using NUTS at Mon Dec 14 15:58:33 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).
```

2.3 3.3 Hierarchical Model

[9]:
```
hier_model_name = 'accident_hierarchical.stan'
hier_stan_model = pystan.StanModel(file=model_path + '/' + hier_model_name)
print(hier_stan_model.model_code)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL
anon_model_40aecd51cc828896ccffcb762a178e52 NOW.
```

```
//
// This Stan program defines a simple model, with a
// vector of values 'y' modeled as normally distributed
// with mean 'mu' and standard deviation 'sigma'.
//
// Learn more about model development with Stan at:
//
//    http://mc-stan.org/users/interfaces/rstan.html
//    https://github.com/stan-dev/rstan/wiki/RStan-Getting-Started
//

data {
  int<lower=0> N; // the number of police force
  int<lower=0> Y; // the number of years has been studied, year 2005 corresponds
```

```
to 1
  matrix[N,Y] accidentData;//accident data
  int prior_choice; // choose different setup for prior distribution
  int xpred; // year of prediction (actual year)
}


parameters {
  real mu_alpha;
  real mu_beta;
  real<lower=0> sigma_alpha;
  real<lower=0> sigma_beta;
  vector[N] alpha;
  vector[N] beta;
  // vector<lower=0>[N] sigma;
  real<lower=0> sigma;
}


transformed parameters{
  matrix[N,Y]mu;
  for(i in 1:N)
    for(j in 1:Y)
      mu[i,j]=alpha[i]+beta[i]*j;
}


model {
  if (prior_choice==3){
    // bigger variance
    mu_alpha~normal(30,40);
    mu_beta~normal(0,10);
    //sigma_alpha~normal(10,10);
    //sigma_beta~normal(3,6);
  } else if (prior_choice==2){
    // uniform prior
  } else {
    // default choice with moderate variance
    mu_alpha~normal(30,20);
    mu_beta~normal(0,4.85);
    //sigma_alpha~normal(10,5);
    //sigma_beta~normal(3,3);
  }

  //for each police force
  for(i in 1:N){
    alpha[i]~normal(mu_alpha,sigma_alpha);
    beta[i]~normal(mu_beta,sigma_beta);
```

```
  }

  //for each police force
  for(i in 1:N){
    //for each observed year
    for(j in 1:Y){
     // accidentData[i,j]~normal(mu[i,j],sigma[i]);
     accidentData[i,j]~normal(mu[i,j],sigma); // share sigma
    }
  }
}


generated quantities{
  //log likelihood
  matrix[N,Y] log_lik;
  matrix[N,Y] yrep;
  //accident prediction in 2020 in different police force
  vector[N] pred;

  //for each police force
  for(i in 1:N){
    // 2005 -> 1, 2006 -> 2, …, 2020 -> 16
    // pred[i]=normal_rng(alpha[i]+beta[i]*(xpred-2004),sigma[i]);
    // share sigma
    pred[i]=normal_rng(alpha[i]+beta[i]*(xpred-2004),sigma);
  }

  for(i in 1:N){
    for(j in 1:Y){
      // do posterior sampling and try to reproduce the original data
      // yrep[i,j]=normal_rng(mu[i,j],sigma[i]);
      yrep[i,j]=normal_rng(mu[i,j],sigma);
      // prepare log likelihood for PSIS-LOO
      // log_lik[i,j]=normal_lpdf(accidentData[i,j]|mu[i,j],sigma[i]);
      // share sigma
      log_lik[i,j]=normal_lpdf(accidentData[i,j]|mu[i,j],sigma);
    }
  }

}
```

[10]:
```
hier_results = test_stan_model(hier_stan_model, accident_data, verbose=verbose)
```

```
Inference for Stan model: anon_model_40aecd51cc828896ccffcb762a178e52.
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]  26.92    0.02   1.09  24.79  26.17  26.92  27.65  29.07   3008    1.0
alpha[2]  35.05    0.02   1.03  33.08  34.36  35.05  35.73   37.1   3187    1.0
alpha[3]  36.65    0.02   1.07  34.48  35.95  36.65  37.35  38.75   3344    1.0
alpha[4]  30.88    0.02   1.05  28.85  30.18  30.87  31.58   33.0   3407    1.0
alpha[5]  30.34    0.02   1.05  28.23  29.64  30.33  31.05  32.43   3156    1.0
alpha[6]  36.85    0.02    1.1  34.63  36.13  36.86  37.58  38.96   3637    1.0
beta[1]    0.05  2.1e-3   0.12  -0.18  -0.03   0.05   0.13   0.29   3218    1.0
beta[2]   -0.89  2.0e-3   0.11  -1.12  -0.97  -0.89  -0.82  -0.68   3195    1.0
beta[3]   -1.12  2.0e-3   0.12  -1.35  -1.19  -1.12  -1.04  -0.89   3395    1.0
beta[4]   -1.15  2.0e-3   0.11  -1.38  -1.23  -1.15  -1.08  -0.93   3363    1.0
beta[5]   -1.46  2.1e-3   0.11  -1.69  -1.54  -1.46  -1.38  -1.24   2995    1.0
beta[6]    -1.3  2.0e-3   0.12  -1.54  -1.38   -1.3  -1.22  -1.06   3540    1.0
sigma      1.99  2.3e-3   0.16    1.7   1.88   1.98   2.09   2.33   5084    1.0

Samples were drawn using NUTS at Wed Dec 16 14:30:18 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).
```

[20]:
```
beta = hier_results.extract()['beta']
print(beta.shape)
change_in_GM = np.quantile(beta[:,4], [0.05, 0.95])
# decay of cases in great manchester
print(2835686*change_in_GM/10000)
```

```
(4000, 6)
[-467.38884453 -361.55504165]
```

[23]:
```
pred = hier_results.extract()['pred']
print(pred.shape)
cases_London = np.quantile(pred[:,0], [0.05, 0.95])
print(cases_London*8961989/10000)
```

```
(4000, 6)
[21455.08090866 28120.7658366 ]
```

3 4. Model Evaluation

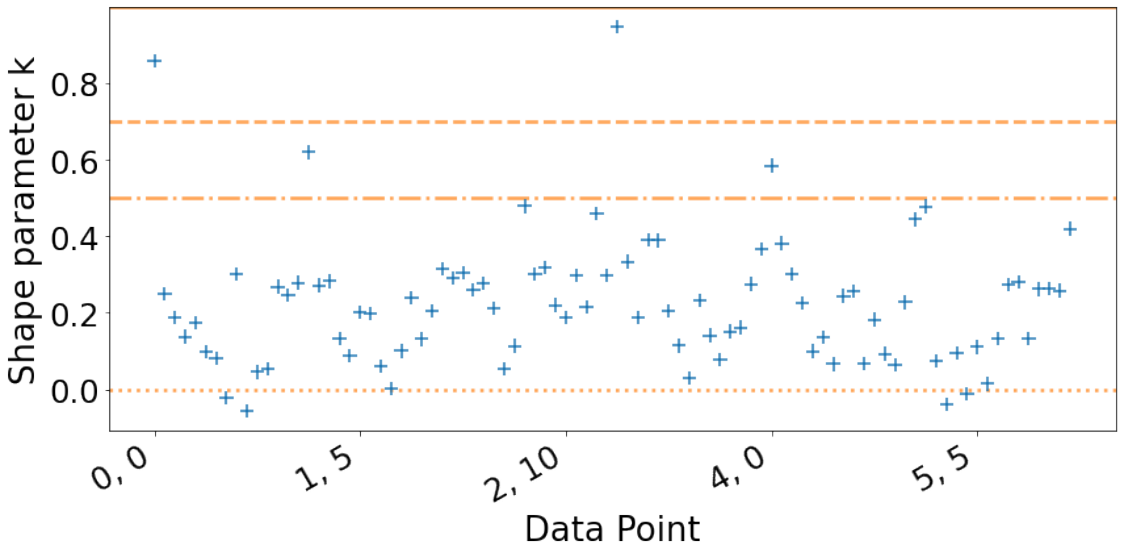
[17]:
```
var_separate =["alpha", "beta", "sigma"] # the variables that need to be plotted
var_pooled = ["alpha", "beta", "sigma"]
var_hier = ["alpha", "beta", "sigma"]
```

3.1 4.1 Cross-Validation with PSIS-LOO

[18]:
```
def get_psis_loo_result(stan_results):
    idata = az.from_pystan(stan_results, log_likelihood="log_lik")
    loo_results = az.loo(idata, pointwise=True)
    print(loo_results)
    khats = loo_results.pareto_k
    az.plot_khat(khats, xlabels=True, annotate=True, figsize=(12, 6))
    plt.show()
```

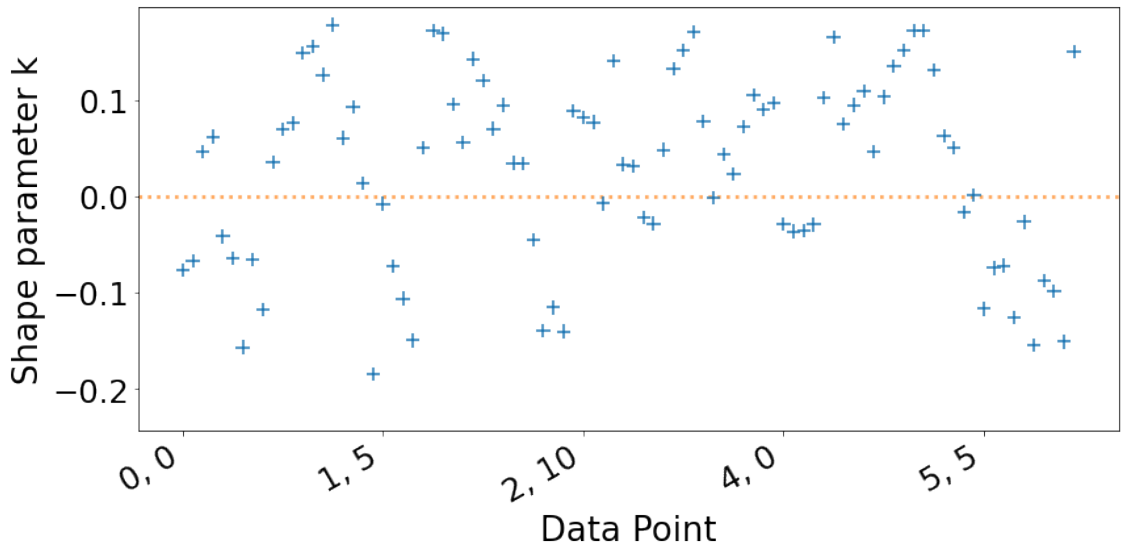
[19]:
```
get_psis_loo_result(separate_results)
```

```
Computed from 4000 by 90 log-likelihood matrix

         Estimate       SE
elpd_loo  -186.51     7.71
p_loo       16.26        -

There has been a warning during the calculation. Please check the results.
------

Pareto k diagnostic values:
                         Count   Pct.
(-Inf, 0.5]   (good)       86   95.6%
 (0.5, 0.7]   (ok)          2    2.2%
   (0.7, 1]   (bad)         2    2.2%
   (1, Inf)   (very bad)    0    0.0%

/home/weijiang/anaconda3/envs/bda/lib/python3.7/site-
packages/arviz/stats/stats.py:684: UserWarning: Estimated shape parameter of
Pareto distribution is greater than 0.7 for one or more samples. You should
consider using a more robust model, this is because importance sampling is less
likely to work well if the marginal posterior and LOO posterior are very
different. This is more likely to happen with a non-robust model and highly
influential observations.
  "Estimated shape parameter of Pareto distribution is greater than 0.7 for "
```



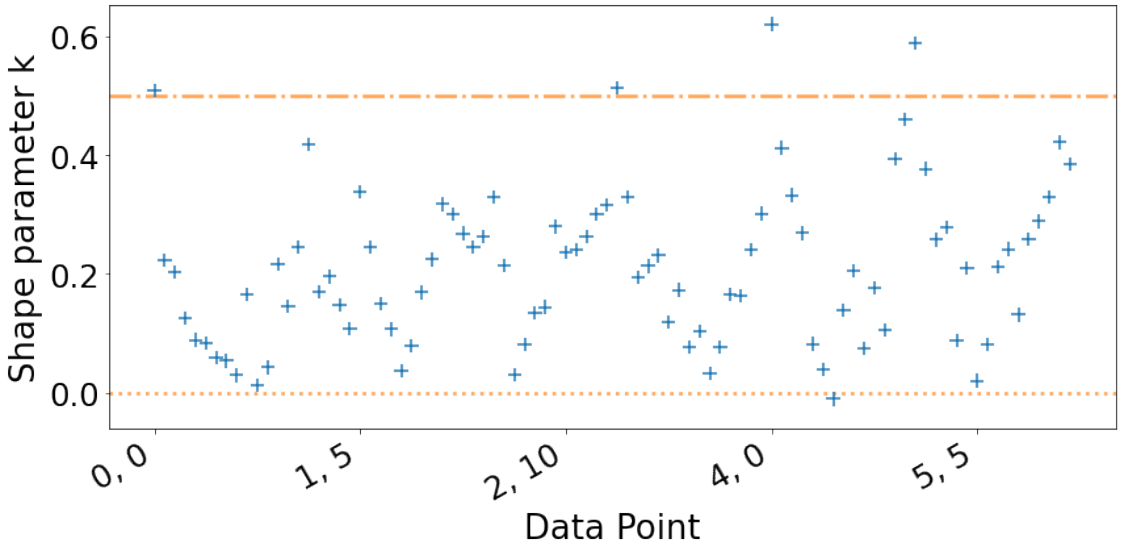
[20]:
```
get_psis_loo_result(pooled_results)
```

```
Computed from 4000 by 90 log-likelihood matrix

         Estimate       SE
elpd_loo  -267.79     6.32
p_loo        2.80        -
------

Pareto k diagnostic values:
                         Count   Pct.
(-Inf, 0.5]   (good)       90  100.0%
 (0.5, 0.7]   (ok)          0    0.0%
   (0.7, 1]   (bad)         0    0.0%
   (1, Inf)   (very bad)    0    0.0%
```



[21]:
```
get_psis_loo_result(hier_results)
```

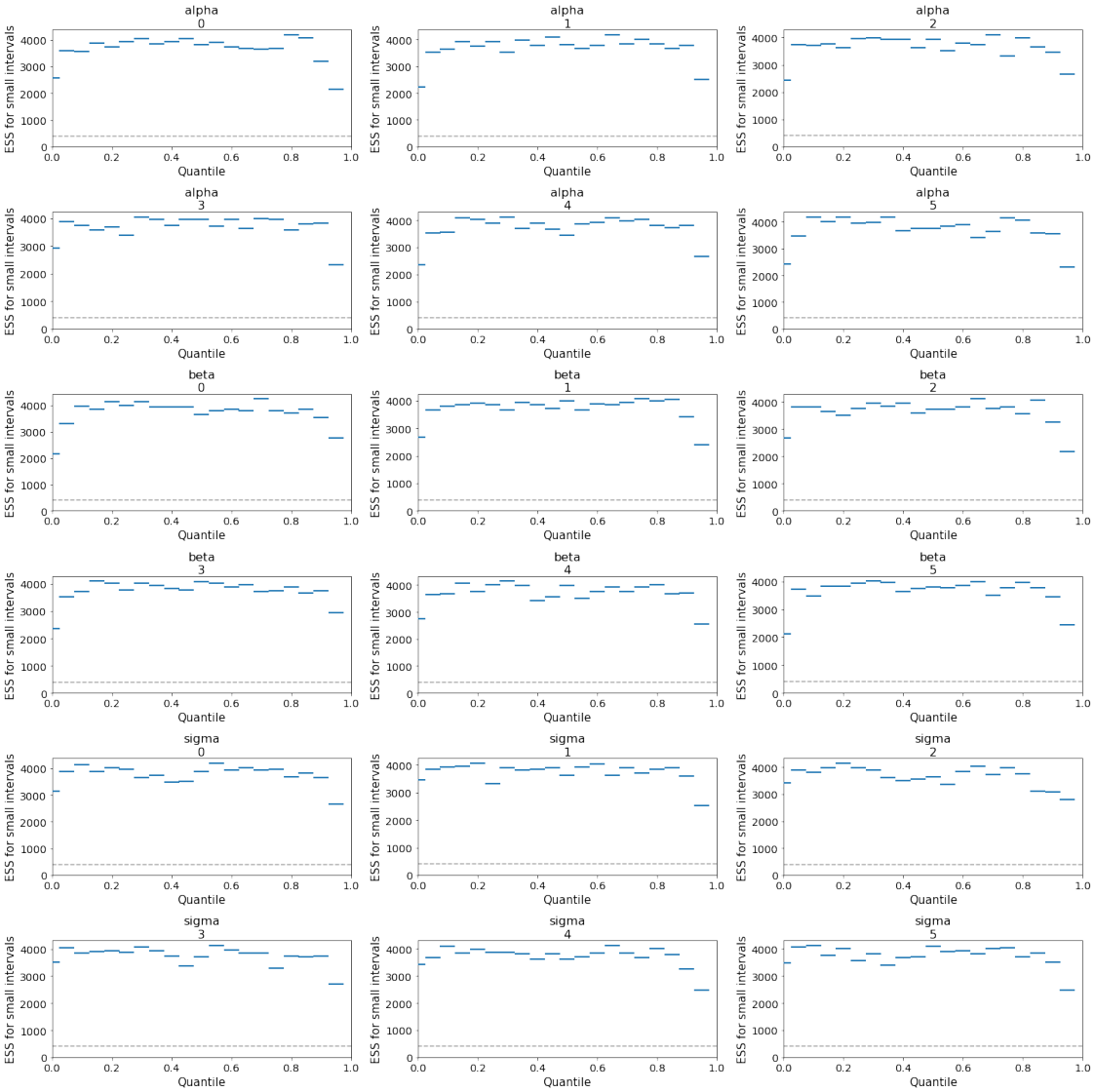
```
Computed from 4000 by 90 log-likelihood matrix

         Estimate       SE
elpd_loo  -196.83     6.96
p_loo       13.35        -
------

Pareto k diagnostic values:
                         Count   Pct.
(-Inf, 0.5]   (good)       86   95.6%
 (0.5, 0.7]   (ok)          4    4.4%
   (0.7, 1]   (bad)         0    0.0%
   (1, Inf)   (very bad)    0    0.0%
```



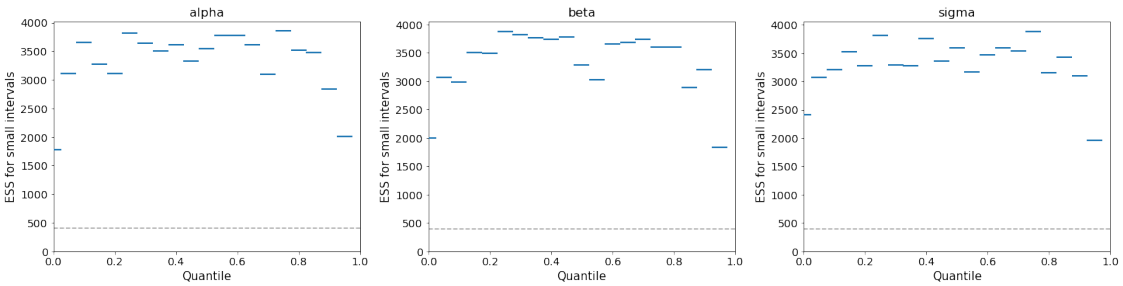
3.2 4.2 Effective Sample Sizes

need to know how to interprete these plots https://mc-stan.org/docs/2_25/reference-manual/effective-sample-size-section.html

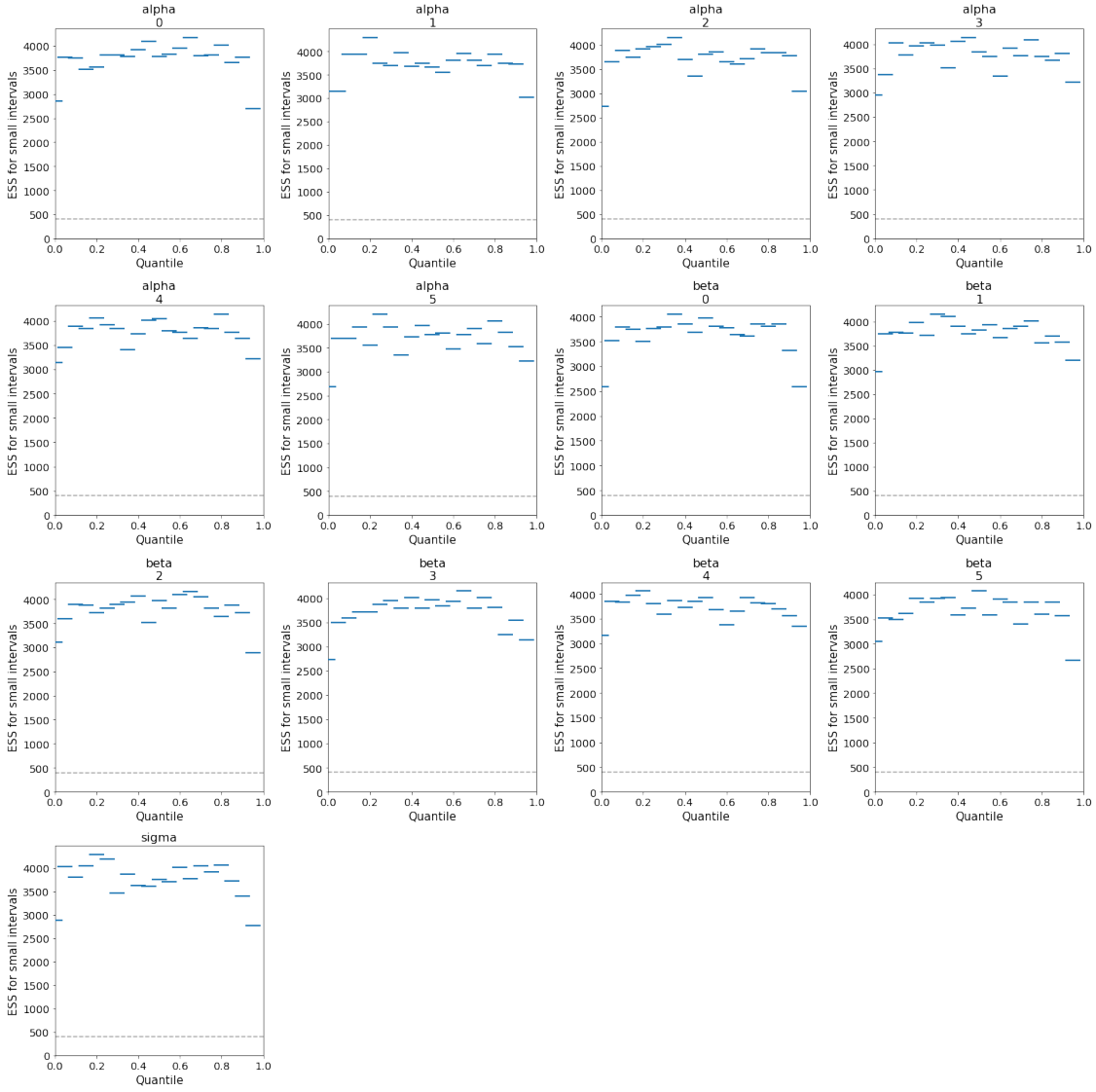
[22]:
```
_ = az.plot_ess(
    separate_results, var_names=var_separate,
    kind="local", marker="_", ms=20, mew=2, figsize=(20, 20)
)
```



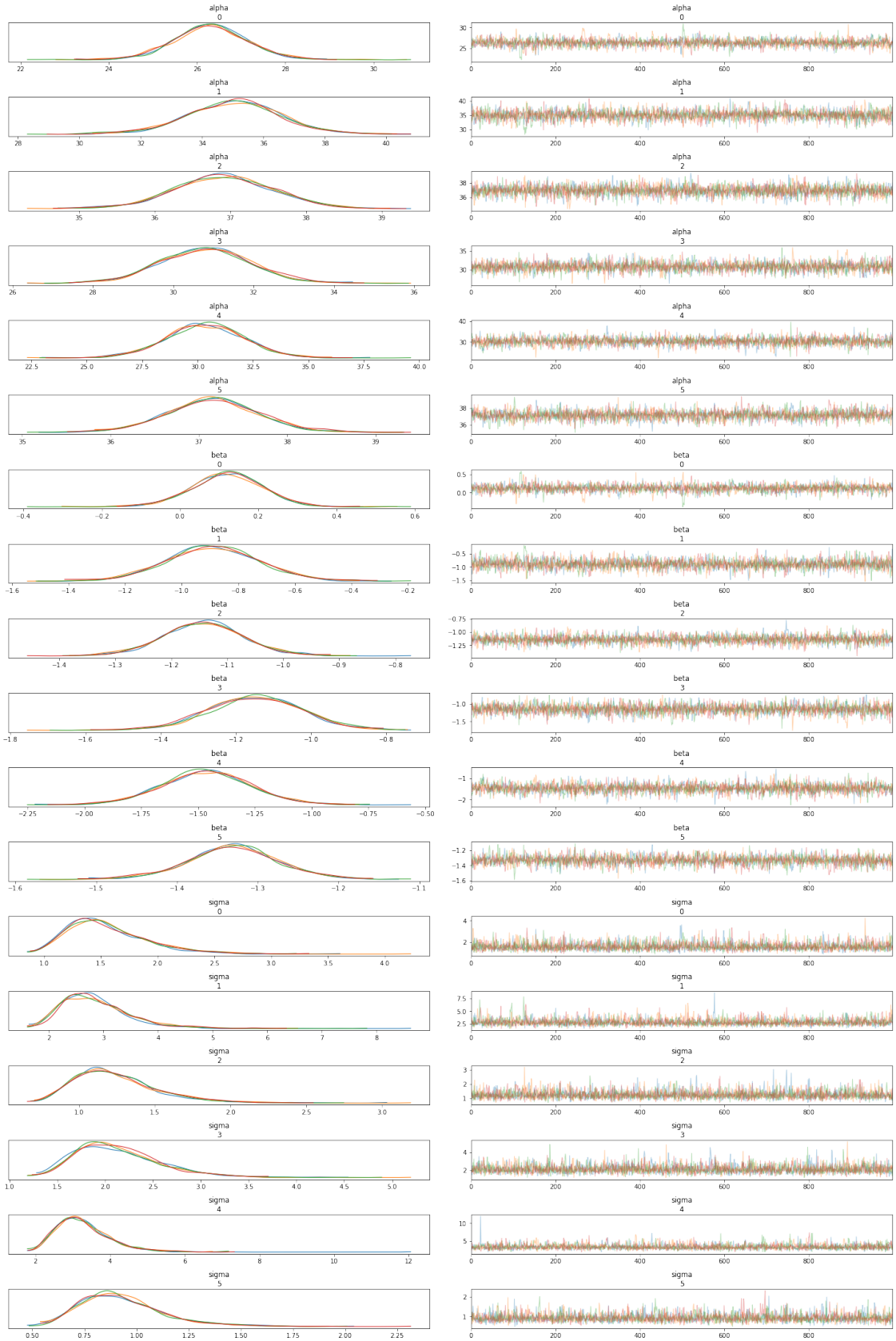
[23]:
```
_ = az.plot_ess(
    pooled_results, var_names=var_pooled,
    kind="local", marker="_", ms=20, mew=2, figsize=(20, 5)
)
```



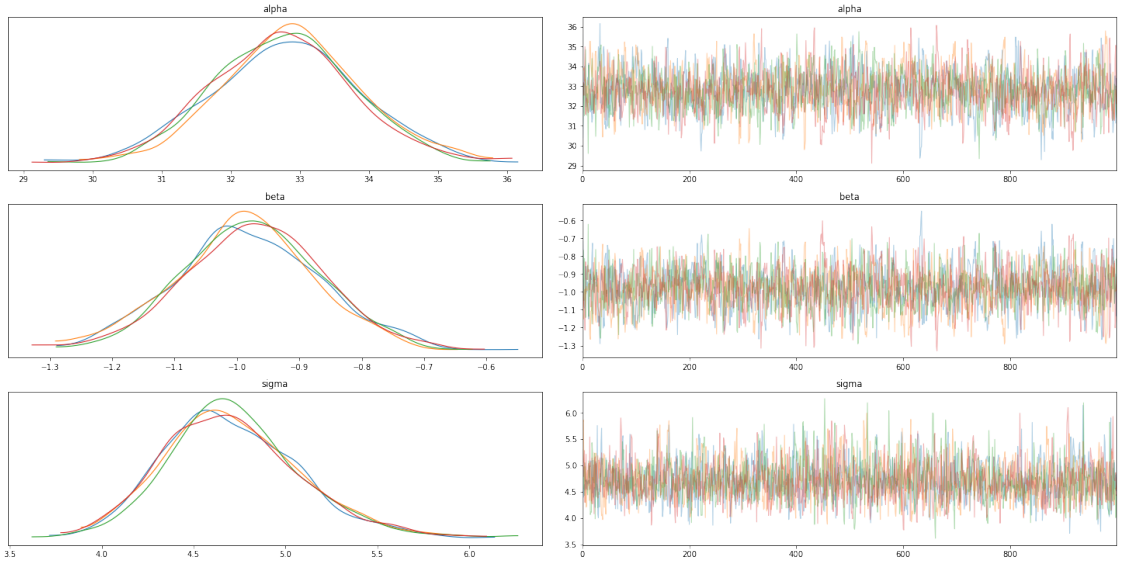
[24]:
```
_ = az.plot_ess(
    hier_results, var_names=var_hier,
    kind="local", marker="_", ms=20, mew=2, figsize=(20, 20)
)
```



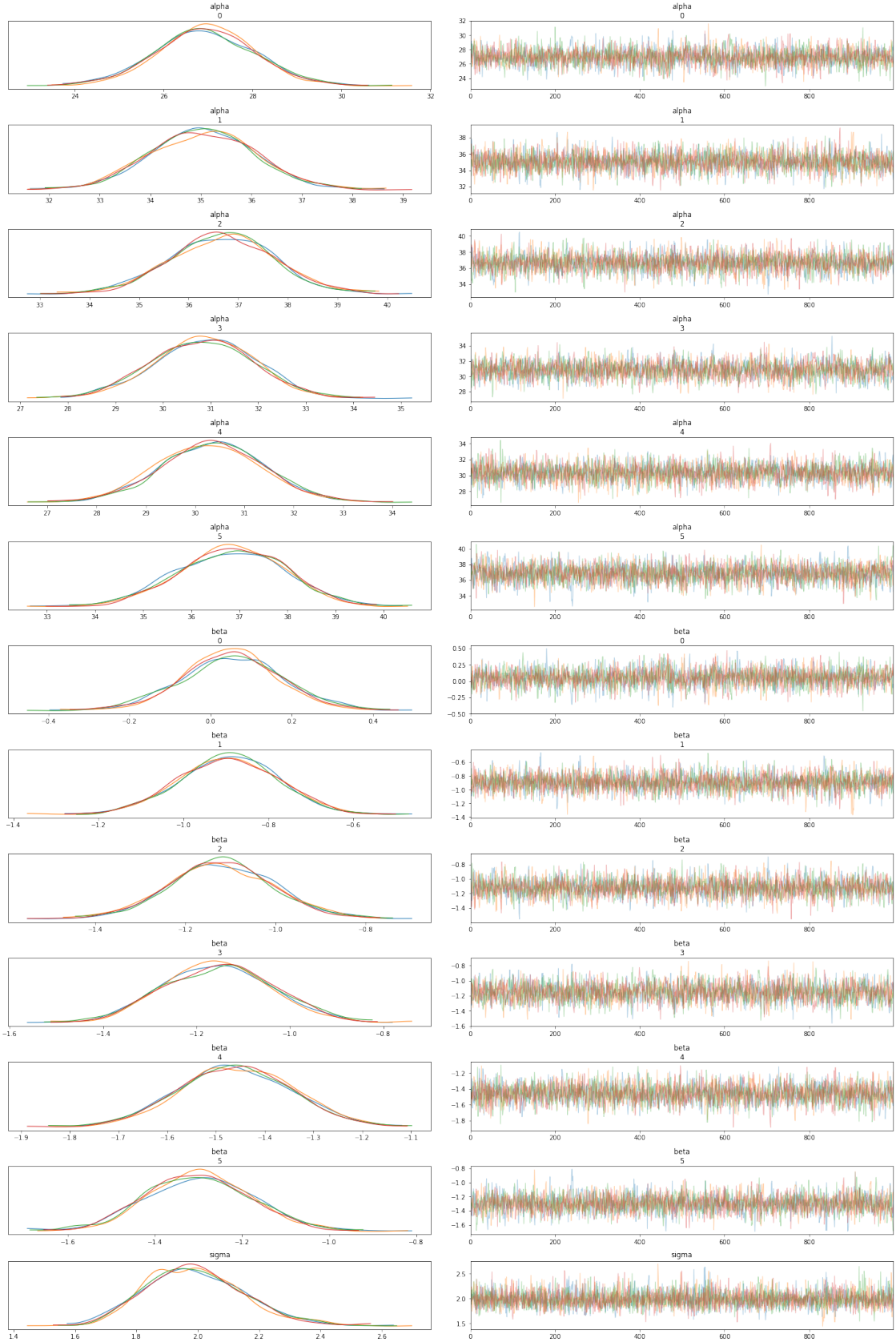
[25]:
```
_ = az.plot_trace(separate_results, var_names = var_separate, figsize=(20, 30))
```


alpha
0
alpha
1
alpha
2
alpha
3
alpha
4
alpha
5
beta
0
beta
1
beta
2
beta
3
beta
4
beta
5
sigma
0
sigma
1
sigma
2
sigma
3
sigma
4
sigma
5

[26]:
```
_ = az.plot_trace(pooled_results, var_names = var_pooled, figsize=(20, 10))
```



[27]:
```
_ = az.plot_trace(hier_results, var_names = var_hier, figsize=(20, 30))
```


alpha
0
alpha
1
alpha
2
alpha
3
alpha
4
alpha
5
beta
0
beta
1
beta
2
beta
3
beta
4
beta
5
sigma

[28]:
```
def get_treedepth(stan_results):
    h = stan_results.to_dataframe(diagnostics=True)
    print('max treedepth for draws: ', h['treedepth__'].max())
    print('min treedepth for draws: ', h['treedepth__'].min())
    print('mean treedepth for draws: ', h['treedepth__'].mean())
    print('treedepth quantiles:\n', h['treedepth__'].quantile([0, 0.25, 0.5, 0.
↪75, 1]))
    print('divergent transitions: ', any(h['divergent__']))
```

need discussion, divergent transitions false means good

[29]:
```
get_treedepth(separate_results)
get_treedepth(pooled_results)
get_treedepth(hier_results)
```

```
max treedepth for draws:  6
min treedepth for draws:  3
mean treedepth for draws:  4.27075
divergent transitions:  False
max treedepth for draws:  4
min treedepth for draws:  1
mean treedepth for draws:  2.79575
divergent transitions:  False
max treedepth for draws:  5
min treedepth for draws:  2
mean treedepth for draws:  3.974
divergent transitions:  False
```

3.3 4.3 Posterior Predictive Checking

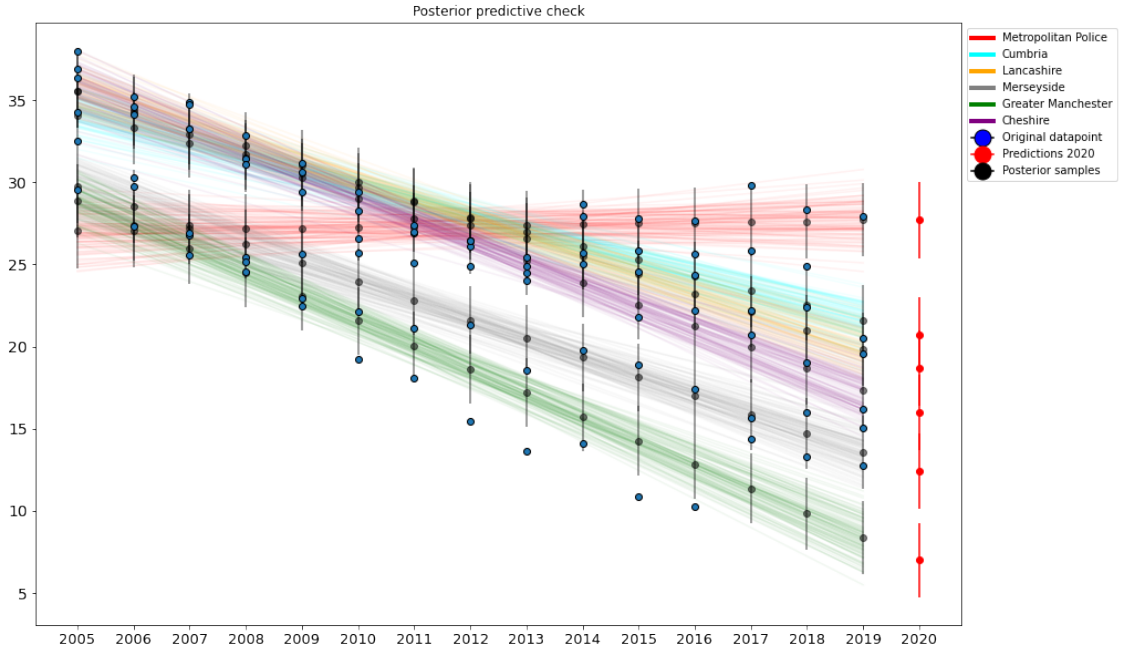
[30]:
```
def plot_posterior_draws(stan_results, accident_data, pooled=False):
    plt.figure(figsize=(15,10))
    year_idx = np.arange(accident_data.shape[1])+1
    actual_years = year_idx + 2004

    colors = ['red', 'cyan', 'orange', 'gray', 'green', 'purple']
    for x in range(1, 7):
        for i in range(100):
            if pooled:
                y = stan_results["beta"][i] * year_idx +␣
↪stan_results["alpha"][i]
            else:
                y = stan_results["beta"][:, x-1][i] * year_idx +␣
↪stan_results["alpha"][:, x-1][i]
```

```
            _ = plt.plot(actual_years, y, color=colors[x-1], alpha=0.05)
        if pooled:
            break


    for x in range(1, 7):
        for j in reversed(range(1, 16)):
            yrep = stan_results['yrep[{},{}]'.format(x, j)]
            _ = plt.errorbar(
                x = actual_years[j-1],
                y = np.mean(yrep),
                yerr=np.std(yrep),
                fmt='--o', zorder=i+j,
                ecolor='black', capthick=2,
                color='black',
                alpha=0.5
            )

    for k in range(1, 7):
        ypred = stan_results['pred[{}]'.format(k)]
        _ = plt.errorbar(
            x = 2020,
            y = np.mean(ypred),
            yerr=np.std(ypred),
            fmt='--o', zorder=i+j+100,
            ecolor='red', capthick=2,
            color='red',
        )


    _ = plt.scatter(np.tile(years, 6), accident_data.flatten(), zorder=j+i+100,␣
↪edgecolors='black')
    # _ = plt.scatter(data_for_stan["years"], data_for_stan["accidentData"],␣
↪zorder=j+i+100, edgecolors='black')
    _ = plt.title("Posterior predictive check")
    _ = plt.legend(bbox_to_anchor=(1.05, 1), loc='lower left', borderaxespad=0.)


    # area_names = ["Metropolitan Police", 'Cumbria','Lancashire',
    #                 'Merseyside','Greater Manchester','Cheshire']

    custom_lines = [
        Line2D([0], [0], color='red', lw=4, label='Metropolitan Police'),
        Line2D([0], [0], color='cyan', lw=4, label='Cumbria'),
        Line2D([0], [0], color='orange', lw=4, label='Lancashire'),
        Line2D([0], [0], color='gray', lw=4, label='Merseyside'),
        Line2D([0], [0], color='green', lw=4, label='Greater Manchester'),
```

```
        Line2D([0], [0], color='purple', lw=4, label='Cheshire'),
        Line2D([0], [0], marker='o', color='black', label='Original datapoint',␣
↪markerfacecolor='b', markersize=15),
        Line2D([0], [0], marker='o', color='red', label='Predictions 2020',␣
↪markersize=15),
        Line2D([0], [0], marker='o', color='black', label='Posterior samples',␣
↪markersize=15),
    ]

    _ = plt.legend(handles=custom_lines, bbox_to_anchor=(1, 1))
    _ = plt.xticks(np.arange(2005, 2021), fontsize=13)
    _ = plt.yticks(fontsize=14)
```

[31]:
```
plot_posterior_draws(hier_results, accident_data)
```

```
WARNING:matplotlib.legend:No handles with labels found to put in legend.
```



[32]:
```
az.r2_score(accident_data, hier_results["yrep"])
```

[32]:
```
r2        0.885536
r2_std    0.005628
dtype: float64
```

[33]:
```
np.mean(hier_results["pred"][:,0])*8961989/10000
```

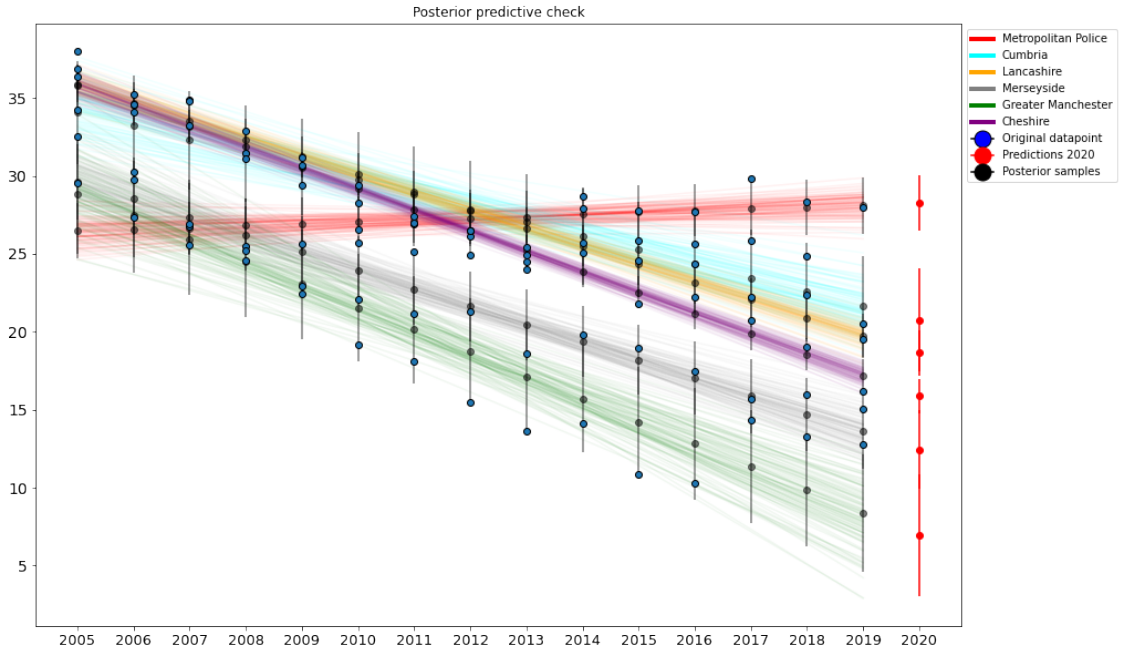
[33]:
```
24827.525814164677
```

[34]:
```
accident_data
```

[34]:
```
array([[29.57, 27.31, 25.58, 25.46, 25.64, 26.57, 26.91, 26.47, 25.43,
        28.66, 27.77, 27.66, 29.84, 28.33, 27.96],
       [38.  , 34.12, 34.88, 31.46, 29.42, 25.68, 25.12, 24.9 , 24.52,
        27.94, 25.86, 25.62, 25.82, 24.88, 20.5 ],
       [36.39, 35.23, 34.77, 32.88, 31.17, 29.42, 26.99, 26.1 , 24.89,
        25.71, 24.59, 24.35, 22.23, 22.39, 19.54],
       [34.26, 30.28, 26.79, 24.54, 22.46, 22.11, 21.14, 21.33, 18.57,
        19.8 , 18.92, 17.44, 15.67, 15.99, 15.03],
       [32.52, 29.72, 26.94, 25.2 , 22.96, 19.2 , 18.08, 15.44, 13.62,
        14.12, 10.84, 10.25, 14.35, 13.27, 12.77],
       [36.89, 34.59, 33.23, 31.09, 30.66, 28.29, 27.42, 26.48, 24.01,
        25.06, 21.77, 22.22, 20.72, 19.02, 16.21]])
```

[35]:
```
plot_posterior_draws(separate_results, accident_data)
```

```
WARNING:matplotlib.legend:No handles with labels found to put in legend.
```

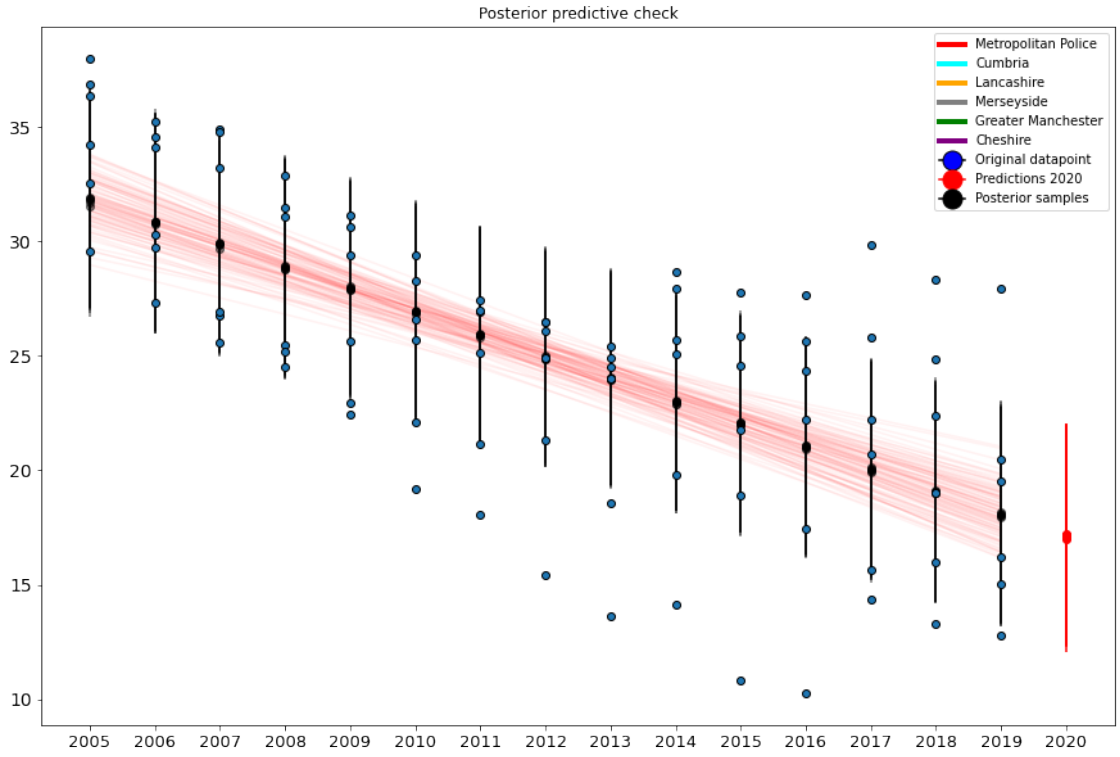


[36]:
```
az.r2_score(accident_data, separate_results["yrep"])
```

[36]:
```
r2        0.872255
r2_std    0.087042
dtype: float64
```

[37]:
```
plot_posterior_draws(pooled_results, accident_data, pooled=True)
```

```
WARNING:matplotlib.legend:No handles with labels found to put in legend.
```



[38]:
```
az.r2_score(accident_data, pooled_results["yrep"])
```

[38]:
```
r2        0.440616
r2_std    0.005579
dtype: float64
```

3.4 4.4 Prior Sensitivity Test

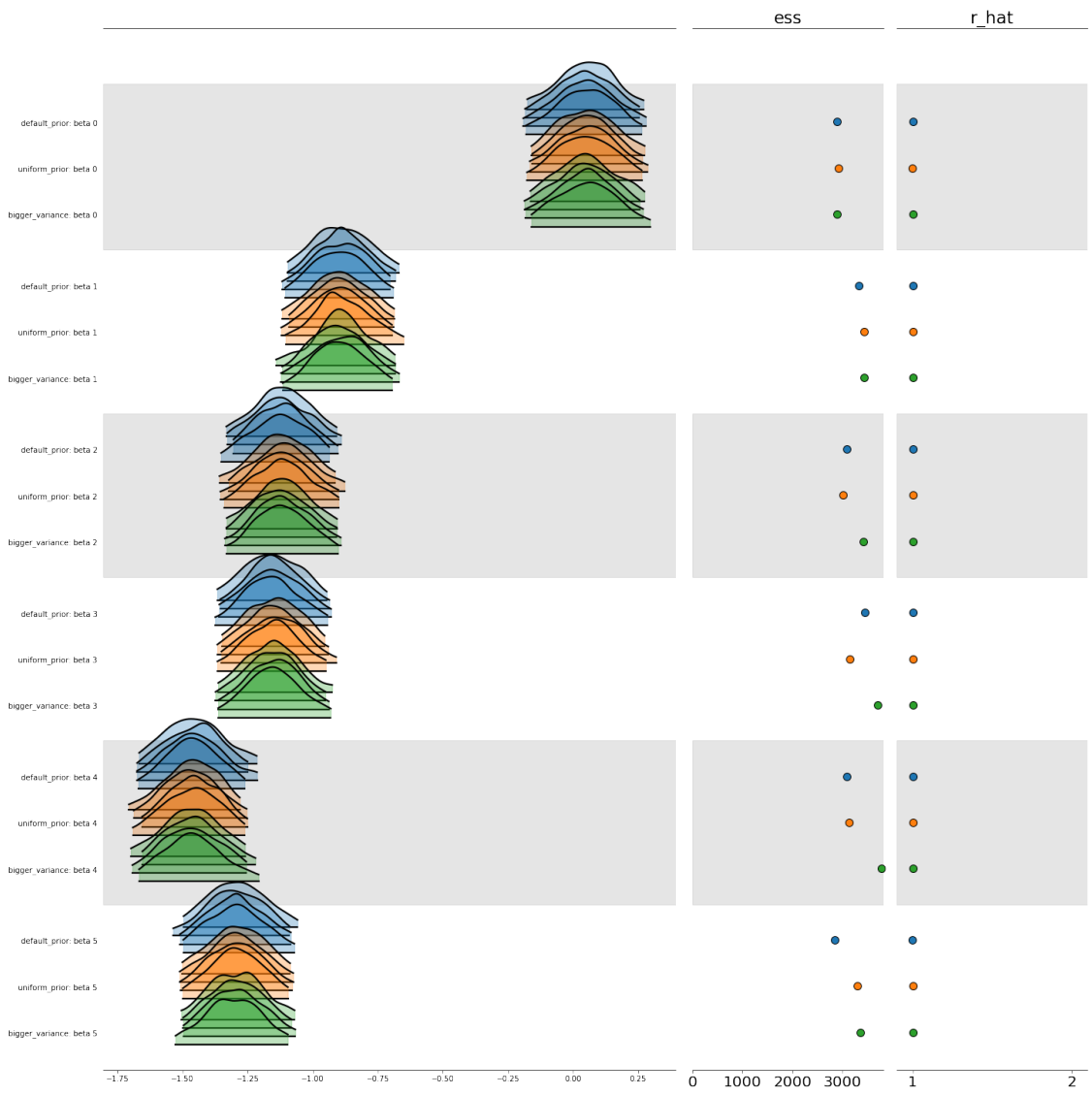
[55]:
```
data_dict = dict()
names = ["default_prior", "uniform_prior", "bigger_variance"]
for i in range(3):
    current_stan_data = dict(
        N = accident_data.shape[0],
        Y = accident_data.shape[1],
        accidentData = accident_data,
        years = np.arange(1, accident_data.shape[1]+1), # stan index starts␣
 ↪from 1
        xpred=2020,
        prior_choice= i+1,
        cov_coeff=-0.1,
```

```
    )
    data_dict[names[i]] = current_stan_data
```

[40]:
```
def get_plot_forest(stan_model, data_dict, pooled=False):
    if pooled:
        figsize = (20, 5)
    else:
        figsize = (20, 20)
    result_dict = dict()
    for key, stan_data in data_dict.items():
        print("Generating results with prior:{} {}".
 ↪format(stan_data["prior_choice"], key))
        sampling_result = stan_model.sampling(data=stan_data)
        #print(sampling_result)
        result_dict[key] = sampling_result
    _ = az.plot_forest(
    list(result_dict.values()),
    model_names=list(result_dict.keys()), var_names=["beta"], markersize=10,
    kind='ridgeplot', ridgeplot_overlap=3, ridgeplot_alpha=0.3, r_hat=True, \
        ess=True, figsize=figsize, textsize=20)
    plt.rcParams['xtick.labelsize'] = 20
    plt.rcParams['ytick.labelsize'] = 20
    plt.show()
```

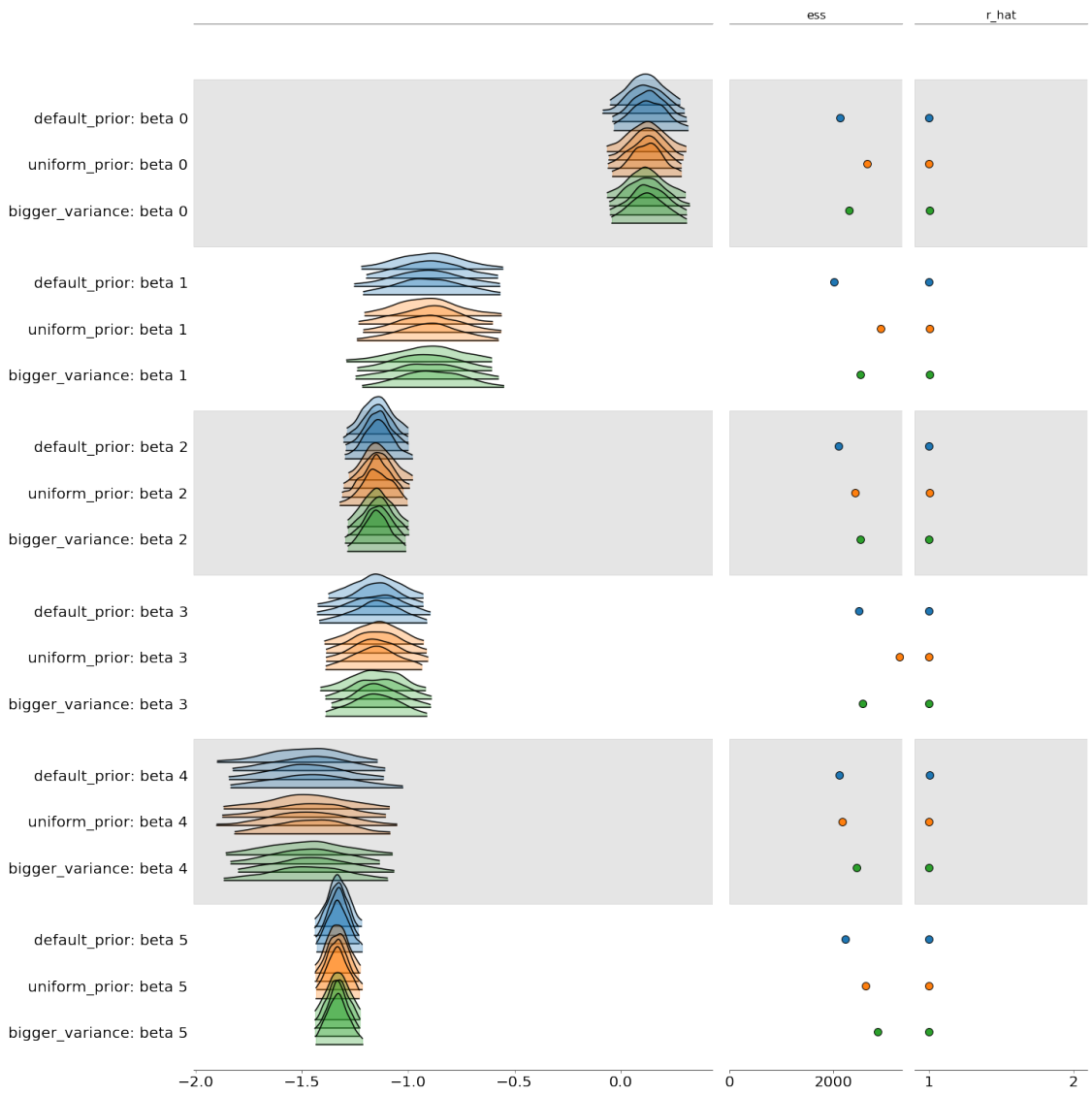
[41]:
```
get_plot_forest(hier_stan_model, data_dict)
```

```
Generating results with prior:1 default_prior
Generating results with prior:2 uniform_prior
Generating results with prior:3 bigger_variance
```



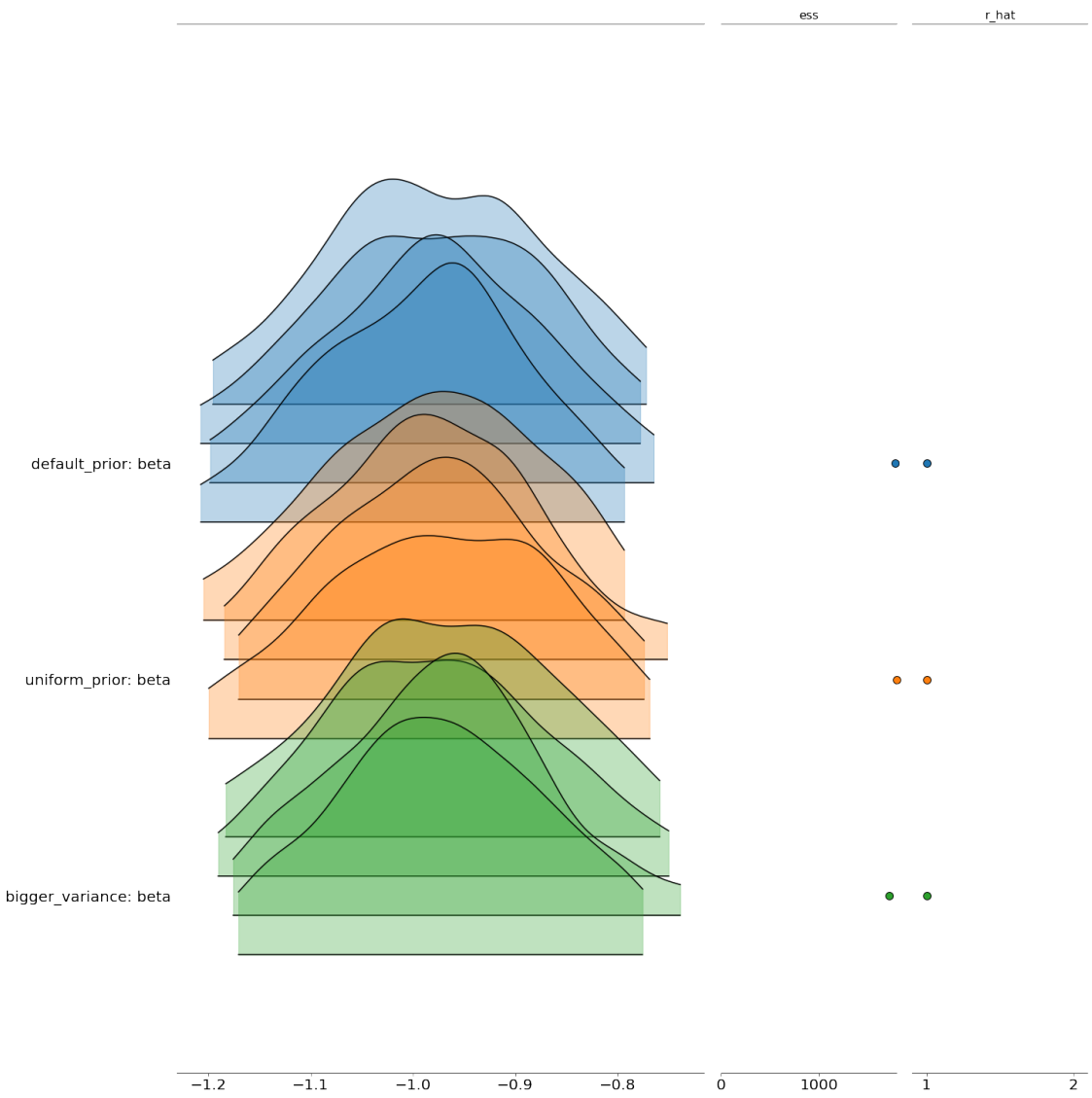
[42]:
```
get_plot_forest(separate_stan_model, data_dict)
```

```
Generating results with prior:1 default_prior
Generating results with prior:2 uniform_prior
Generating results with prior:3 bigger_variance
```



[57]:
```
get_plot_forest(pooled_stan_model, data_dict)
```

```
Generating results with prior:1 default_prior
Generating results with prior:2 uniform_prior
Generating results with prior:3 bigger_variance
```



3.5 A1. Hierarchical Model with Covariates

[44]:
```
hier_cov_model_name = 'accident_hierarchical_cov.stan'
hier_cov_model = pystan.StanModel(file=model_path + '/' + hier_cov_model_name)
print(hier_cov_model.model_code)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL
anon_model_c61a7838f3f01dd545a64710e9ee5eac NOW.
```

```
//
// This Stan program defines a simple model, with a
// vector of values 'y' modeled as normally distributed
// with mean 'mu' and standard deviation 'sigma'.
```

```
//
// Learn more about model development with Stan at:
//
//    http://mc-stan.org/users/interfaces/rstan.html
//    https://github.com/stan-dev/rstan/wiki/RStan-Getting-Started
//

data {
  int<lower=0> N; // the number of police force
  int<lower=0> Y; // the number of years has been studied, year 2005 corresponds
to 1
  matrix[N,Y] accidentData;//accident data
  int prior_choice; // choose different setup for prior distribution
  int xpred; // year of prediction (actual year)
  real cov_coeff; // covariance coefficient
}


parameters {
  real mu_alpha;
  real mu_beta;
  real<lower=0> sigma_alpha;
  real<lower=0> sigma_beta;
  // vector<lower=0>[N] sigma;
  real<lower=0> sigma;
  matrix[2, N] alpha_beta; // the first row is alpha, the second row is beta
}


transformed parameters{
  cov_matrix[2] cov_m;
  vector[2] mean_vec;
  matrix[N,Y] mu;

  mean_vec[1] = mu_alpha;
  mean_vec[2] = mu_beta;

  cov_m[1,1] = sigma_alpha * sigma_alpha;
  cov_m[1,2] = cov_coeff * sigma_alpha * sigma_beta;
  cov_m[2,1] = cov_coeff * sigma_alpha * sigma_beta;
  cov_m[2,2] = sigma_beta * sigma_beta;

  for(i in 1:N)
    for(j in 1:Y)
      mu[i,j]=alpha_beta[1, i]+alpha_beta[2, i]*j;
}
```

```
model {
  if (prior_choice==3){
    // bigger variance
    mu_alpha~normal(30,40);
    mu_beta~normal(0,10);
    //sigma_alpha~normal(10,10);
    //sigma_beta~normal(3,6);
  } else if (prior_choice==2){
    // uniform prior
  } else {
    // default choice with moderate variance
    mu_alpha~normal(30,20);
    mu_beta~normal(0,4.85);
    //sigma_alpha~normal(10,5);
    //sigma_beta~normal(3,3);
  }

  //for each police force
  for(i in 1:N){
    alpha_beta[,i]~multi_normal(mean_vec, cov_m);
  }

  //for each police force
  for(i in 1:N){
    //for each observed year
    for(j in 1:Y){
     // accidentData[i,j]~normal(mu[i,j],sigma[i]);
     accidentData[i,j]~normal(mu[i,j],sigma); // share sigma
    }
  }
}


generated quantities{
  //log likelihood
  matrix[N,Y] log_lik;
  matrix[N,Y] yrep;
  //accident prediction in 2020 in different police force
  vector[N] pred;
  vector[N] alpha;
  vector[N] beta;

  //for each police force
  for(i in 1:N){
    // 2005 -> 1, 2006 -> 2, …, 2020 -> 16
    // pred[i]=normal_rng(alpha_beta[1, i]+alpha_beta[2,
i]*(xpred-2004),sigma[i]);
    // share sigma
```

```
    alpha[i] = alpha_beta[1, i];
    beta[i] = alpha_beta[2, i];
    pred[i]=normal_rng(alpha_beta[1, i]+alpha_beta[2, i]*(xpred-2004),sigma);
  }

  for(i in 1:N){
    for(j in 1:Y){
      // do posterior sampling and try to reproduce the original data
      // yrep[i,j]=normal_rng(mu[i,j],sigma[i]);
      yrep[i,j]=normal_rng(mu[i,j],sigma);
      // prepare log likelihood for PSIS-LOO
      // log_lik[i,j]=normal_lpdf(accidentData[i,j]|mu[i,j],sigma[i]);
      // share sigma
      log_lik[i,j]=normal_lpdf(accidentData[i,j]|mu[i,j],sigma);
    }
  }

}
```

[45]:
```
hiercov_results = test_stan_model(hier_cov_model, accident_data,␣
↪verbose=verbose, cov_coeff=-0.1)
```

```
WARNING:pystan:n_eff / iter below 0.001 indicates that the effective sample size
has likely been overestimated
WARNING:pystan:Rhat above 1.1 or below 0.9 indicates that the chains very likely
have not mixed
```

```
Inference for Stan model: anon_model_c61a7838f3f01dd545a64710e9ee5eac.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.
```

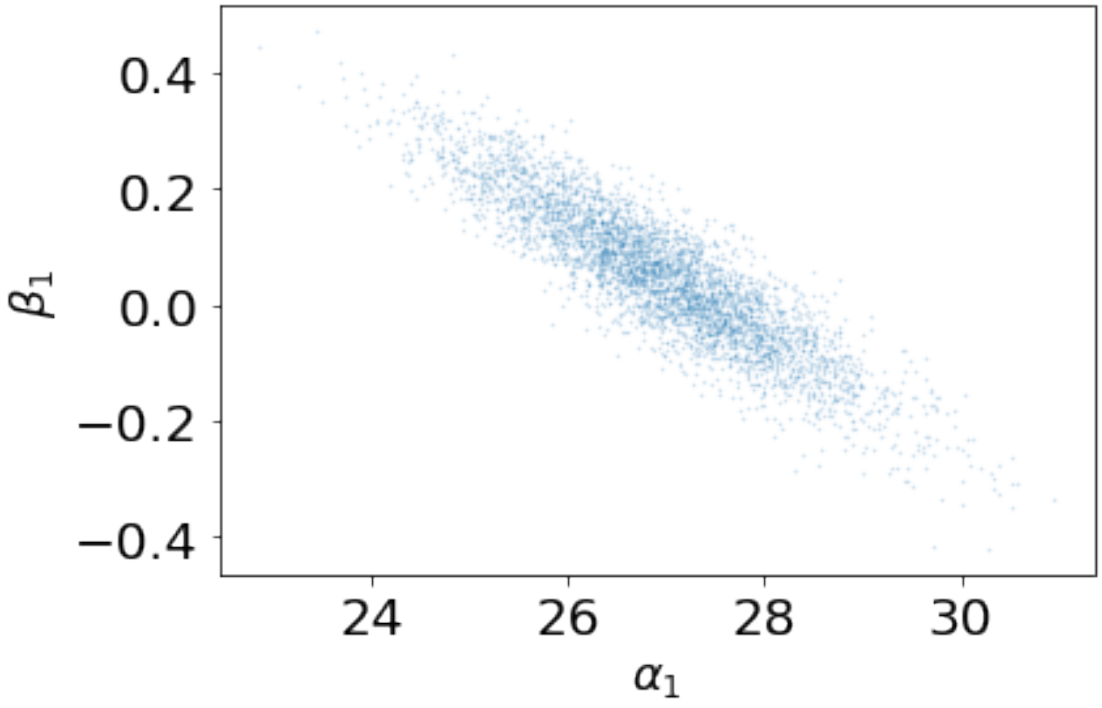
	mean	se_mean	sd	2.5%	25%	50%	75%	97.5%	n_eff	Rhat
alpha[1]	26.93	0.02	1.11	24.78	26.21	26.93	27.67	29.1	3294	1.0
alpha[2]	35.05	0.02	1.05	32.97	34.34	35.07	35.77	37.05	3366	1.0
alpha[3]	36.71	0.02	1.1	34.55	35.96	36.69	37.42	38.92	3402	1.0
alpha[4]	30.9	0.02	1.04	28.87	30.19	30.89	31.58	32.99	3614	1.0
alpha[5]	30.31	0.02	1.05	28.3	29.58	30.3	31.01	32.42	2923	1.0
alpha[6]	36.84	0.02	1.05	34.71	36.13	36.86	37.53	38.88	3225	1.0
beta[1]	0.05	2.1e-3	0.12	-0.19	-0.03	0.05	0.13	0.29	3258	1.0
beta[2]	-0.9	2.0e-3	0.12	-1.12	-0.98	-0.9	-0.82	-0.67	3449	1.0
beta[3]	-1.12	2.1e-3	0.12	-1.36	-1.2	-1.12	-1.04	-0.89	3432	1.0
beta[4]	-1.16	1.9e-3	0.11	-1.38	-1.23	-1.16	-1.08	-0.94	3607	1.0
beta[5]	-1.46	2.1e-3	0.12	-1.69	-1.53	-1.46	-1.38	-1.23	2926	1.0

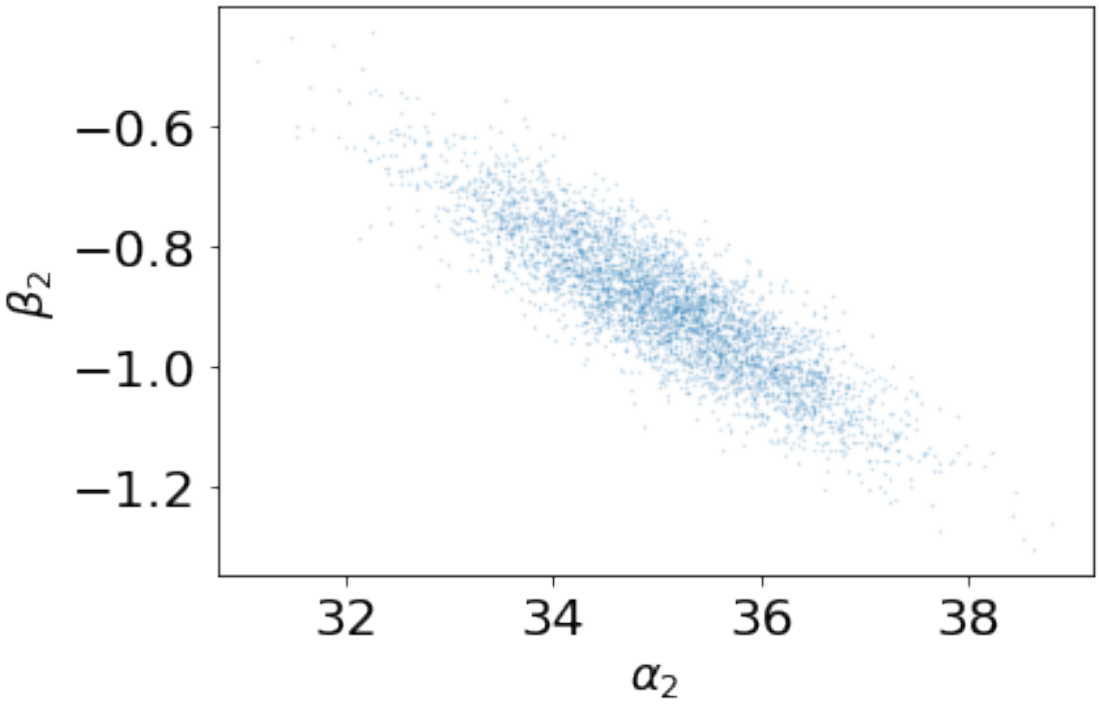
```
beta[6]    -1.3  2.0e-3   0.12  -1.52  -1.37  -1.29  -1.22  -1.06   3229    1.0
sigma      1.99  2.5e-3   0.17    1.7   1.87   1.98    2.1   2.35   4513    1.0

Samples were drawn using NUTS at Mon Dec 14 16:01:01 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).
```

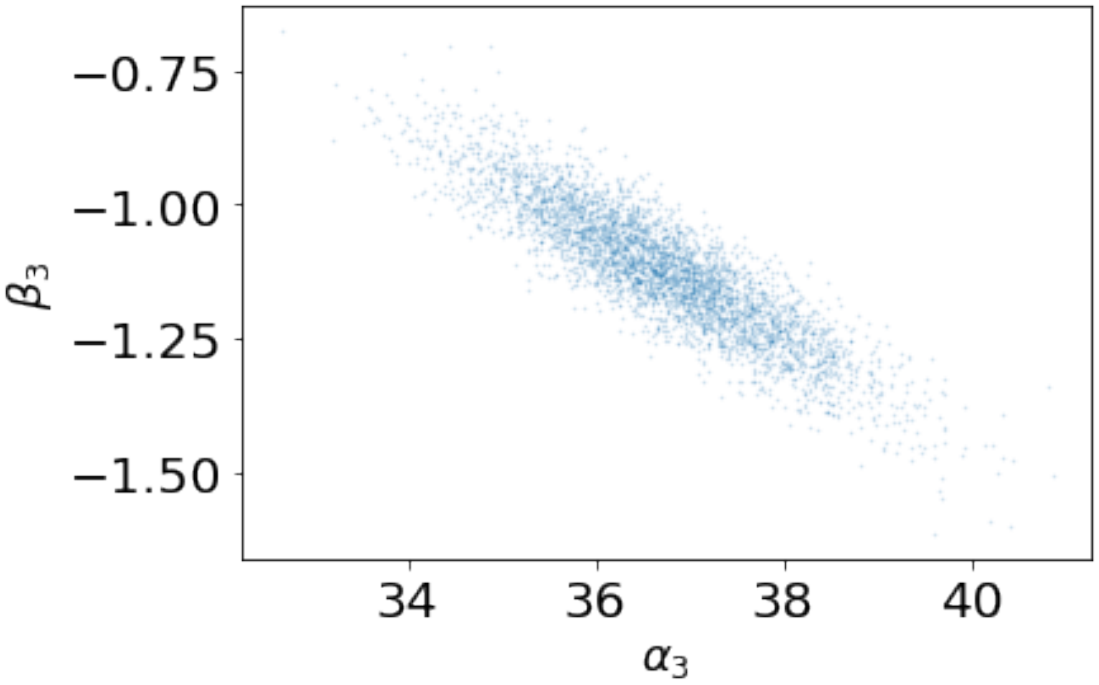
[46]:
```
extracted_results=hiercov_results.extract();
```

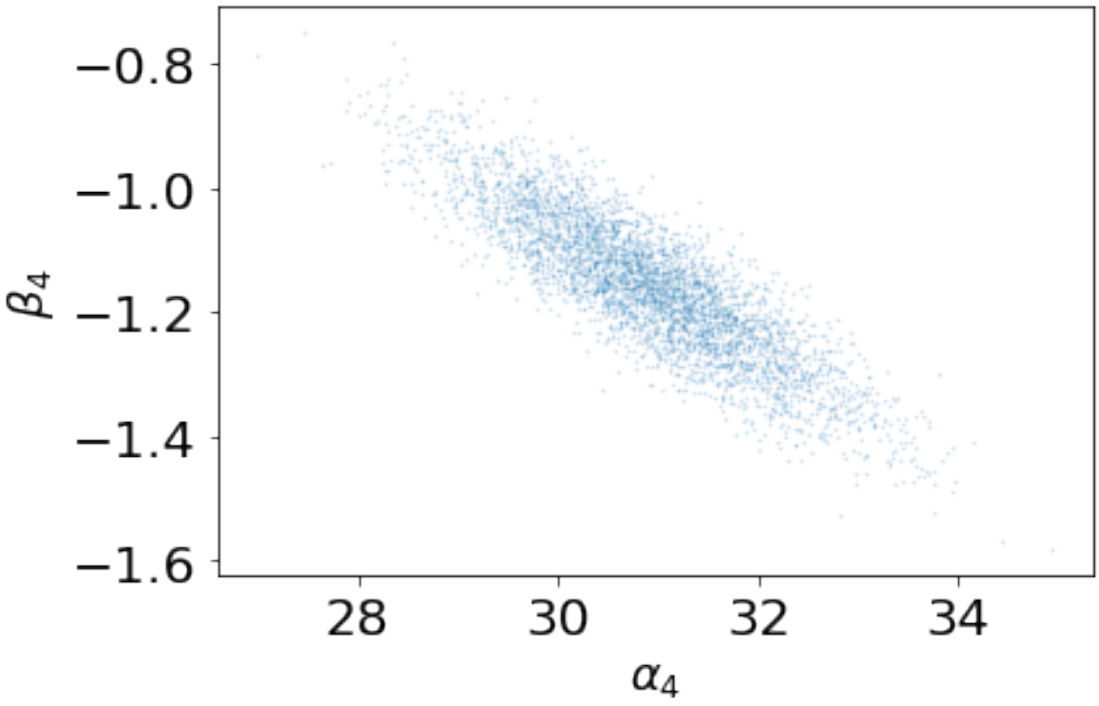
[47]:
```
def plot_cov(stan_results):
    extracted_results=stan_results.extract();
    alpha = extracted_results["alpha"]
    beta = extracted_results["beta"]
    for i in range(alpha.shape[1]):
        plt.figure(figsize=(6, 4))
        plt.scatter(alpha[:,i], beta[:,i], alpha=0.2, s=1, marker='.')
        plt.ylabel(r'$\beta_{}$'.format(i+1),fontsize=18)
        plt.xlabel(r'$\alpha_{}$'.format(i+1),fontsize=18)
```

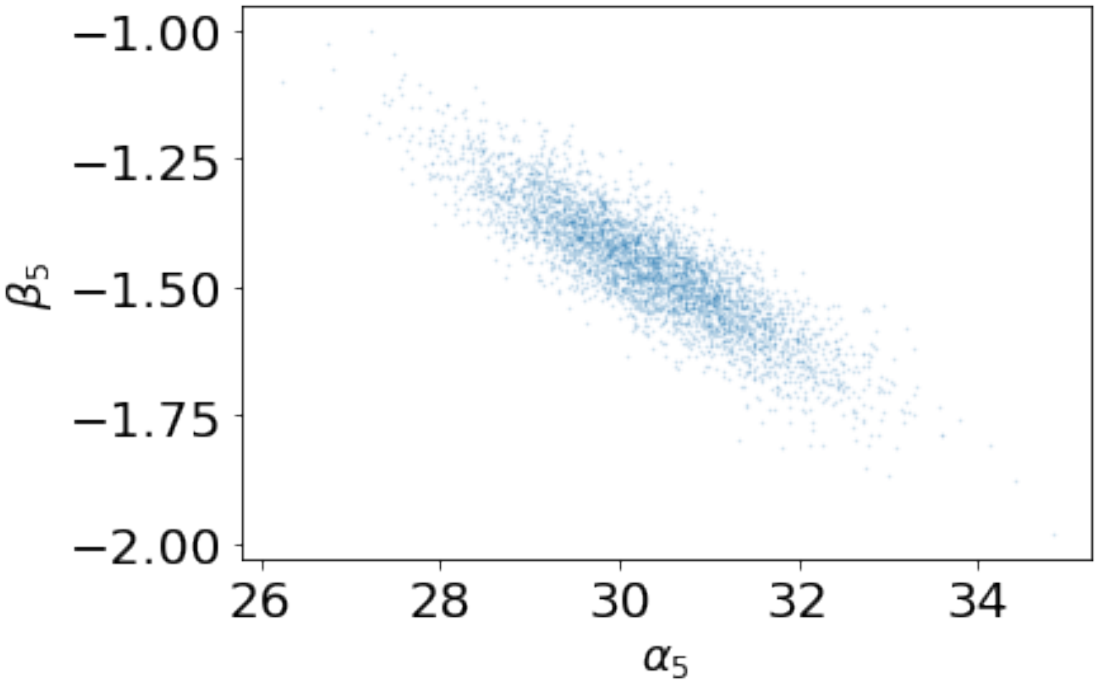
[48]:
```
plot_cov(hiercov_results)
```

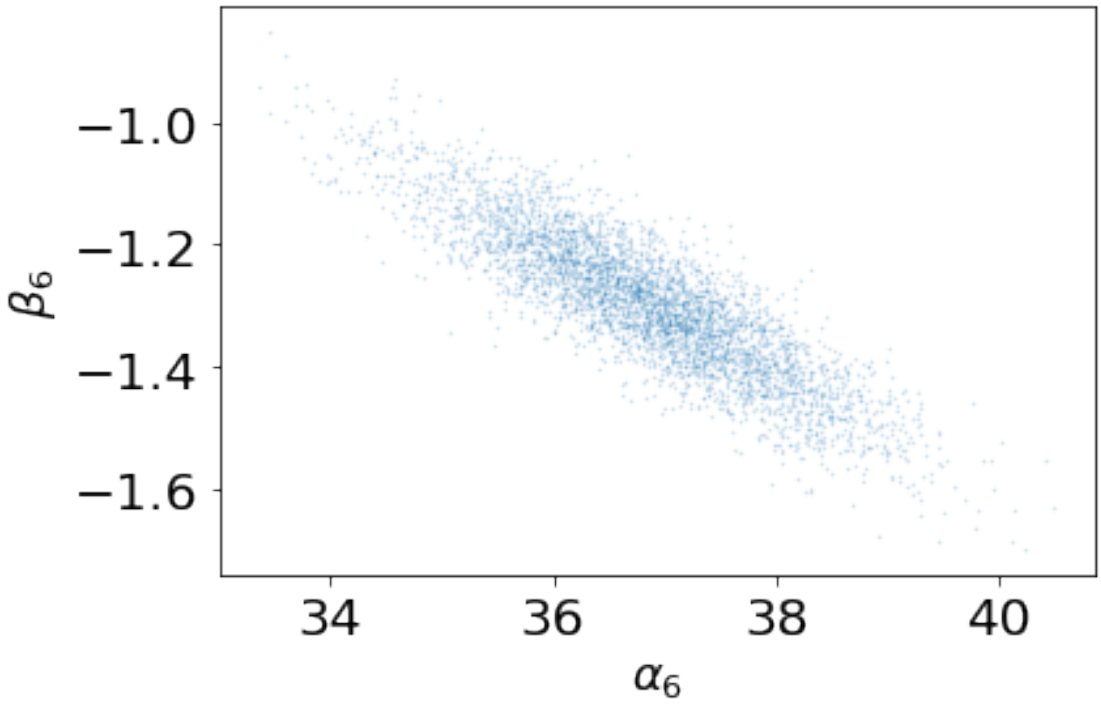



−0.6
−0.8
−1.0
−1.2
β_2
32
34
36
38
α_2

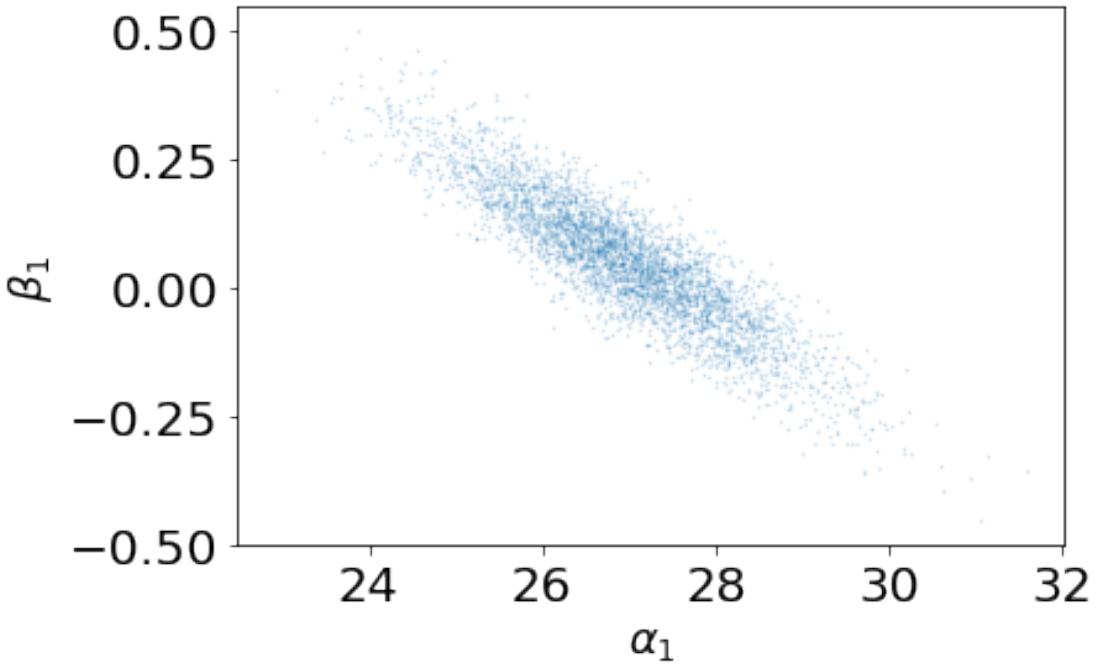

−0.75
−1.00
−1.25
−1.50
β_3
34
36
38
40
α_3

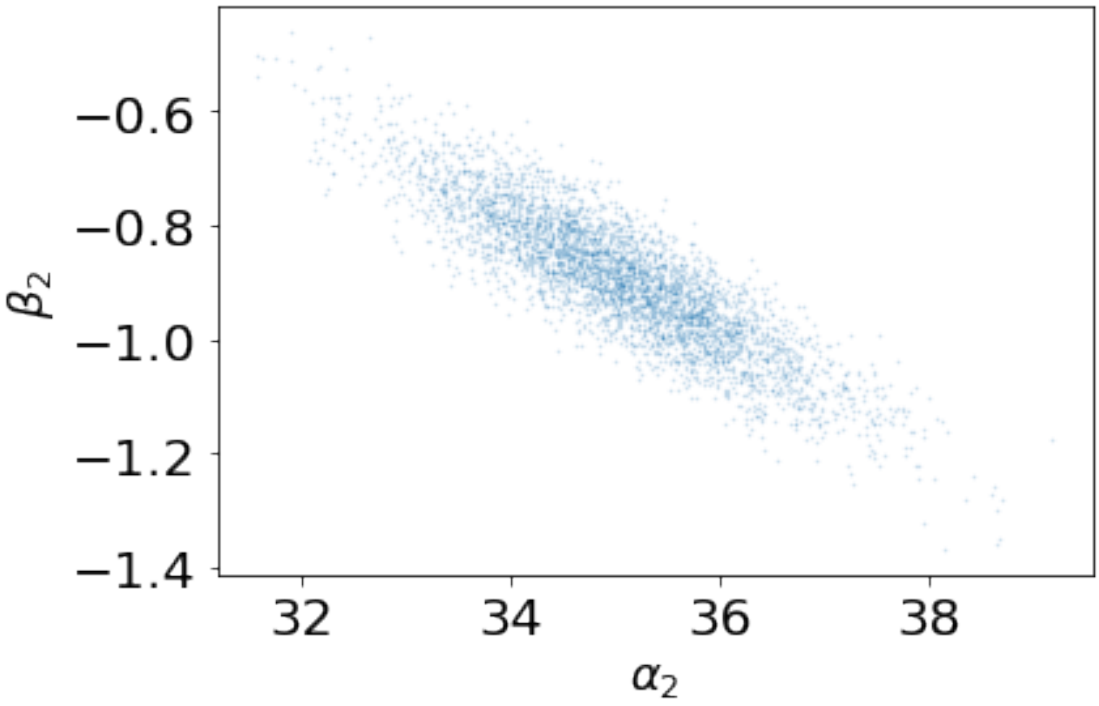

−0.8
−1.0
−1.2
−1.4
−1.6
β_4
28
30
32
34
α_4

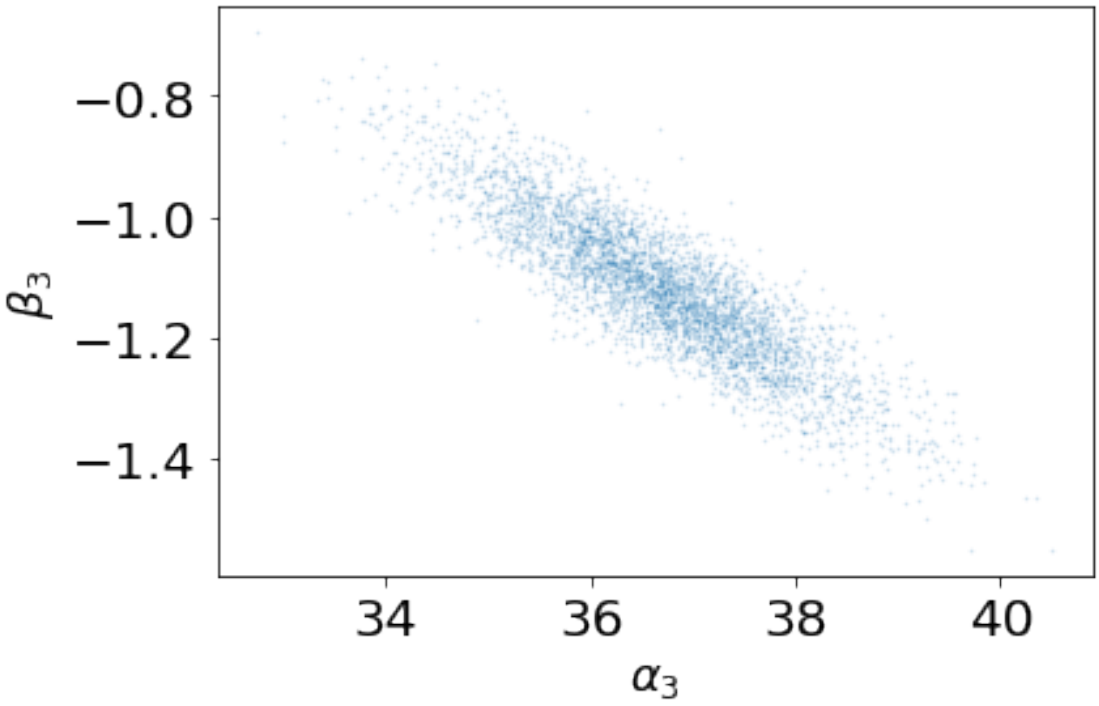

−1.00
−1.25
−1.50
−1.75
−2.00
β_5
26
28
30
32
34
α_5

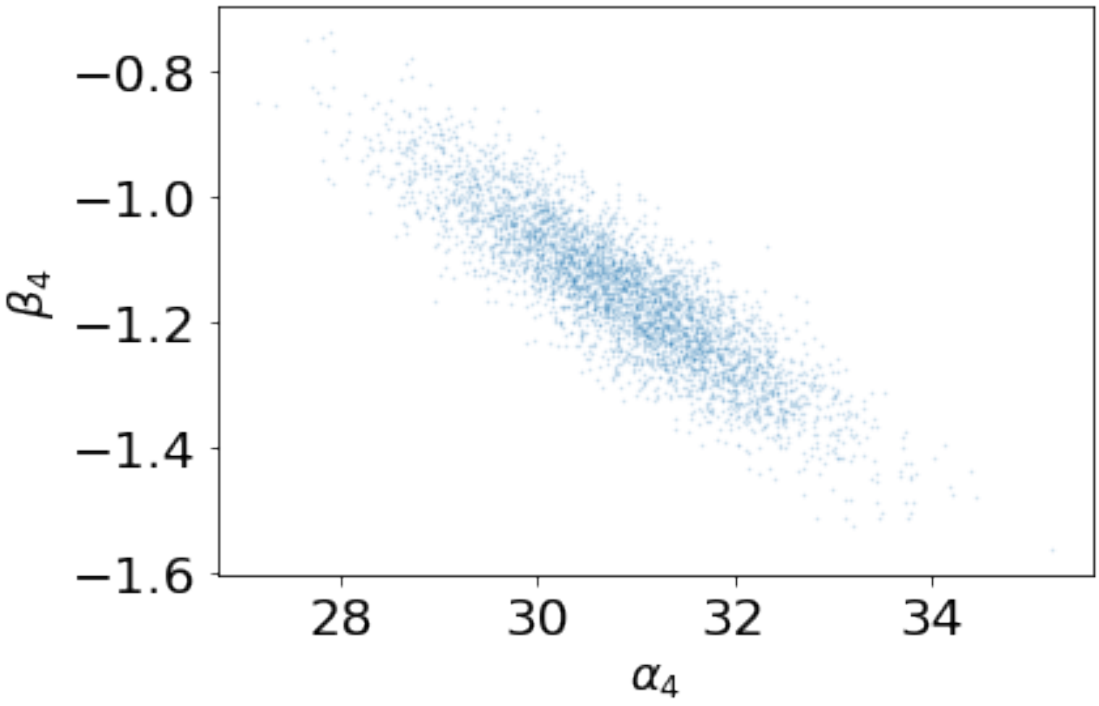


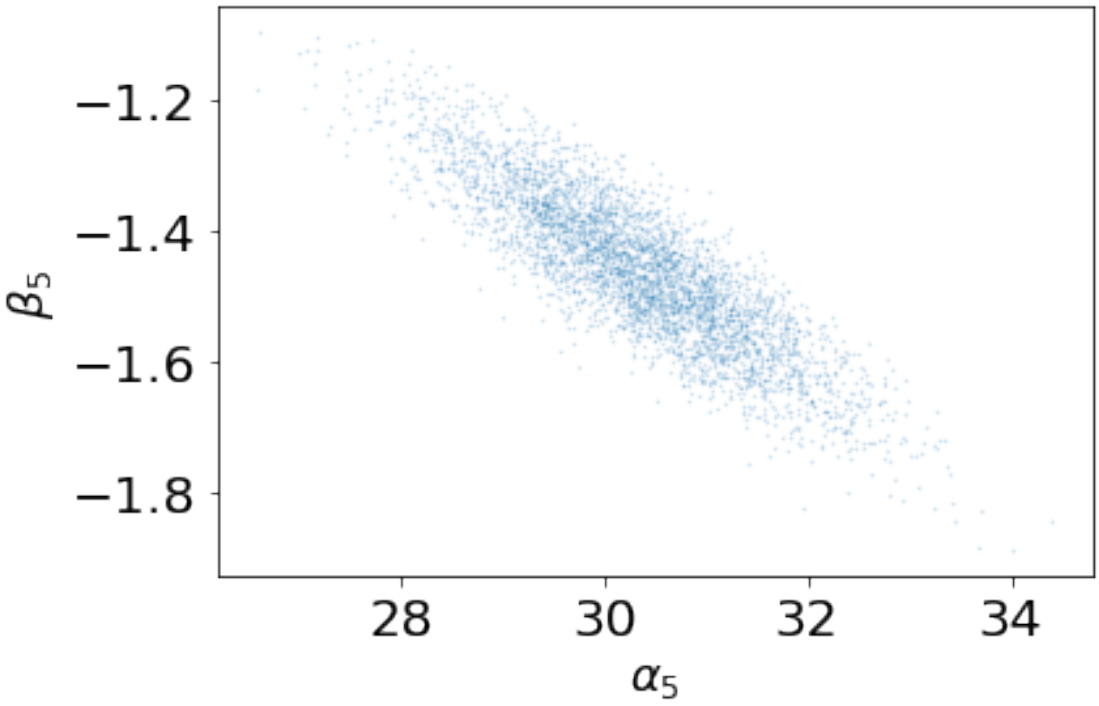
[49]:
```
plot_cov(hier_results)
```

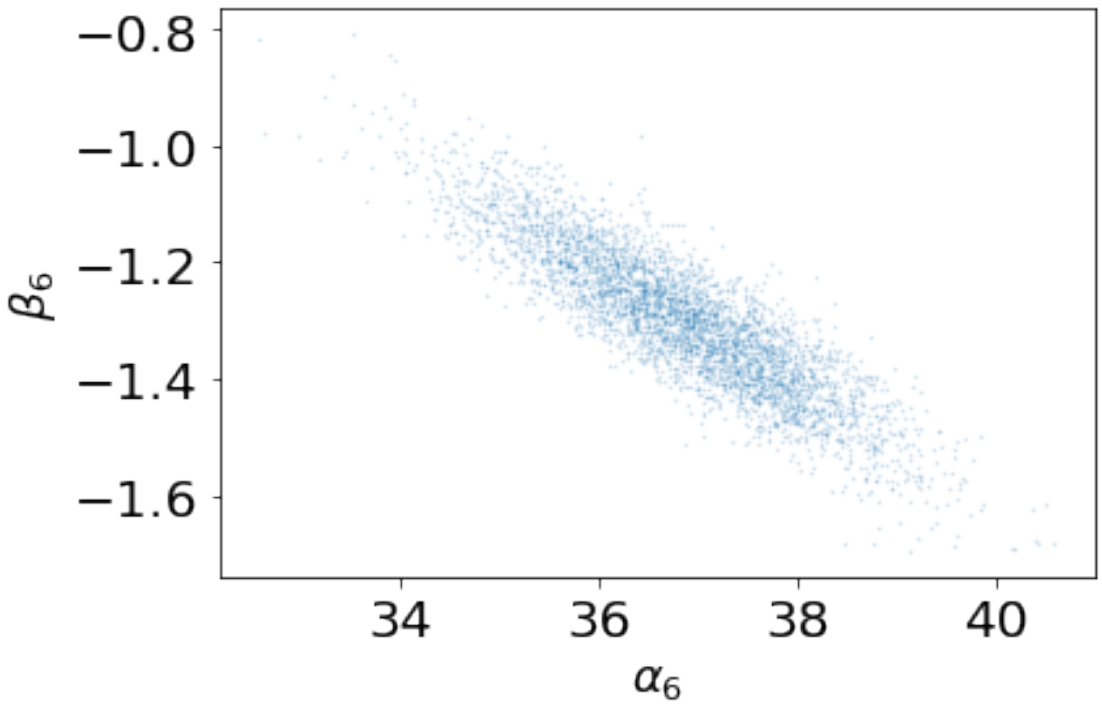



−0.6
−0.8
−1.0
−1.2
−1.4
β_2
32
34
36
38
α_2


−0.8
−1.0
−1.2
−1.4
β_3
34
36
38
40
α_3


−0.8
−1.0
−1.2
−1.4
−1.6
β_4
28
30
32
34
α_4

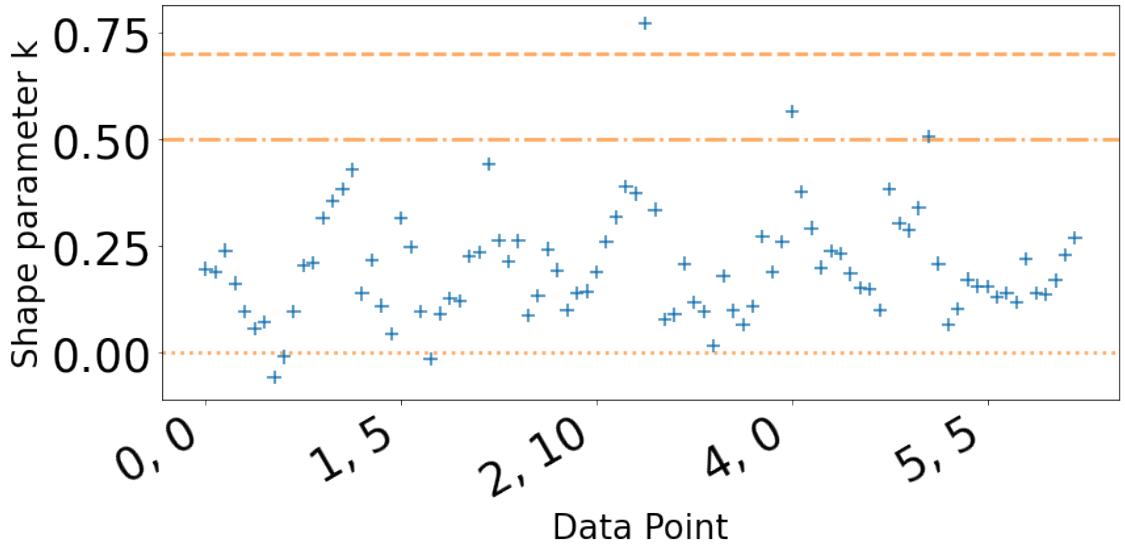

−1.2
−1.4
−1.6
−1.8
β_5
28
30
32
34
α_5



[50]:
```
get_psis_loo_result(hiercov_results)
```

```
Computed from 4000 by 90 log-likelihood matrix

         Estimate       SE
elpd_loo  -196.75     7.00
p_loo       13.36        -

There has been a warning during the calculation. Please check the results.
------

Pareto k diagnostic values:
                         Count   Pct.
(-Inf, 0.5]   (good)       87   96.7%
 (0.5, 0.7]   (ok)          2    2.2%
   (0.7, 1]   (bad)         1    1.1%
   (1, Inf)   (very bad)    0    0.0%

/home/weijiang/anaconda3/envs/bda/lib/python3.7/site-
packages/arviz/stats/stats.py:684: UserWarning: Estimated shape parameter of
Pareto distribution is greater than 0.7 for one or more samples. You should
consider using a more robust model, this is because importance sampling is less
likely to work well if the marginal posterior and LOO posterior are very
different. This is more likely to happen with a non-robust model and highly
```

```
influential observations.
  "Estimated shape parameter of Pareto distribution is greater than 0.7 for "
```



[51]:
```
az.r2_score(accident_data, hiercov_results["yrep"])
```

[51]:
```
r2        0.885874
r2_std    0.005438
dtype: float64
```

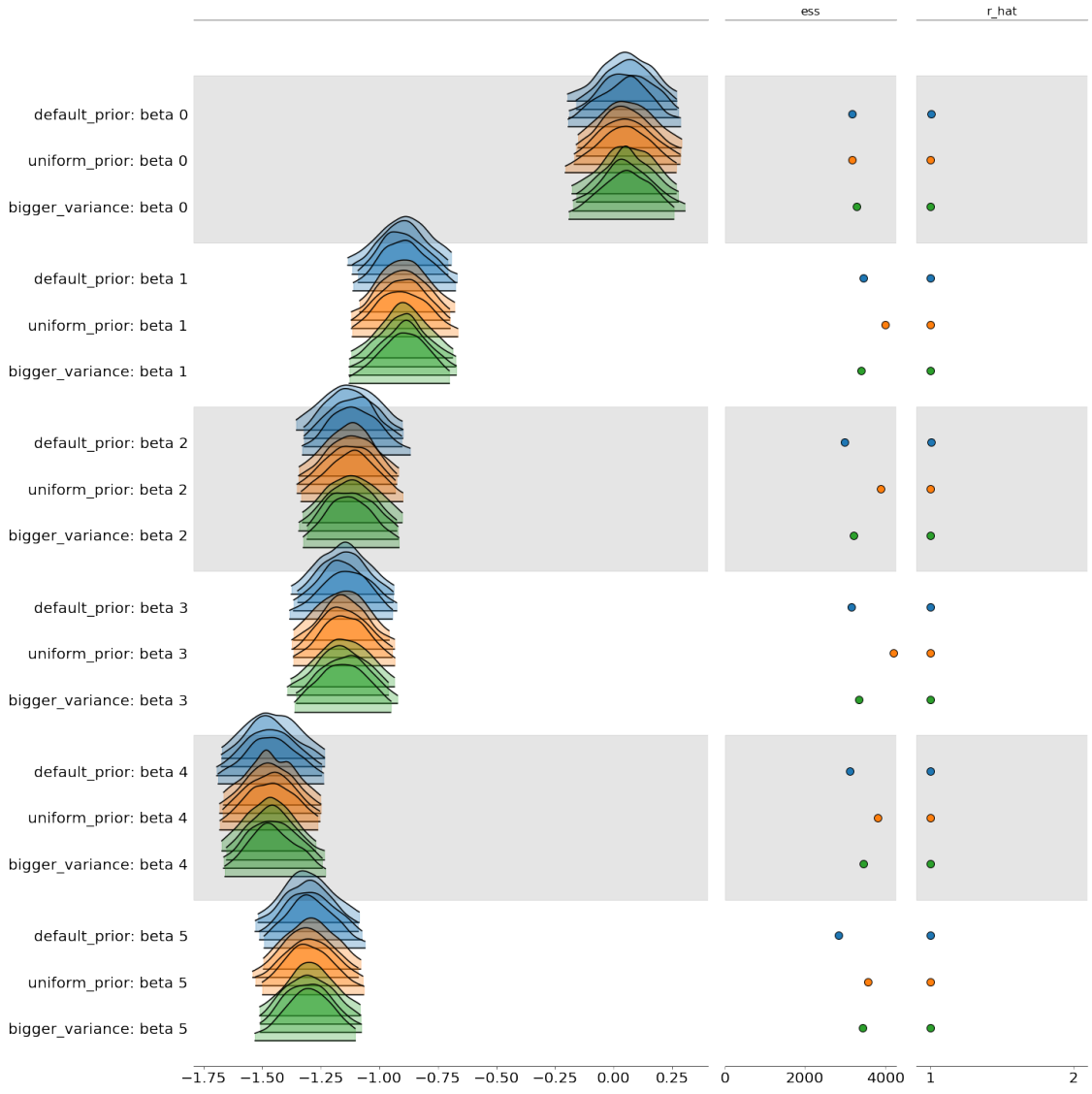
[52]:
```
print(hiercov_results.stansummary(pars=["alpha", "beta", "sigma"]))
```

```
Inference for Stan model: anon_model_c61a7838f3f01dd545a64710e9ee5eac.
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]  26.93    0.02   1.11  24.78  26.21  26.93  27.67   29.1   3294    1.0
alpha[2]  35.05    0.02   1.05  32.97  34.34  35.07  35.77  37.05   3366    1.0
alpha[3]  36.71    0.02    1.1  34.55  35.96  36.69  37.42  38.92   3402    1.0
alpha[4]   30.9    0.02   1.04  28.87  30.19  30.89  31.58  32.99   3614    1.0
alpha[5]  30.31    0.02   1.05   28.3  29.58   30.3  31.01  32.42   2923    1.0
alpha[6]  36.84    0.02   1.05  34.71  36.13  36.86  37.53  38.88   3225    1.0
beta[1]    0.05  2.1e-3   0.12  -0.19  -0.03   0.05   0.13   0.29   3258    1.0
beta[2]    -0.9  2.0e-3   0.12  -1.12  -0.98   -0.9  -0.82  -0.67   3449    1.0
beta[3]   -1.12  2.1e-3   0.12  -1.36   -1.2  -1.12  -1.04  -0.89   3432    1.0
beta[4]   -1.16  1.9e-3   0.11  -1.38  -1.23  -1.16  -1.08  -0.94   3607    1.0
beta[5]   -1.46  2.1e-3   0.12  -1.69  -1.53  -1.46  -1.38  -1.23   2926    1.0
beta[6]    -1.3  2.0e-3   0.12  -1.52  -1.37  -1.29  -1.22  -1.06   3229    1.0
sigma      1.99  2.5e-3   0.17    1.7   1.87   1.98    2.1   2.35   4513    1.0
```

```
Samples were drawn using NUTS at Mon Dec 14 16:01:01 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).
```

[56]:
```
get_plot_forest(hier_cov_model, data_dict)
```

```
Generating results with prior:1 default_prior
Generating results with prior:2 uniform_prior
Generating results with prior:3 bigger_variance
```



[ ]: