BDA - Project

Anonymous

Contents

1. Project Introduction **1**

2. Data Preprocessing and Visualization **2**
2.X Elementary Statistics 2

3. Probability Models **3**
3.1 Separate Model 3
3.2 Pooled Model 6
3.3 Hierarchical Model 8

4. Model Evaluation **10**
4.1 Cross-Validation with PSIS-LOO 11
4.2 Effective Sample Sizes 14
4.3 HMC Convergence Analysis 17
4.4 Posterior Predictive Plot 21
4.5 Prior Sensitivity Test 25

1. Project Introduction

Road traffic and safety have become one of the major problems in people's safety concern. According to WHO, the annual road traffic deaths has reached 1.35 million in 2018, which makes road accident the leading killer of people aged from 5 to 29. In the UK, traffic accidents has caused more than 1700 deaths and more than 150,000 injuries in 2019 alone source. Therefore, understanding and projecting the trend of growth (decrease) about the number of traffic accidents, could raise the awareness of the general population and call for collaborative effort to address this problem.

In this project, we try to explore the `Road Safety Data` from the Department of Transport in the UK. The dataset accurately presents the time, location, police force, vehicles and number of citizens involved in every accident, and it is publicly available at Road Safety Data. We will try to capture the trend of the number of cases in different areas using a normal model with linear mean, and provide statistical results in a Bayesian perspective. Concretely, we study the number of accidents in 6 representative areas: Metropolitan Area, Cumbria, Lancashire, Merseyside, Greater Manchester and Cheshire.

The remaining contents of this report are structured as follows: Section 2 presents the process of data pre-processing and information extraction. It also provides an intuitive overview with the visualization of the elementary statistics. Section 3 introduces and tests the probability models that we choose for this dataset, which includes a separate model, a pooled model and a hierarchical model. Section 4 discusses the fitting results of the three models and evaluates the quality of them based on convergence, cross validation and sensitivity. Finally, Section 5 draws a conclusion for our project and looks into possible methods and outcome of future work. This submission is completed in `python` with `pystan`.

```
import numpy as np
import matplotlib.pyplot as plt
# with out this, plots from matplotlib won't knit on windows
import matplotlib
matplotlib.use('TkAgg')
import pystan
import arviz as az
from pathlib import Path
from matplotlib.patches import Patch
from matplotlib.lines import Line2D

verbose=False

import pystan
print("pystan version:", pystan.__version__)
```

```
## pystan version: 2.19.0.0
```

2. Data Preprocessing and Visualization

2.X Elementary Statistics

```
model_path = './Stan'
data_file = './data/data.txt'
accident_data = np.loadtxt(data_file)
print(accident_data.shape)
```

```
## (6, 15)
```

```
mean_value = np.mean(accident_data) # mean value approximately 25 cases per 10,000 people
# it's very un likely to change 50% of the mean, so 2.57*sigma = mean_value/2
sigma_cand = mean_value / (2*2.57)
sigma_cand
```

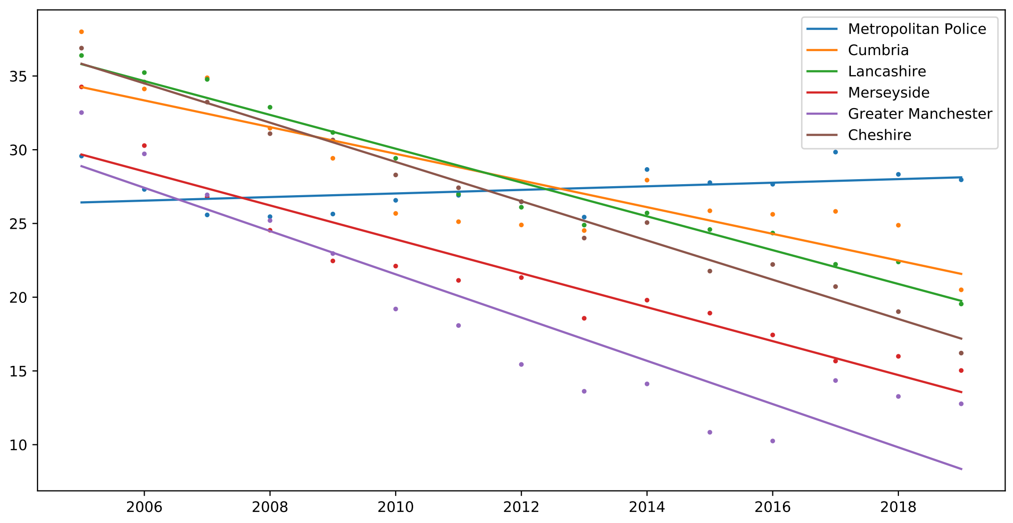
```
## 4.854734111543451
```

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

```
plt.figure(figsize=(12, 6));
years = np.arange(2005, 2020, 1).astype(np.int)
print(years.shape)
```

```
## (15,)
```

```
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.show()
```



3. Probability Models

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:

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

```
separate_model_name = 'accident_separate.stan'
separate_stan_model = pystan.StanModel(file=model_path + '/' + separate_model_name)
```

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

```
print(separate_stan_model.model_code)
```

```
## //
## // 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]);
##     }
##   }
##
## }
```

```python
def test_stan_model(stan_model, data, verbose = False):
    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
    )
    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
```

```python
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]  26.29    0.02   0.87  24.54  25.76  26.29  26.84  28.07   2768    1.0
## alpha[2]  35.14    0.03   1.58  32.02   34.1  35.16  36.15  38.27   2667    1.0
## alpha[3]  36.96    0.01   0.68  35.55  36.54  36.96  37.39  38.29   2572    1.0
## alpha[4]  30.83    0.02   1.18  28.53  30.08  30.84  31.59  33.18   2613    1.0
## alpha[5]  30.34    0.03   1.79  26.72  29.17  30.35  31.49  33.95   2706    1.0
## alpha[6]  37.15 10.0e-3   0.52  36.11  36.81  37.15  37.47  38.22   2763    1.0
## beta[1]    0.12  1.9e-3    0.1  -0.07   0.06   0.12   0.18   0.31   2666    1.0
## beta[2]    -0.9  3.4e-3   0.17  -1.24  -1.02   -0.9  -0.79  -0.57   2645    1.0
## beta[3]   -1.15  1.4e-3   0.08  -1.29  -1.19  -1.15   -1.1  -0.99   2726    1.0
## beta[4]   -1.15  2.5e-3   0.13  -1.42  -1.24  -1.15  -1.07  -0.89   2765    1.0
## beta[5]   -1.47  3.8e-3    0.2  -1.86  -1.59  -1.47  -1.33  -1.08   2699    1.0
## beta[6]   -1.33  1.1e-3   0.06  -1.45  -1.37  -1.33  -1.29  -1.22   2735    1.0
## sigma[1]   1.54  6.7e-3   0.34   1.05    1.3   1.49   1.72   2.37   2601    1.0
## sigma[2]   2.84    0.01   0.65    1.9   2.39   2.75   3.17   4.37   3144    1.0
## sigma[3]   1.25  5.2e-3   0.29   0.83   1.05   1.19    1.4   2.01   3164    1.0
## sigma[4]   2.14  8.3e-3   0.48   1.45   1.81   2.05   2.36   3.31   3272    1.0
## sigma[5]   3.27    0.01   0.71   2.21   2.77   3.17   3.64   4.96   3447    1.0
## sigma[6]   0.93  3.7e-3   0.21   0.62   0.78    0.9   1.04   1.43   3245    1.0
##
## Samples were drawn using NUTS at Sun 29 Nov 2020 11:49:45 PM .
## 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).
```

3.2 Pooled Model

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

```
pooled_model_name = 'accident_pooled.stan'
pooled_stan_model = pystan.StanModel(file=model_path + '/' + pooled_model_name)
```

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

```
print(pooled_stan_model.model_code)
```

```
## //
## // 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);
##     }
##   }
##
## }
```

```
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  32.81    0.03   1.05  30.74  32.12  32.83  33.53  34.84   1708    1.0
## beta   -0.98  2.8e-3   0.11  -1.21  -1.06  -0.98  -0.91  -0.75   1742    1.0
## sigma   4.71  8.4e-3   0.36   4.06   4.46   4.68   4.94   5.45   1834    1.0
##
## Samples were drawn using NUTS at Sun 29 Nov 2020 11:50:47 PM .
## 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).
```

3.3 Hierarchical Model

```
hier_model_name = 'accident_hierarchical.stan'
hier_stan_model = pystan.StanModel(file=model_path + '/' + hier_model_name)
```

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

```
print(hier_stan_model.model_code)
```

```
## //
## // 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);
##     }
##   }
##
## }
```

```
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]  26.93    0.02    1.1  24.79  26.21  26.91  27.67  29.12   3216    1.0
## alpha[2]  35.02    0.02   1.05  32.91  34.34  35.02  35.73  37.05   3322    1.0
## alpha[3]  36.71    0.02   1.05  34.65  36.02  36.68  37.42  38.82   3322    1.0
## alpha[4]  30.88    0.02   1.01   28.9  30.18  30.87  31.56  32.87   3646    1.0
## alpha[5]  30.31    0.02   1.06  28.23  29.61  30.31   31.0  32.37   2794    1.0
## alpha[6]  36.82    0.02   1.05  34.71  36.12  36.81  37.53  38.84   3728    1.0
## beta[1]    0.05  2.1e-3   0.12  -0.18  -0.03   0.05   0.13   0.29   3172    1.0
## beta[2]   -0.89  2.0e-3   0.12  -1.12  -0.97  -0.89  -0.82  -0.66   3394    1.0
## beta[3]   -1.12  2.0e-3   0.11  -1.35   -1.2  -1.12  -1.05  -0.89   3359    1.0
## beta[4]   -1.15  1.9e-3   0.11  -1.37  -1.23  -1.15  -1.08  -0.93   3576    1.0
## beta[5]   -1.46  2.1e-3   0.12  -1.69  -1.54  -1.46  -1.38  -1.22   3225    1.0
## beta[6]    -1.3  1.9e-3   0.11  -1.51  -1.37   -1.3  -1.22  -1.07   3825    1.0
## sigma      1.98  2.4e-3   0.17    1.7   1.87   1.97   2.09   2.35   4825    1.0
##
## Samples were drawn using NUTS at Sun 29 Nov 2020 11:51:52 PM .
## 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).
```

4. Model Evaluation

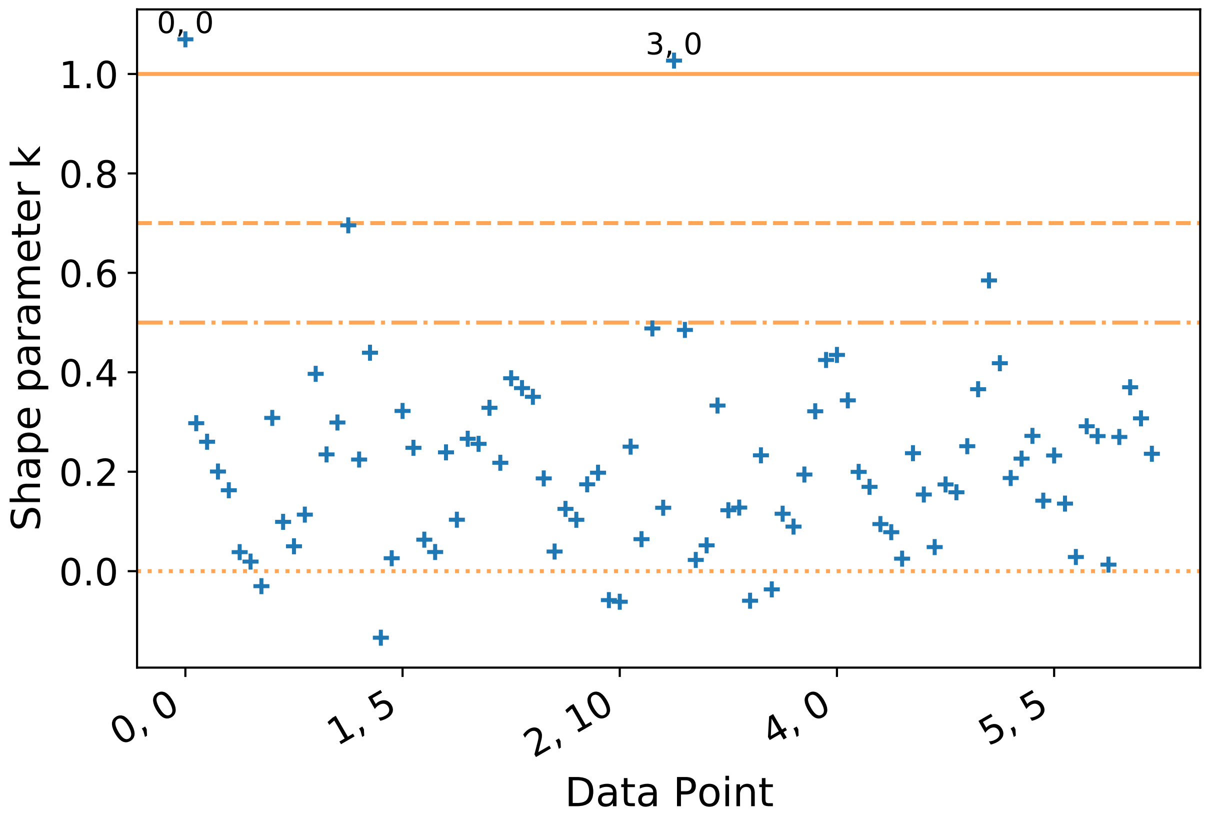
```
var_separate =["alpha", "beta", "sigma"] # the variables that need to be plotted
var_pooled = ["alpha", "beta", "sigma"]
var_hier = ["alpha", "beta", "mu_alpha", "sigma_alpha", "mu_beta", "sigma_beta"]
```

4.1 Cross-Validation with PSIS-LOO

```
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)
    plt.show()
```

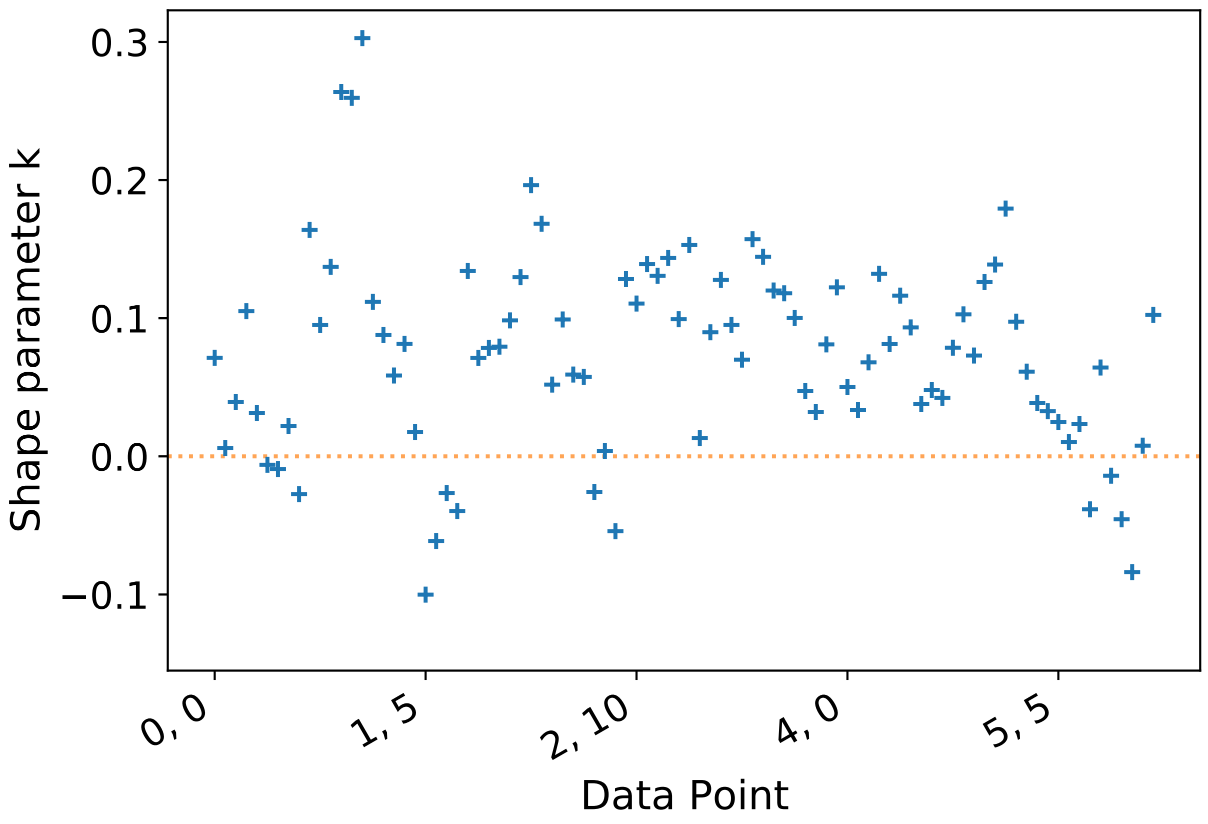
```
get_psis_loo_result(separate_results)
```

```
## Computed from 4000 by 90 log-likelihood matrix
##
##           Estimate       SE
## elpd_loo  -186.32     7.82
## p_loo       16.20        -
##
## 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)         0    0.0%
##    (1, Inf)   (very bad)    2    2.2%
##
##
## /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 "
```



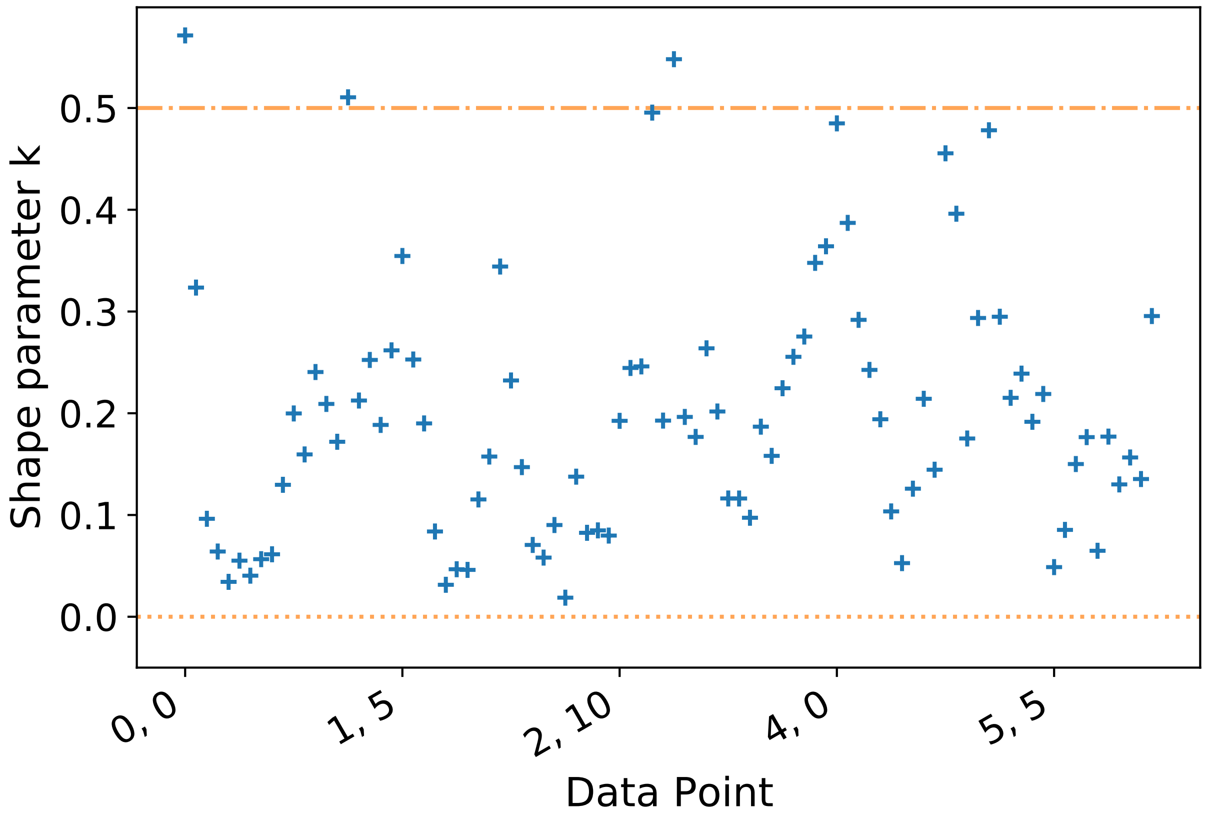
```
get_psis_loo_result(pooled_results)
```

```
## Computed from 4000 by 90 log-likelihood matrix
##
##           Estimate       SE
## elpd_loo   -267.77     6.33
## 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%
```



```
get_psis_loo_result(hier_results)
```

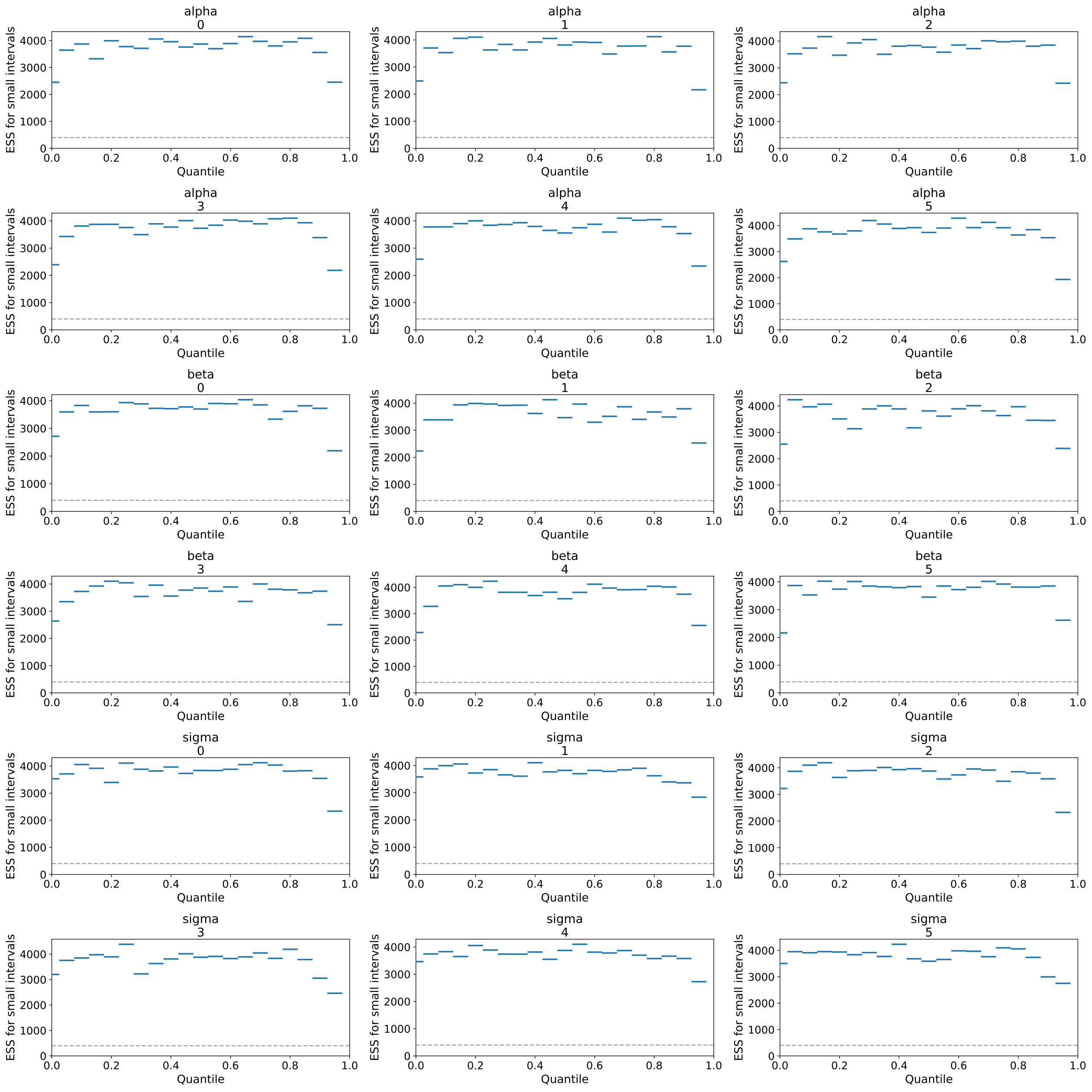
```
## Computed from 4000 by 90 log-likelihood matrix
##
##          Estimate       SE
## elpd_loo  -196.39     6.92
## p_loo       13.02        -
## ------
##
## Pareto k diagnostic values:
##                          Count   Pct.
## (-Inf, 0.5]   (good)       87   96.7%
##  (0.5, 0.7]   (ok)          3    3.3%
##    (0.7, 1]   (bad)         0    0.0%
##    (1, Inf)   (very bad)    0    0.0%
```



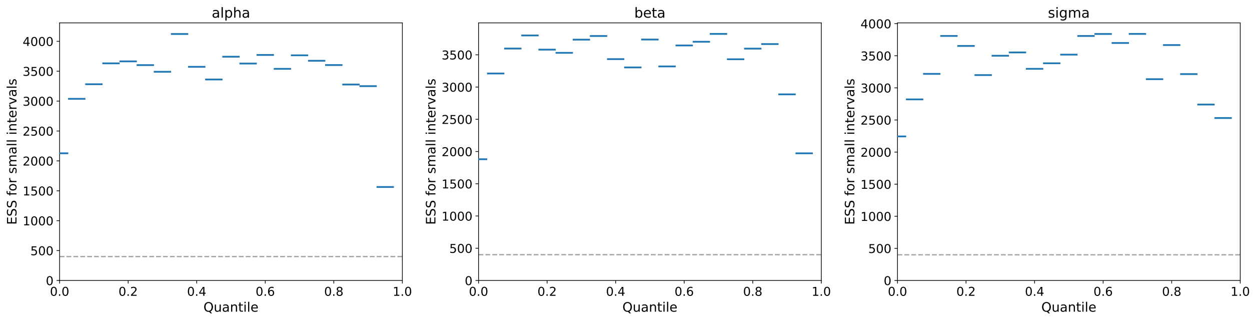
4.2 Effective Sample Sizes

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

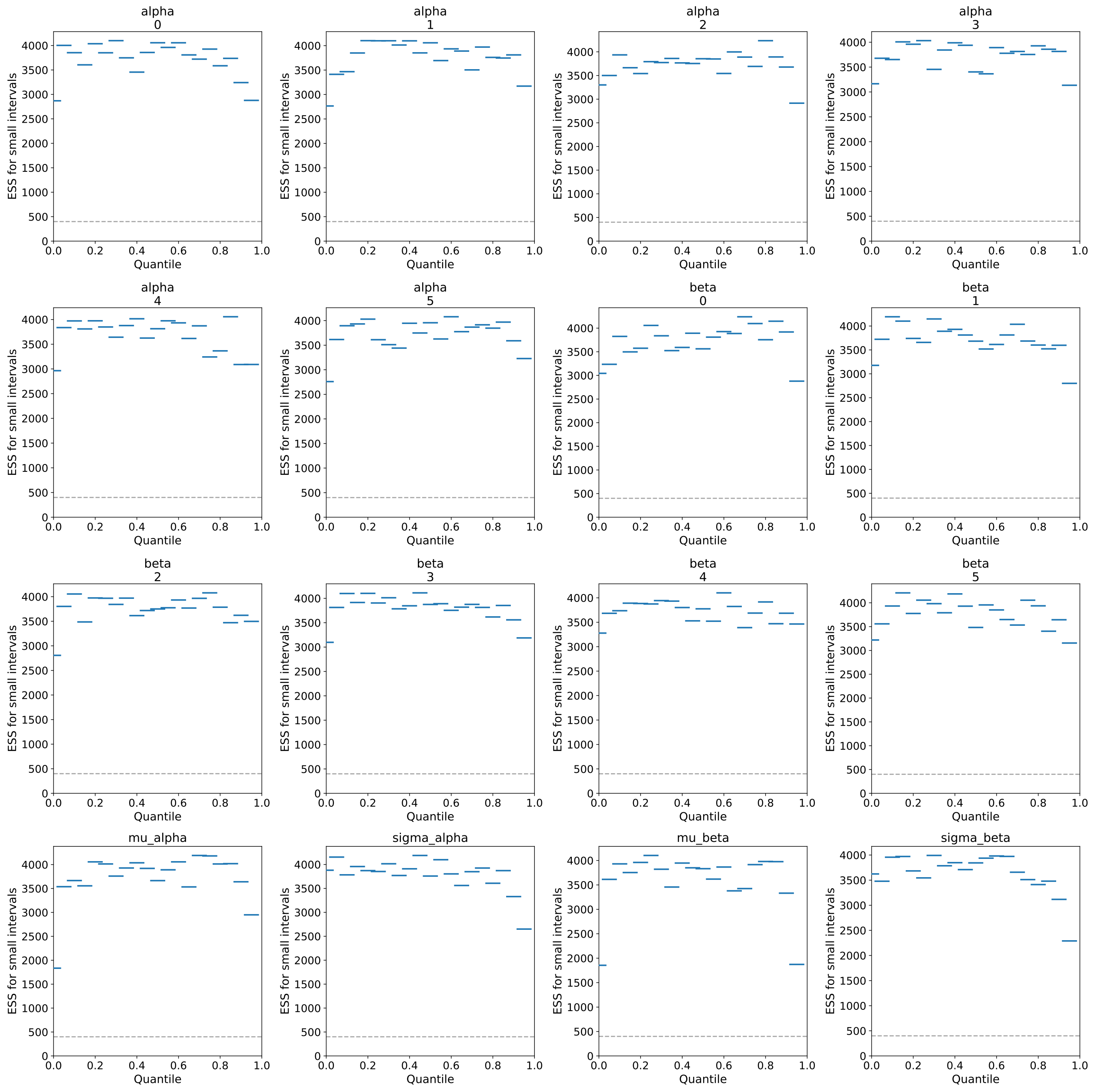
```
_ = az.plot_ess(
    separate_results, var_names=var_separate,
    kind="local", marker="_", ms=20, mew=2, figsize=(20, 20)
)
plt.show()
```



```python
_ = az.plot_ess(
    pooled_results, var_names=var_pooled,
    kind="local", marker="_", ms=20, mew=2, figsize=(20, 5)
)
plt.show()
```

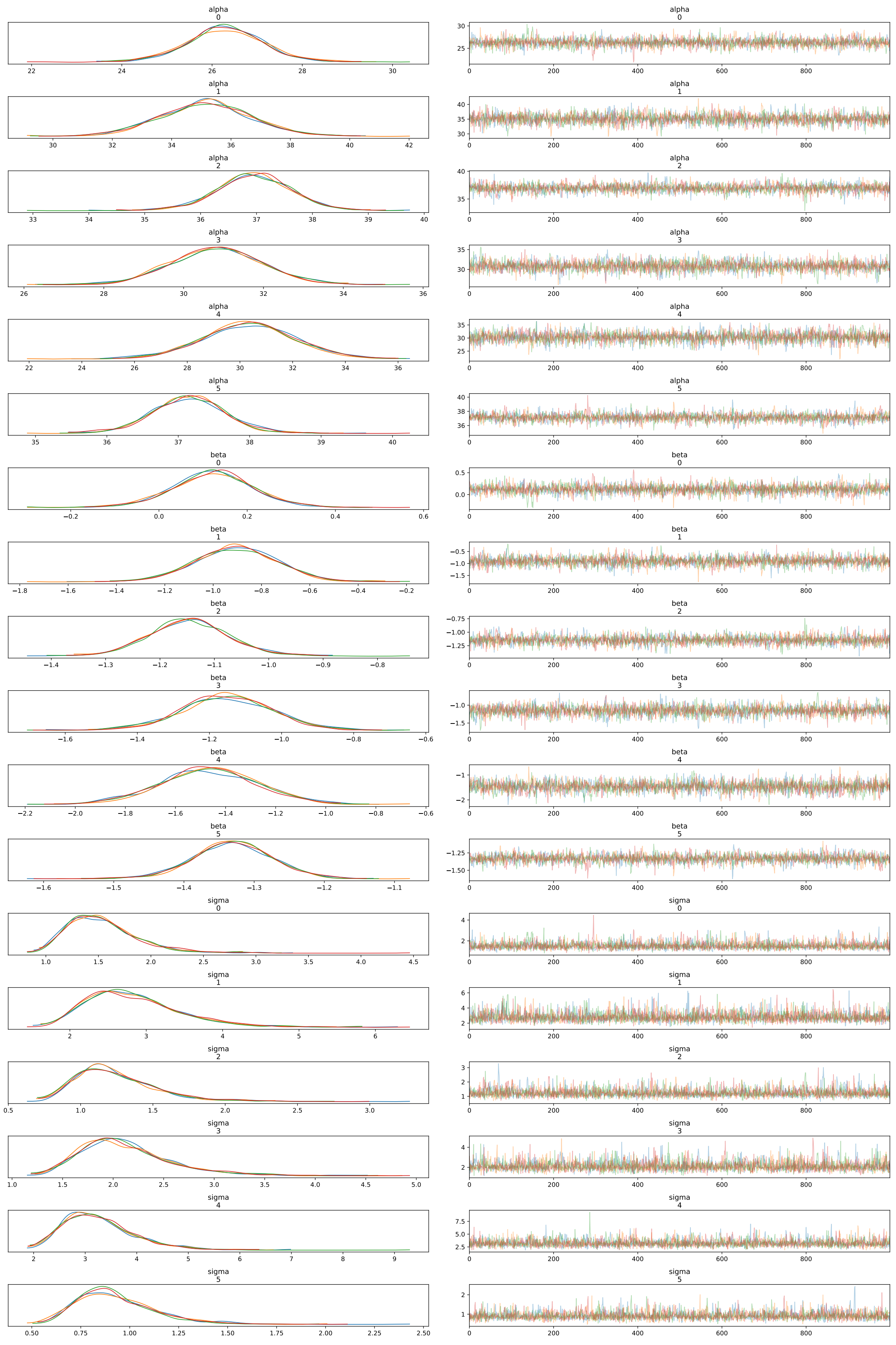


```
_ = az.plot_ess(
    hier_results, var_names=var_hier,
    kind="local", marker="_", ms=20, mew=2, figsize=(20, 20)
)
plt.show()
```

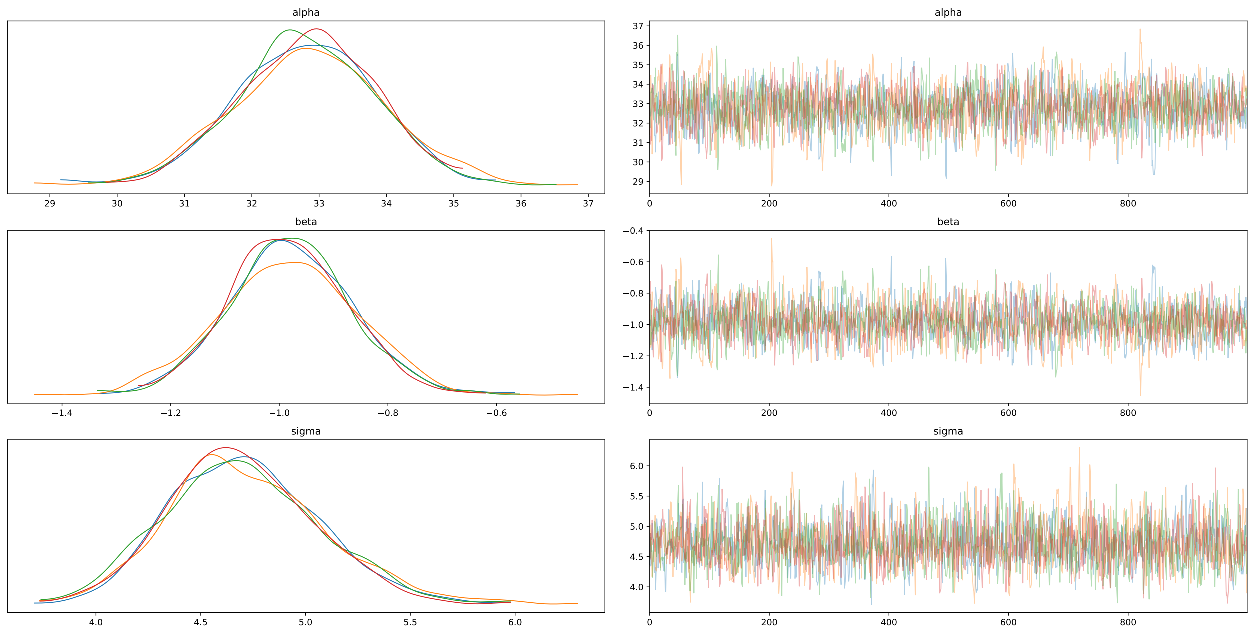


4.3 HMC Convergence Analysis

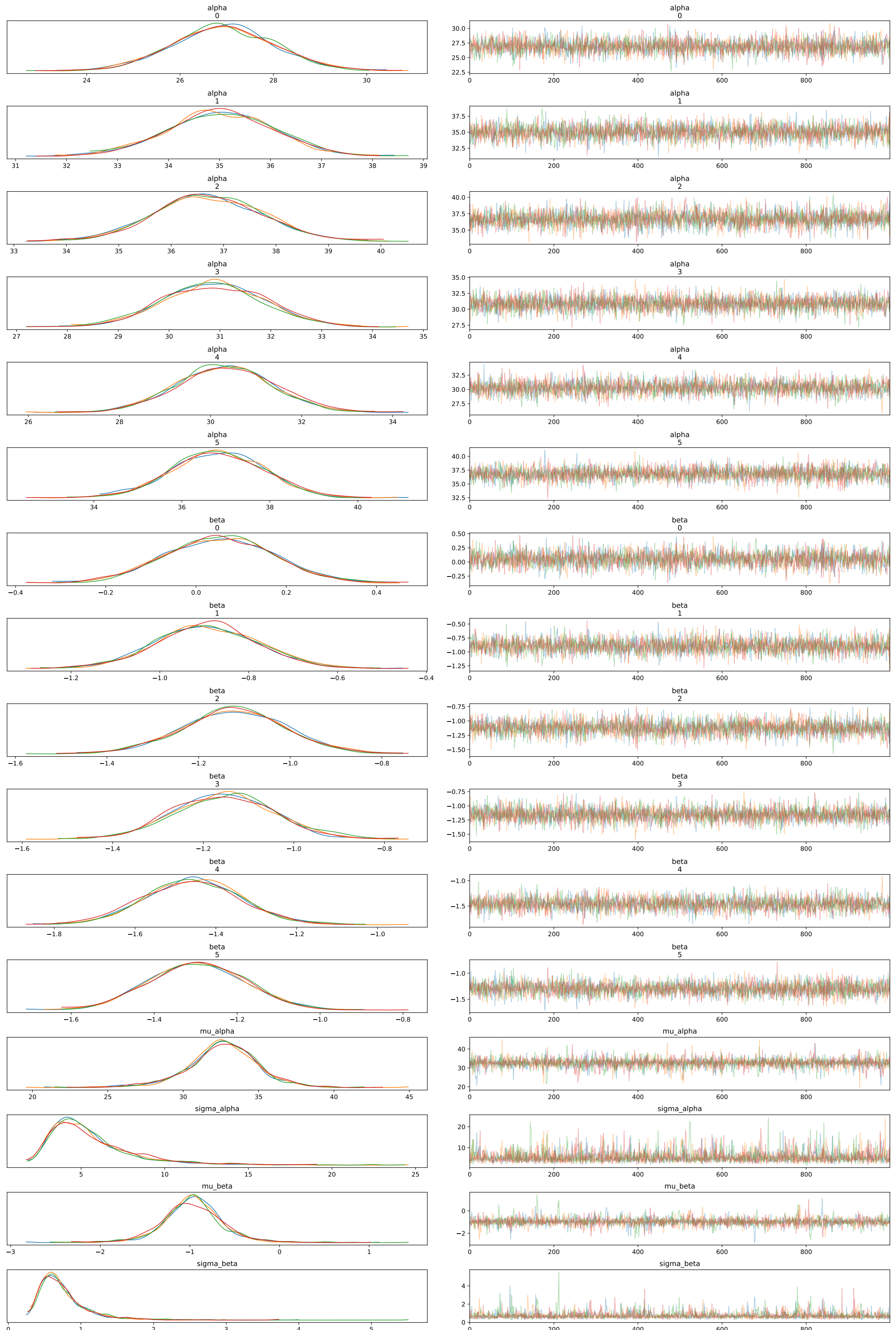
```
_ = az.plot_trace(separate_results, var_names = var_separate, figsize=(20, 30))
plt.show()
```


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

```
_ = az.plot_trace(pooled_results, var_names = var_pooled, figsize=(20, 10))
plt.show()
```



```
_ = az.plot_trace(hier_results, var_names = var_hier, figsize=(20, 30))
plt.show()
```


alpha 0
alpha 1
alpha 2
alpha 3
alpha 4
alpha 5
beta 0
beta 1
beta 2
beta 3
beta 4
beta 5
mu_alpha
sigma_alpha
mu_beta
sigma_beta

```python
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('divergent transitions: ', any(h['divergent__']))
```

```python
get_treedepth(separate_results)
```

```
## max treedepth for draws:  6
## min treedepth for draws:  2
## mean treedepth for draws:  4.19425
## divergent transitions:  False
```

```python
get_treedepth(pooled_results)
```

```
## max treedepth for draws:  4
## min treedepth for draws:  1
## mean treedepth for draws:  2.8985
## divergent transitions:  False
```

```python
get_treedepth(hier_results)
```

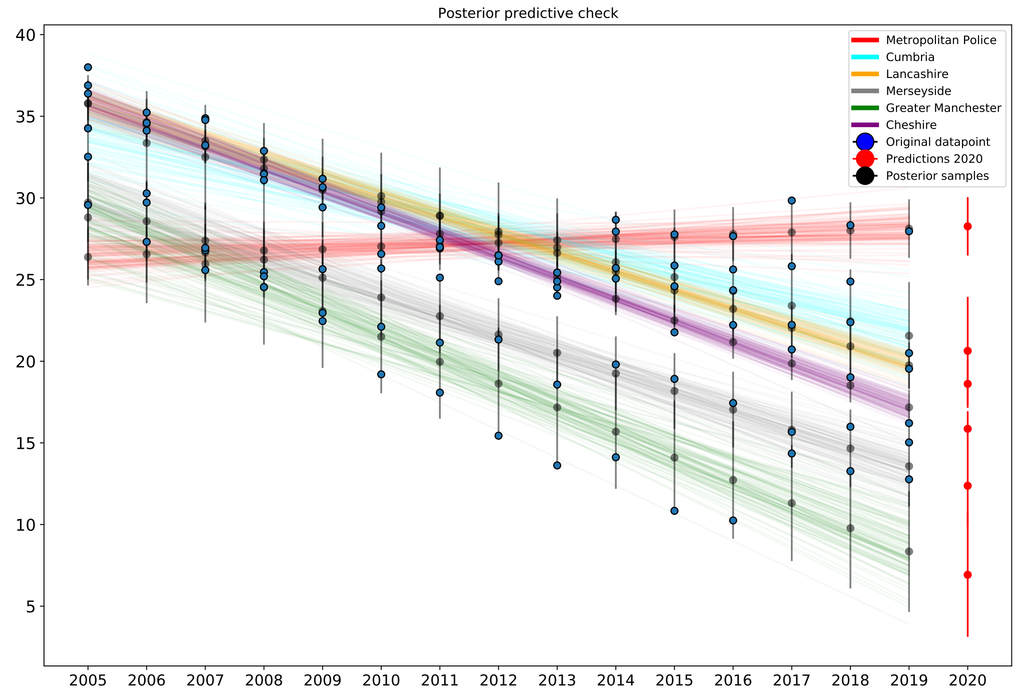
```
## max treedepth for draws:  6
## min treedepth for draws:  2
## mean treedepth for draws:  3.9555
## divergent transitions:  False
```

4.4 Posterior Predictive Plot

```python
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
    _ = 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', ma
        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)
    plt.show()
```

```
plot_posterior_draws(separate_results, accident_data)
```

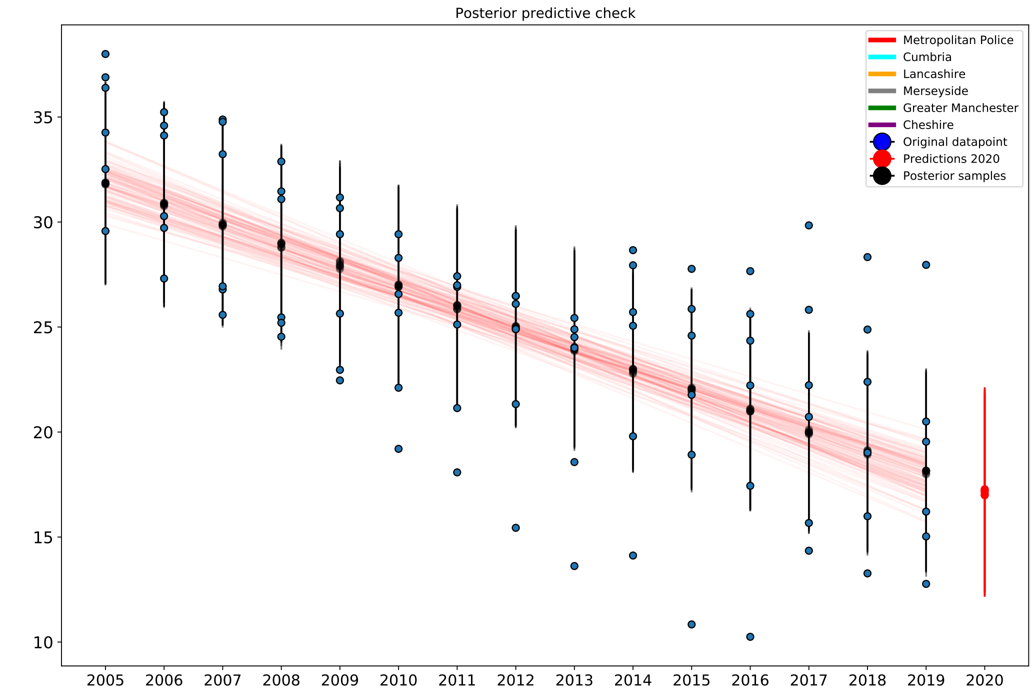
```
## WARNING:matplotlib.legend:No handles with labels found to put in legend.
```



```
plot_posterior_draws(pooled_results, accident_data, pooled=True)
```

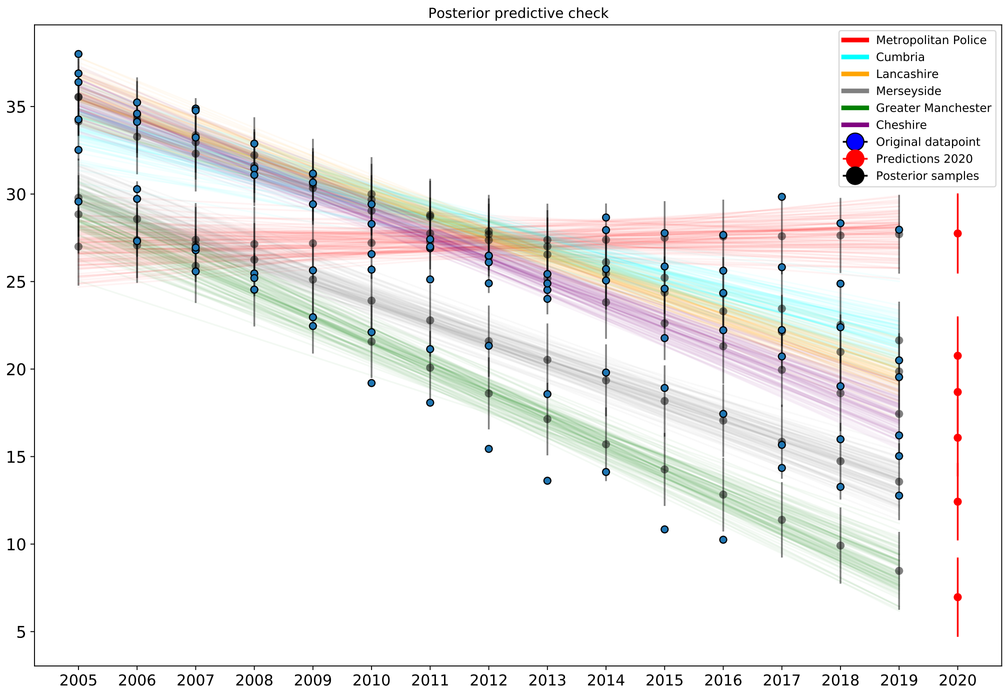
```
## WARNING:matplotlib.legend:No handles with labels found to put in legend.
```

Posterior predictive check



```
plot_posterior_draws(hier_results, accident_data)
```

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



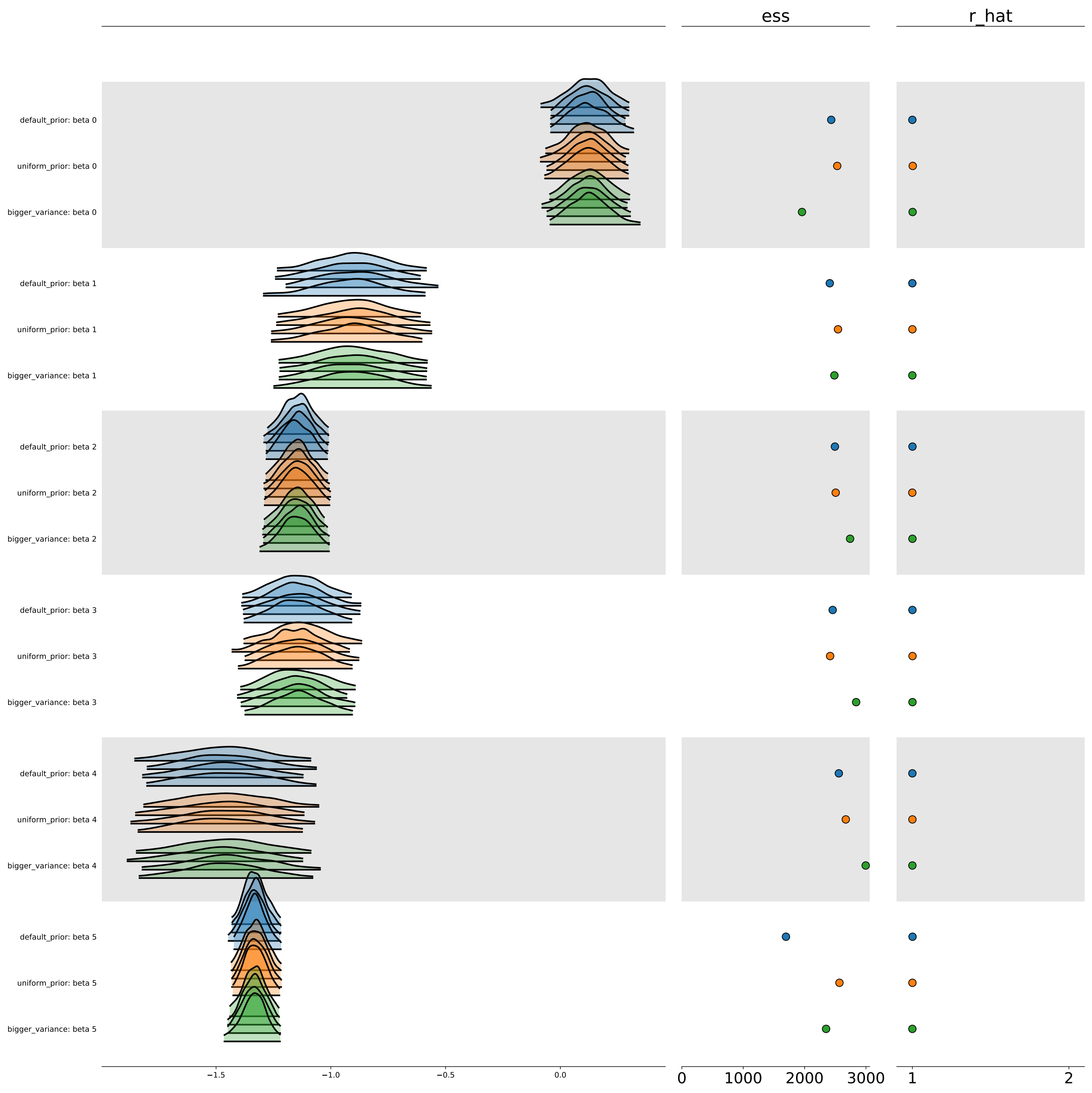
4.5 Prior Sensitivity Test

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

```python
def get_plot_forest(stan_model, data_dict, pooled=False):
    if pooled:
        figsize = (20, 20)
    else:
        figsize = (20, 5)
    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=(20, 20), textsize=20)
    plt.rcParams['xtick.labelsize'] = 20
    plt.rcParams['ytick.labelsize'] = 20
    plt.show()
```

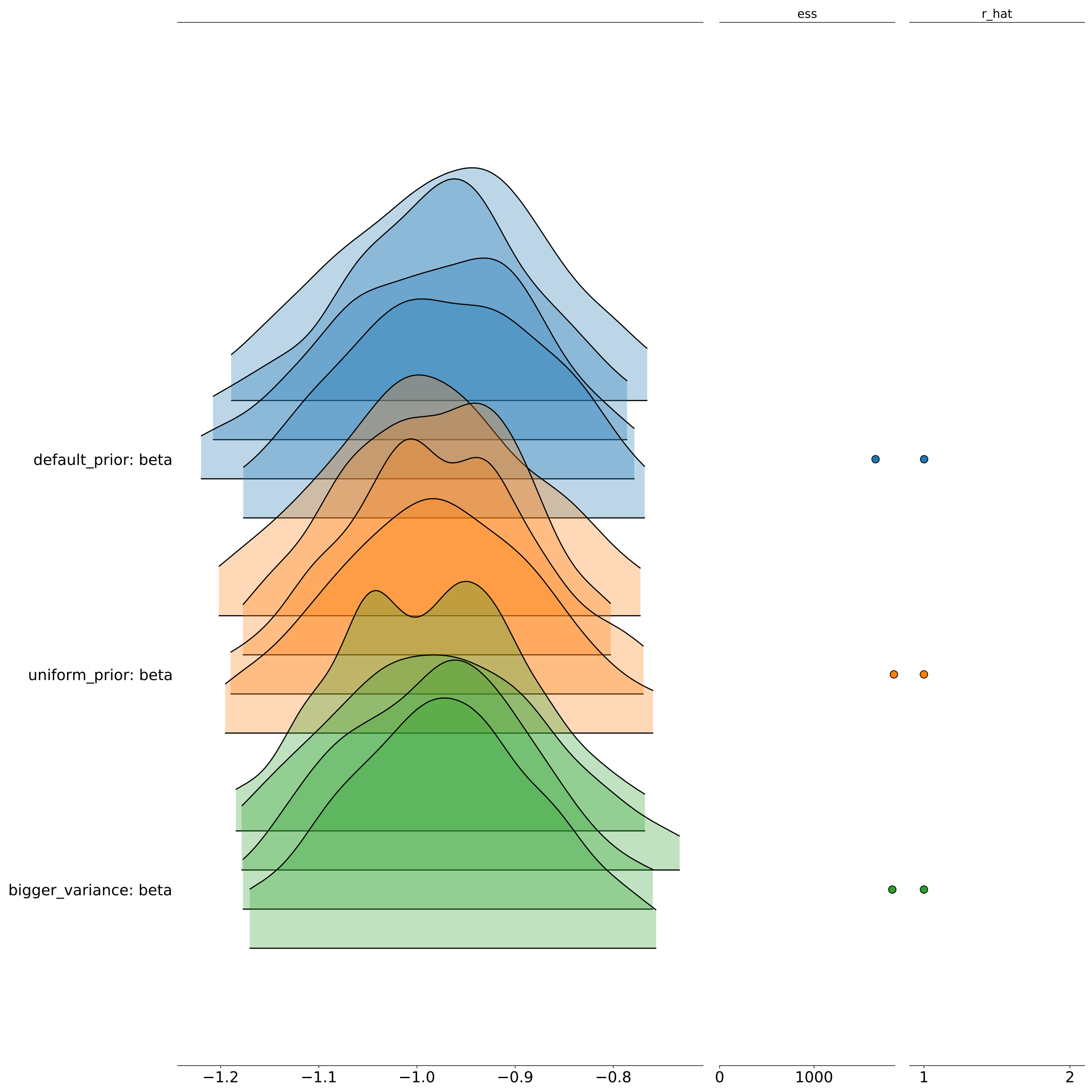
```
get_plot_forest(separate_stan_model, data_dict)
```

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



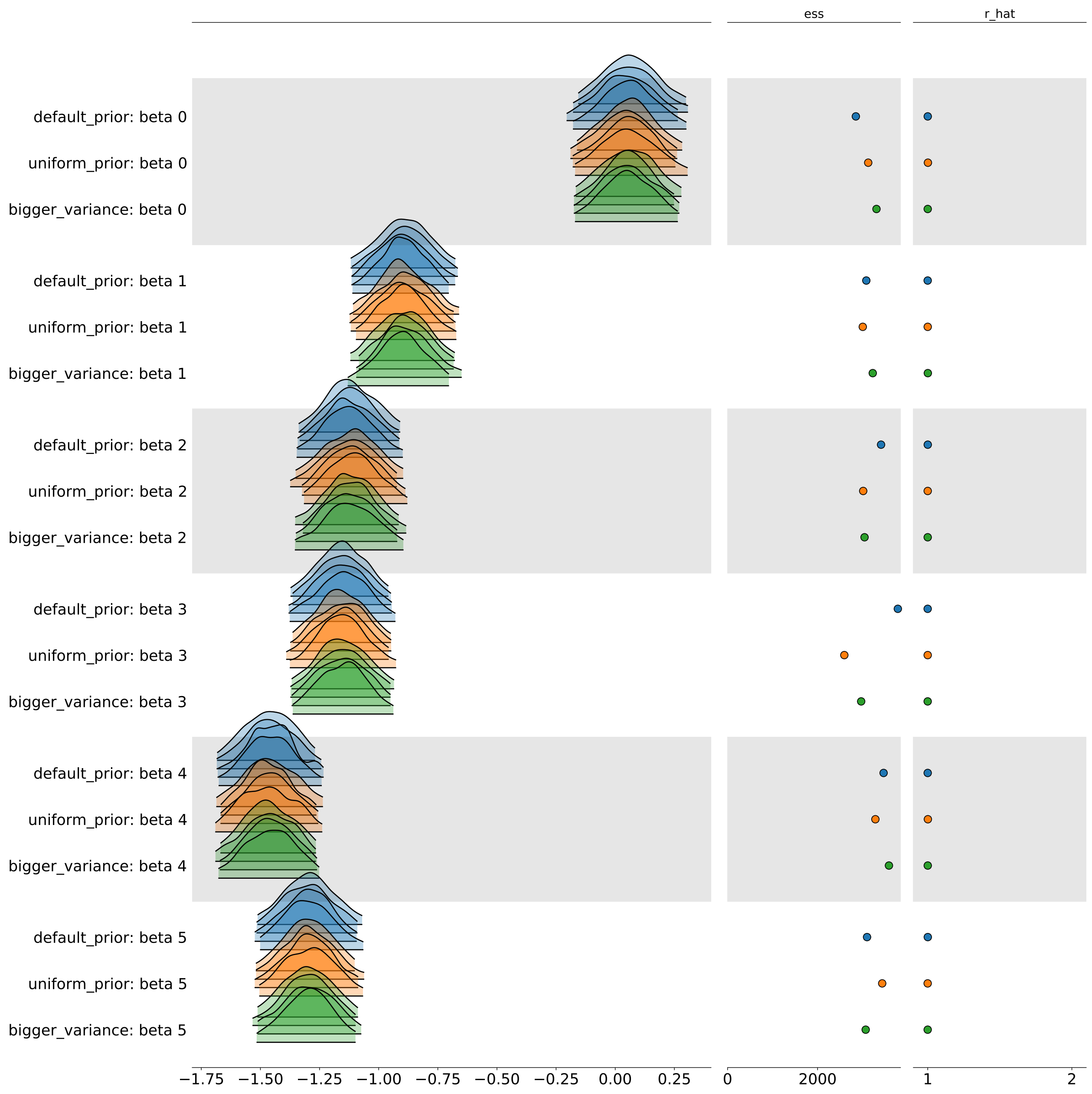
```
get_plot_forest(pooled_stan_model, data_dict, pooled=True)
```

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



```
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
```


ess
r_hat
default_prior: beta 0
uniform_prior: beta 0
bigger_variance: beta 0
default_prior: beta 1
uniform_prior: beta 1
bigger_variance: beta 1
default_prior: beta 2
uniform_prior: beta 2
bigger_variance: beta 2
default_prior: beta 3
uniform_prior: beta 3
bigger_variance: beta 3
default_prior: beta 4
uniform_prior: beta 4
bigger_variance: beta 4
default_prior: beta 5
uniform_prior: beta 5
bigger_variance: beta 5
−1.75 −1.50 −1.25 −1.00 −0.75 −0.50 −0.25 0.00 0.25
0 2000
1 2