BDA - Project

Anonymous

Contents

1. Project Introduction

Road traffic and safety have become one of the major problems in people's safety concerns. According to 1, 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 have caused more than 1700 deaths and more than 150,000 injuries in 2019 alone 2. Therefore, understanding and projecting the trend of growth (decrease) in the rate of traffic accidents, could raise the awareness of the general population and call for a 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 3. 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 from a Bayesian perspective. Concretely, we study the number of accidents in 6 areas: Metropolitan Area (London), 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 of the visualization of 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 performance based on convergence, cross-validation, posterior prediction and sensitivity. Finally, Section 5 draws a conclusion for our project and looks into possible methods and future work. This submission is completed in `python` with `pystan`.

```python
import numpy as np
import pandas as pd
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 Description

2.1 Data Preprocess

As the first section described, Road Safety Data contains millions of car accident records from 2005 to 2019 in the UK. Those records contain much information irrelevant to our goals such as the number of vehicles, speed limit, light conditions and many other factors. In addition, the records provide no information on the number of accidents per year directly. Fortunately, it is possible to obtain our interested data by counting car accident records in which dates are the same year. However, there is an extensive difference in the number of car accidents which mainly results from the tremendous difference in population. This difference hinders fair comparison of traffic management systems in different areas. To eliminate this negative effect, we perform data normalization, which is dividing the number of car accidents by the population in each area. Although Road Safety Data does not provide population information, we eventually find `public population information` via UK parliament.

To sum up:

1. Extract two columns from the original data, including "date" and "police force". Date shows the date of one accident and the police force indicates the responsible police office of the accident spot.
2. Count the number of records per police force area in each year(2005-2019).
3. Calculate accident rates per 10,000 people.

2.2 Data Visualization

After preprocessing the raw data, we could perform some elementary statistical analysis to obtain an intuitive grasp of the data. The data here summarizes the number of accidents in 6 areas from 2005 to 2019 (15 years in total), therefore the processed data have size (6,15).

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

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

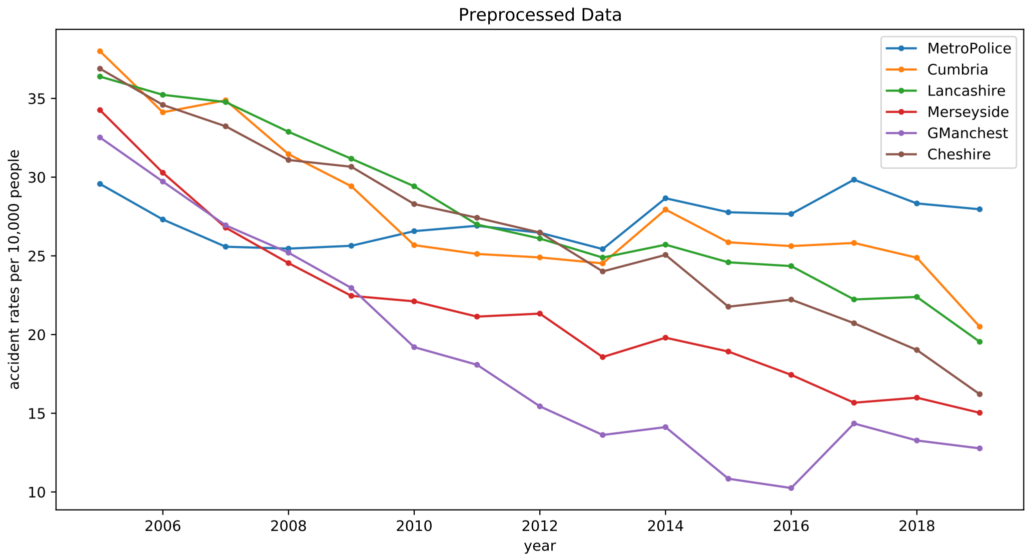
The processed data are shown as follows:

```
area_names = ["MetroPolice", 'Cumbria','Lancashire',
              'Merseyside','GManchest','Cheshire']
# region population 2020 [8961989,500012,1508941,1429910,2835686,1066647]
years = np.arange(2005, 2020, 1).astype(np.int)
df = pd.DataFrame(accident_data, index=area_names, columns=years)
# print(df.to_markdown())
```

Area	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019
MetroPolice	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
Cumbria	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
Lancashire	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
Merseyside	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
GManchest	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
Cheshire	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

We could also plot these data points, and from this figure, we could see that the trends in some areas are not very close to linear variation.

```
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()
```



The following elementary statistics could also help us gain some insights into the data. Greater Manchester has the lowest mean accident rate, while Cumbria has the highest mean. Notably, the average accident rates in Metropolitan Police and Lancashire are fairly close to the highest.

```
# print(df.T.describe().round(2).to_markdown())
```

	MetroPolice	Cumbria	Lancashire	Merseyside	GManchest	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

Also, we could calculate the mean accident rate in each area, which turns out to be approximately 25. This mean vlaue also provides justification for our choice of variance in the probability model (Section 3).

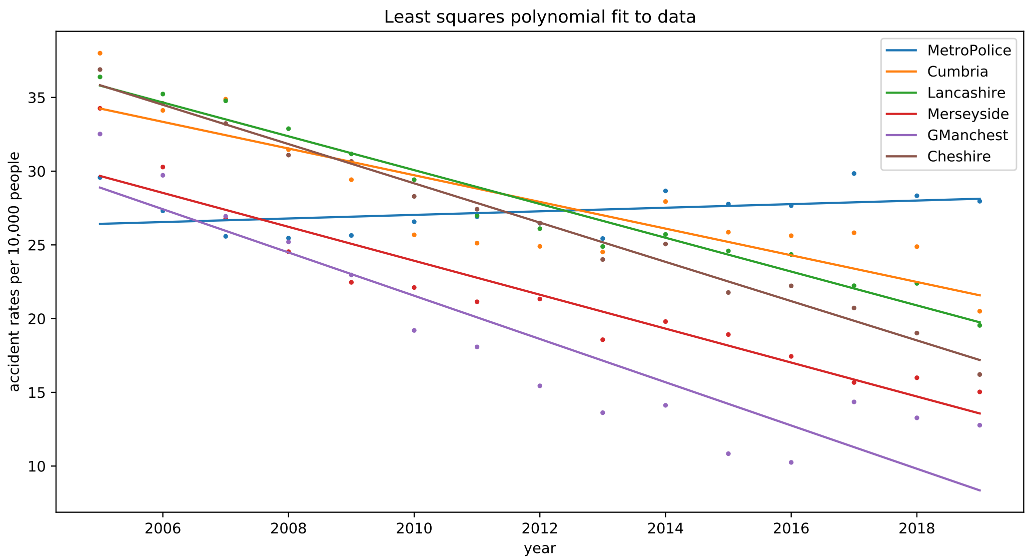
```
# mean value approximately 25 cases per 10,000 people
mean_value = np.mean(accident_data)
mean_value
```

```
## 24.953333333333337
```

Then, we fit a simple linear model with the least mean square method to the data points. From the results, we could see that the number of cases in the Metropolitan Police area has been increasing slowly, while the cases in the other five police areas have been decreasing with similar trends.

```
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()
```



3. Probability Models

For this dataset, we use normal probability models with linear mean, which could be generally written as:

$$accidents \sim Normal(\alpha + \beta * year, \sigma) \tag{1}$$

Intuitively, it is very unlikely that the accident rate change by 50% of the mean (approximately 25), so we could choose the variance of the slope β in the linear mean. From a standard normal distribution table, we've found that -2.57 corresponds to 0.5 in the normal CDF, therefore the interval $[-2.57\sigma, 2.57\sigma]$ contains 99% of the probability mass. Based on these assumptions, we could use $2.57\sigma = mean/2$ to find a candidate for variance of the slope, and the results is about 4.85.

```
# 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
```

Then, we build separate, pooled and hierarchical model to fit our data.

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.

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

Both the intercept and slope are modeled with a normal prior. From the linear least mean square fitting, we observe that the mean values are around 25, therefore we set the mean value of α's prior to a number nearby (30). Besides, we've kept the variance large to provide relatively weakly informative prior, because we observed large differences among the areas. The prior for β centers around zero and has a variance at 4.85 as discussed above.

Here we load the stan model and display its implementation. In our models, we set 2005 to year 1 since we don't have data before that year. Further, we implemented three sets of priors for sensitivity analysis (Section 4.5) and prepared log likelihood for cross validation.

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

```
##   }
##
## }
```

We define a common function to test the three different stan models, and for simplicity we just report the estimated values of the key parameters α, β and σ, which are directly used to generate data points in our model.

```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
```

From the summary by stan, we could see that the accident rate in Metropolitan Area has a slowly-increasing trend in these years, but all the other five areas have witnessed a decrease in the accident rate. In this part, the fitted model agrees with our intuition and elementary statistical 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]   26.32    0.02   0.82  24.69   25.8  26.32  26.82  27.94   2816    1.0
## alpha[2]   35.12    0.03   1.55  31.95  34.15  35.11  36.11  38.19   2695    1.0
## alpha[3]   36.94    0.01   0.69  35.51  36.52  36.95  37.38  38.28   2560    1.0
## alpha[4]   30.81    0.03   1.21   28.4  30.06   30.8  31.57  33.18   2327    1.0
## alpha[5]   30.37    0.03   1.77   26.9  29.24  30.36  31.52  33.91   2586    1.0
## alpha[6]   37.14    0.01   0.53   36.1   36.8  37.15  37.48  38.19   2352    1.0
## beta[1]     0.12  1.8e-3   0.09  -0.06   0.06   0.12   0.18    0.3   2593    1.0
## beta[2]     -0.9  3.3e-3   0.17  -1.24  -1.01   -0.9  -0.79  -0.57   2706    1.0
## beta[3]    -1.15  1.5e-3   0.08  -1.29   -1.2  -1.15   -1.1  -0.99   2642    1.0
## beta[4]    -1.15  2.7e-3   0.13  -1.41  -1.23  -1.15  -1.06  -0.89   2400    1.0
## beta[5]    -1.47  3.7e-3   0.19  -1.86  -1.59  -1.47  -1.34  -1.09   2754    1.0
## beta[6]    -1.33  1.2e-3   0.06  -1.44  -1.36  -1.33  -1.29  -1.22   2323    1.0
## sigma[1]    1.55  6.4e-3   0.35   1.04   1.31    1.5   1.74    2.4   3036    1.0
## sigma[2]    2.85    0.01   0.63   1.93   2.41   2.74   3.19   4.34   3409    1.0
## sigma[3]    1.24  4.9e-3   0.27   0.83   1.05   1.19   1.37   1.92   3086    1.0
## sigma[4]    2.15  8.7e-3   0.49   1.43   1.81   2.07   2.39   3.34   3237    1.0
## sigma[5]    3.28    0.01   0.74    2.2   2.76   3.16   3.66   4.99   2993    1.0
## sigma[6]    0.93  4.3e-3   0.21   0.63   0.78   0.89   1.04   1.46   2480    1.0
##
## Samples were drawn using NUTS at Tue 01 Dec 2020 11:44:56 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

In the pooled model, all 6 areas obey the same normal distribution. The mathematical expression of this model can be specified by the following equations:

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

Note that for a fair comparison of the models, we use the same prior as the separate model here.

Again, we prepared three sets of priors for sensitivity checking.

```python
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.
```

```python
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);
##     }
##   }
##
## }
```

The summary for the pooled model is shown below. From the results, we could see that the value for α and β is around the mean of the multiple α_i and β_i for the separate model. However, the variance is larger than the separate model, since it has to cover different scenarios in all areas.

```
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   1.05  30.76  32.09  32.79   33.5  34.86   1672    1.0
## beta   -0.98  2.8e-3   0.12  -1.21  -1.06  -0.98   -0.9  -0.75   1699    1.0
## sigma   4.71  8.1e-3   0.36   4.06   4.46   4.68   4.95   5.49   2016    1.0
##
## Samples were drawn using NUTS at Tue 01 Dec 2020 11:46:12 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]   26.96    0.02 1.09 24.86 26.22 26.95 27.68 29.15  2918   1.0
## alpha[2]   35.06    0.02 1.07 33.02 34.33 35.06  35.8 37.13  3372   1.0
## alpha[3]   36.66    0.02 1.06 34.63 35.94 36.65 37.36 38.78  3254   1.0
## alpha[4]   30.91    0.02 1.04 28.86 30.21 30.92  31.6 33.01  3364   1.0
## alpha[5]   30.36    0.02 1.05 28.33 29.66 30.37 31.08 32.39  3664   1.0
## alpha[6]   36.84    0.02 1.06 34.77 36.12 36.83 37.54 38.96  3890   1.0
## beta[1]     0.05 2.3e-3 0.12 -0.19 -0.04  0.05  0.13  0.29  2909   1.0
## beta[2]     -0.9 2.1e-3 0.12 -1.12 -0.98  -0.9 -0.82 -0.67  3227   1.0
## beta[3]    -1.12 2.0e-3 0.11 -1.34 -1.19 -1.12 -1.04 -0.89  3191   1.0
## beta[4]    -1.16 2.0e-3 0.12 -1.39 -1.23 -1.16 -1.08 -0.92  3509   1.0
## beta[5]    -1.46 1.9e-3 0.12 -1.68 -1.54 -1.46 -1.38 -1.23  3535   1.0
## beta[6]     -1.3 1.9e-3 0.12 -1.52 -1.38  -1.3 -1.22 -1.07  3878   1.0
## sigma       1.99 2.4e-3 0.17   1.7  1.88  1.98   2.1  2.35  4680   1.0
##
## Samples were drawn using NUTS at Tue 01 Dec 2020 11:47:22 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

In this section, we would like to evaluate three models using convergence analysis, cross-validation with PSIS-LOO, posterior predictive checking and prior sensitivity test. We are especially concerned about the choices of the three essential parameters α, β and σ in our model.

```
vars2plot =["alpha", "beta", "sigma"] # the variables that need to be plotted
```

4.1 Convergence Analysis

As the third section described, we use a linear model for prediction, so in this section, we check the distribution of alpha, beta (which determines the mean value of normal distribution) and sigma (variance of normal distribution).

4.1.1 Rhat value

As we can see in Section 3, all the $\hat{R}$ values of α, β and σ int the three models are 1.0, which means the Markov Chains in the three models have converged in the sense of $\hat{R}$. Actually, the idea behind $\hat{R}$ is the law of total variance:

$$Var(X) = E[Var(X|Y)] + Var\left(E[X|Y]\right),$$

where the first term means average variability within a chain (W), and the second means variability between chains (B). $\hat{R}$ is total variance divided by W then take square root:

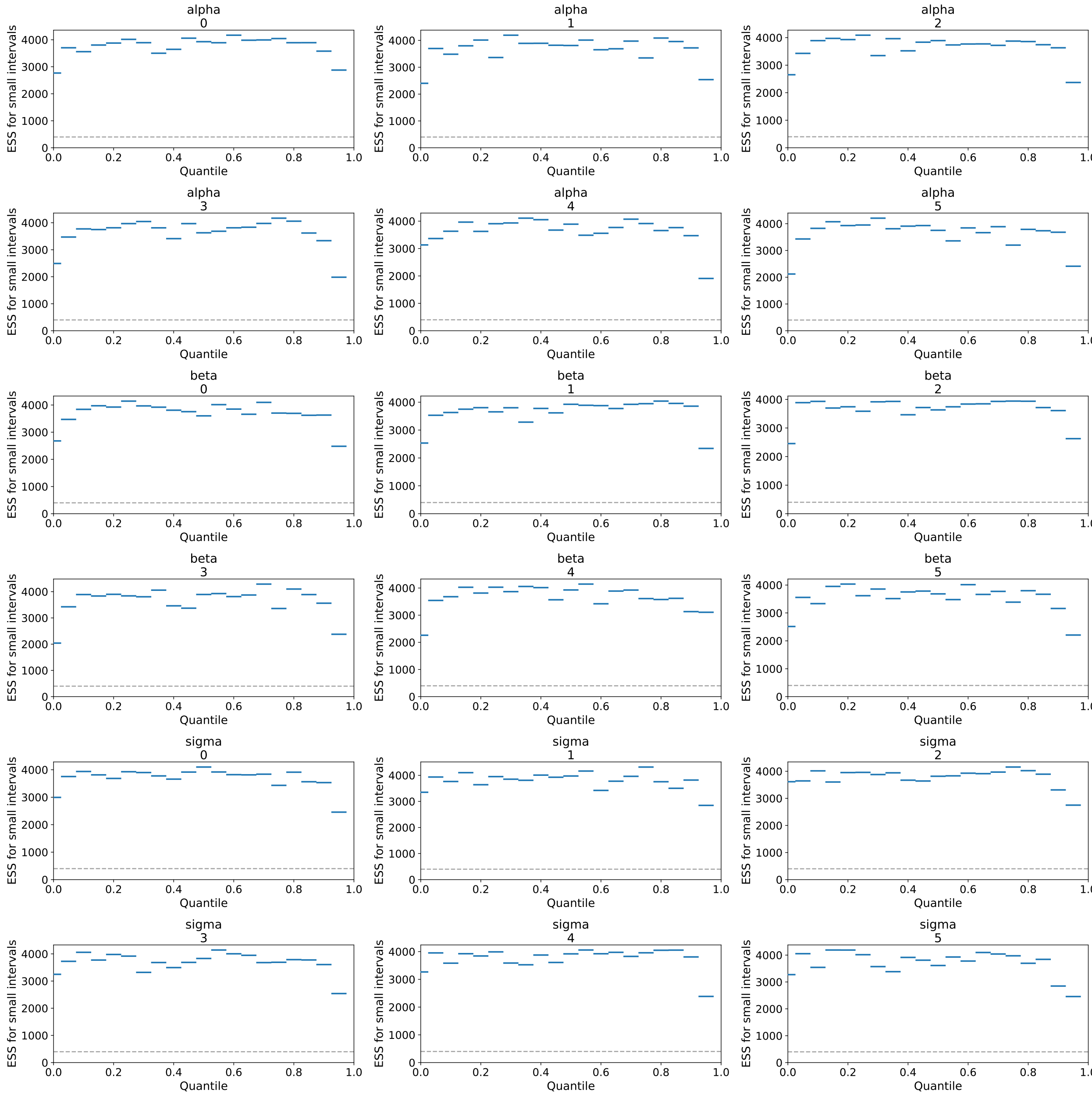
$$\hat{R} = \sqrt{\frac{N-1}{N} + \frac{1}{N}\frac{B}{W}} = \sqrt{1 + \frac{1}{N}(\frac{B}{W} - 1)}$$

From the expression we could see, if N is large enough, $\hat{R}$ will always converge to 1. If B is much smaller than W, $\hat{R}$ could be smaller than 1, which means the chains have explored a similar area in the distribution and they do not have many differences. If B is much larger than W, $\hat{R}$ could be somewhat larger than 1, which means the chains are fairly different from each other and the exploration of the distribution is not enough. In our case, $\hat{R}$ is 1.0, so W and B should be approximately equal, indicating that the chains have explored and converged to similar areas in the posterior distribution.

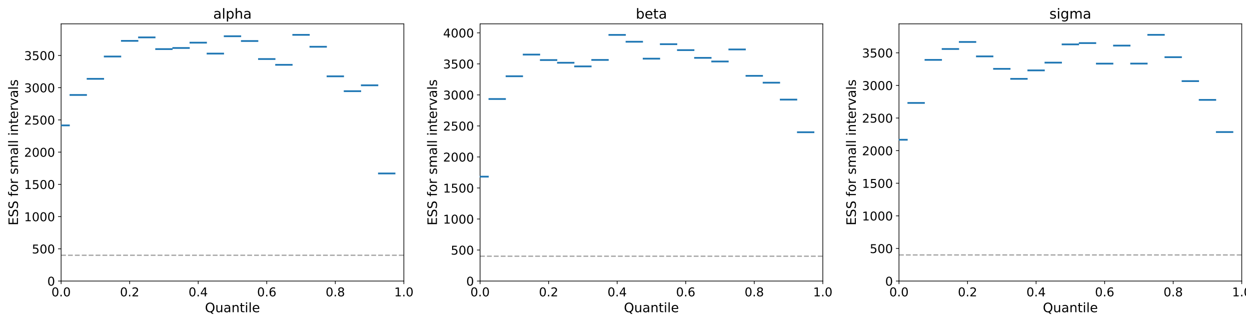
4.1.2 Effective Sample Sizes

In general, MCMC methods raise an issue that samples will be auto-correlated within a chain. Effective sample size(ESS) is thus used to check if a sample was a simple random sample. To evaluate the convergence of our models, we comply with such a rule: the higher the effective sample size, the more likely the chains for the draws have converged. As 4 stated, ESS should be at least 100 times the number of chains. Therefore, we plot the following figures: the dotted line represents 100*n_chains (we use 4 chains in practice). We can firmly conclude our model has converged since all ESS values are over the dotted line.

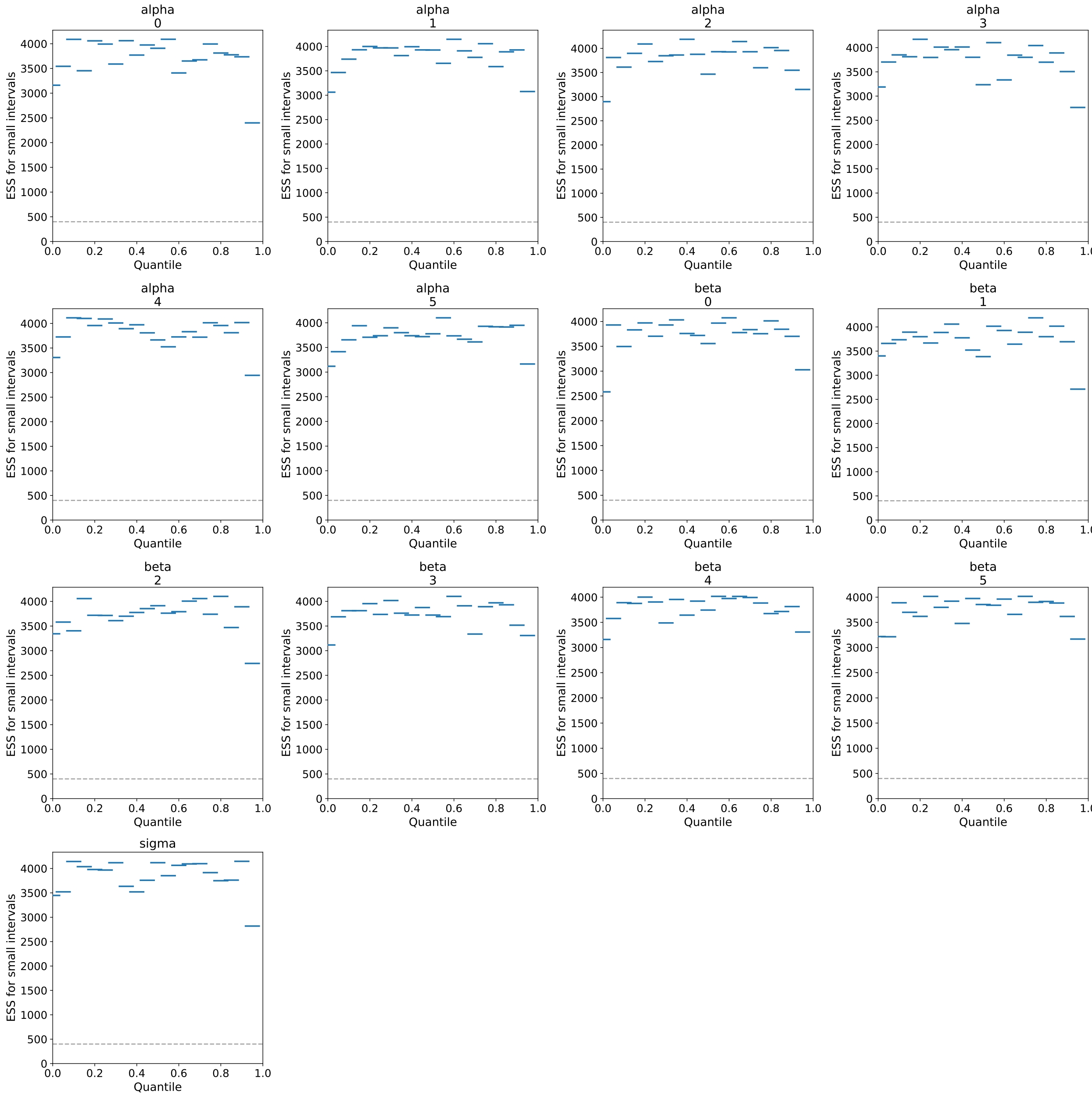
```
_ = az.plot_ess(
    separate_results, var_names=vars2plot,
    kind="local", marker="_", ms=20, mew=2, figsize=(20, 20)
)
plt.show()
```




```
_ = az.plot_ess(
    pooled_results, var_names=vars2plot,
    kind="local", marker="_", ms=20, mew=2, figsize=(20, 5)
)
plt.show()
```



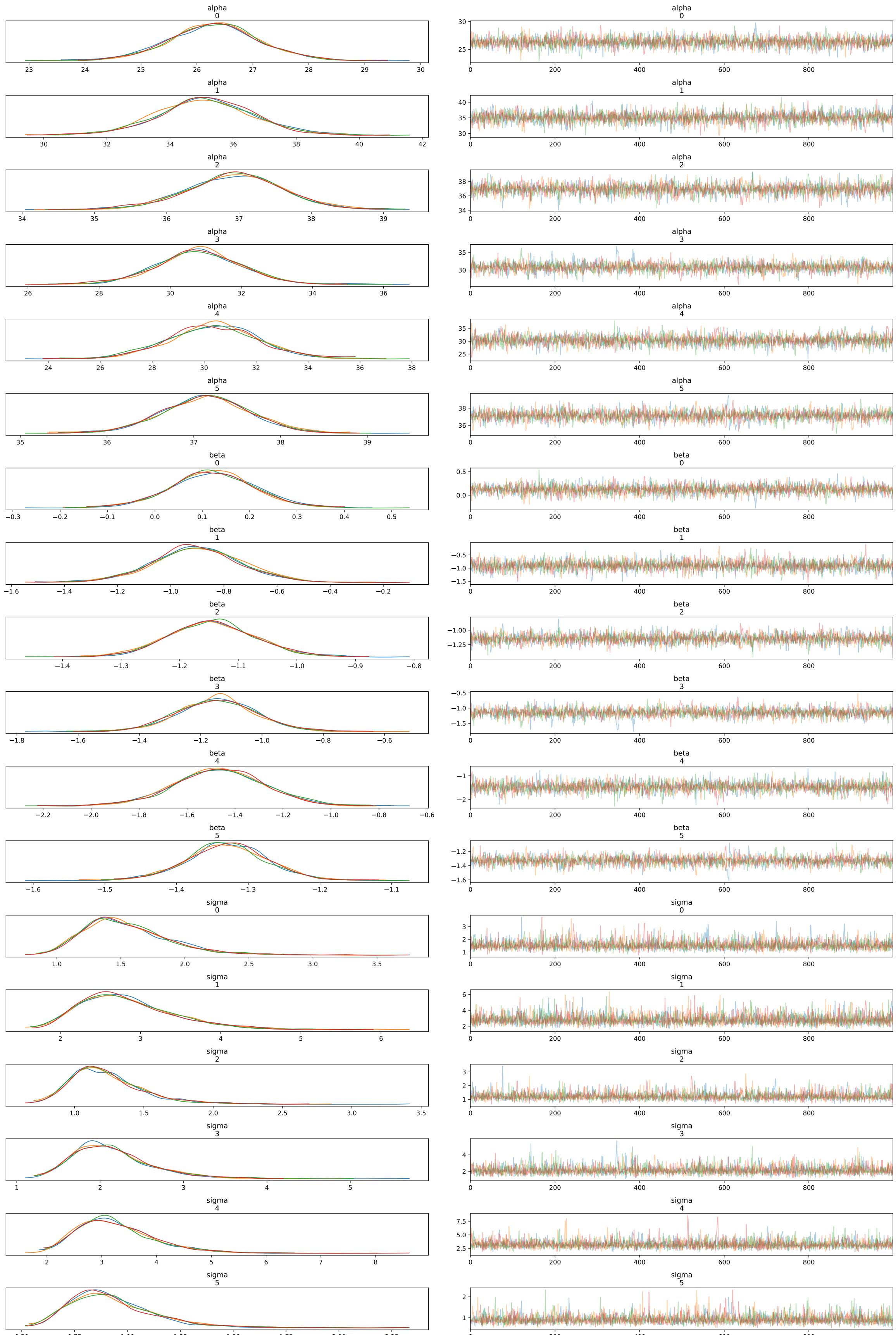

```python
_ = az.plot_ess(
    hier_results, var_names=vars2plot,
    kind="local", marker="_", ms=20, mew=2, figsize=(20, 20)
)
plt.show()
```



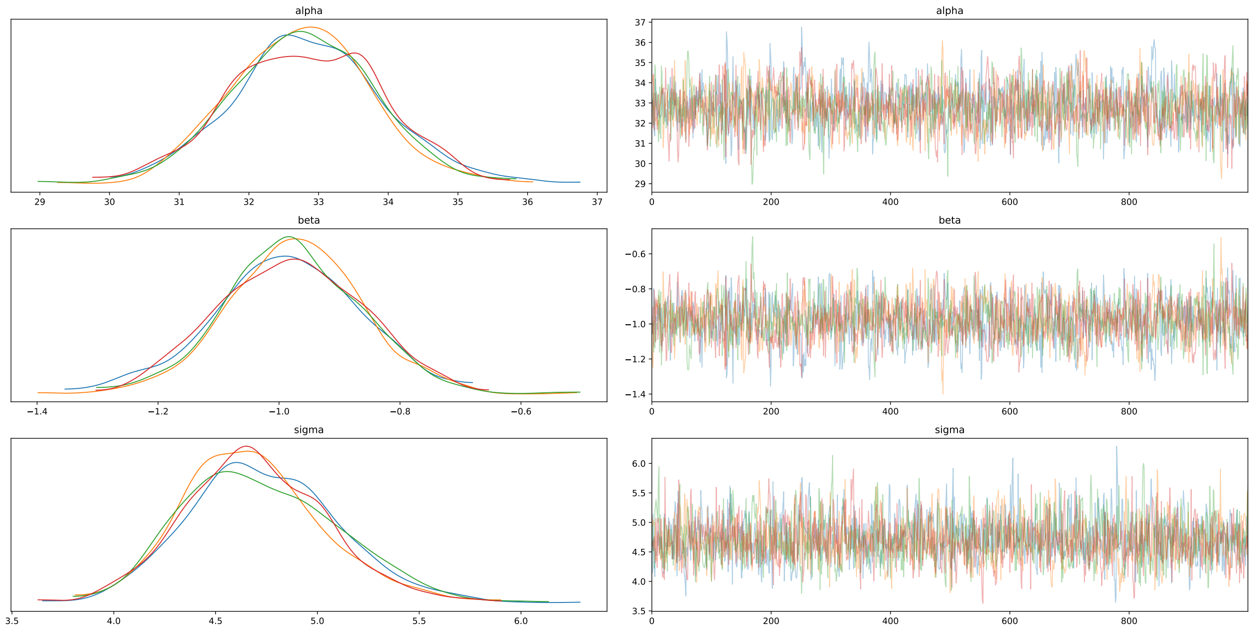
4.1.3 Chains Analysis

This section presents the plot of all chains in the sampling process. Generally, in all three models, all the chains have converged to distribution, since they just fluctuate within a small range and have a lot of overlap. In the pooled model, the variation among the chains is relatively larger than that in the separate and hierarchical model. Because it is more difficult to accurately represent all the different areas with the same set of parameters. so the chains would shift between different choices (especially in the plot for `sigma`). In the hierarchical model and separate model, each chain just needs to tackle a subset of data, instead of the whole data set, so the between-chain variation will be smaller.

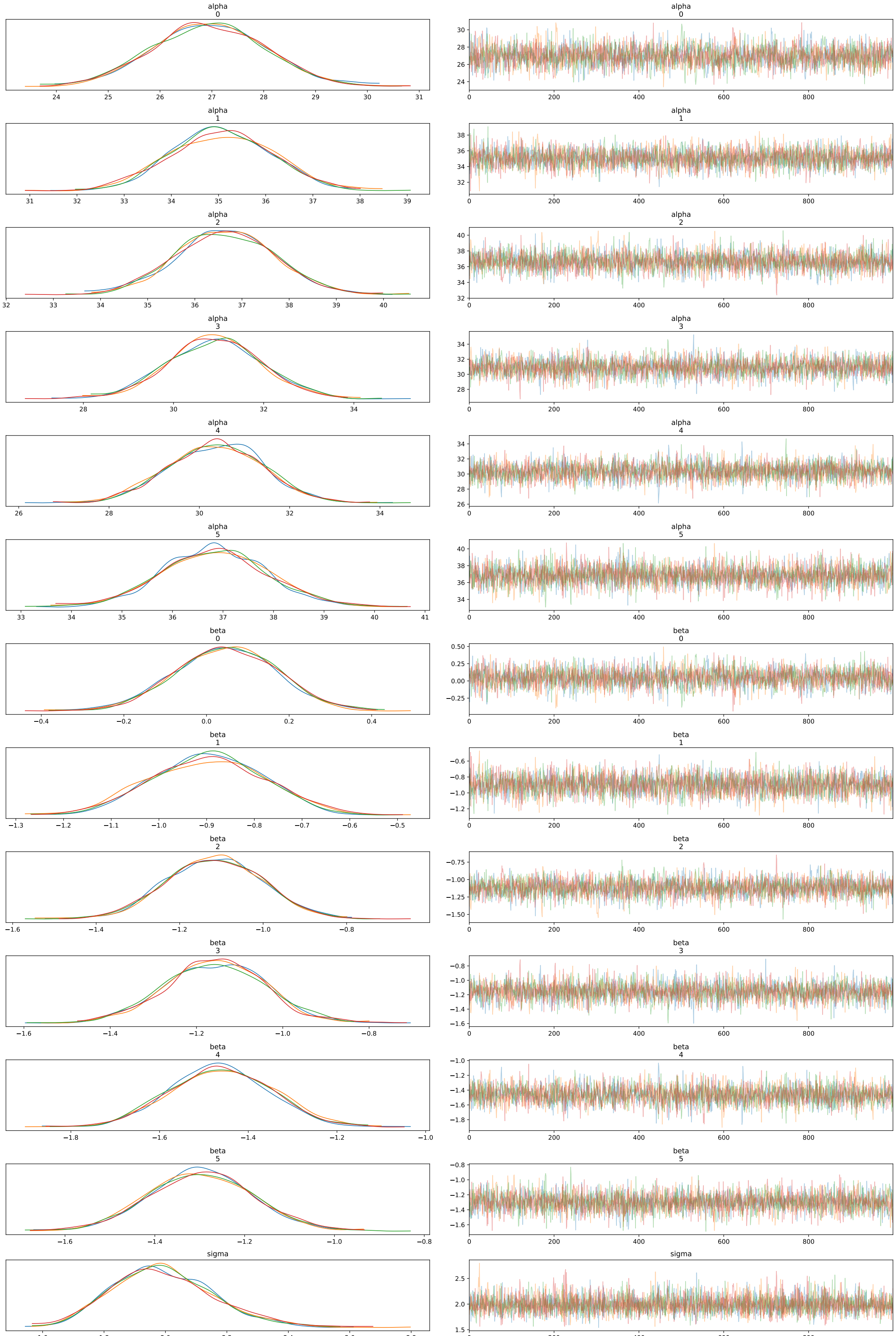
```
_ = az.plot_trace(separate_results, var_names = vars2plot, figsize=(20, 30))
plt.show()
```


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 = vars2plot, figsize=(20, 10))
plt.show()
```




```
_ = az.plot_trace(hier_results, var_names = vars2plot, 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

4.1.4 HMC/NUTS

"The validity of the estimates is not guaranteed if there are post-warmup divergences." 5 To avoid divergent transitions, we firstly check out the maximum treedepth, which is 6, 4 and 5 for separate, pooled and hierarchical models. Considering the default treedepth in stan is set to 10, our models have not exceeded this limit, therefore we just keep the depth as default. Secondly, we evaluate if our training process produces any divergent transitions. The results below also prove our models are convergent.

```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('treedepth quantiles:\n', h['treedepth__'].quantile([0, 0.25, 0.5, 0.75, 1]))
    print('divergent transitions: ', any(h['divergent__']))
```

```python
get_treedepth(separate_results)
```

```
## max treedepth for draws:  6
## min treedepth for draws:  3
## mean treedepth for draws:  4.05675
## treedepth quantiles:
##  0.00    3.0
## 0.25    4.0
## 0.50    4.0
## 0.75    4.0
## 1.00    6.0
## Name: treedepth__, dtype: float64
## divergent transitions:  False
```

```python
get_treedepth(pooled_results)
```

```
## max treedepth for draws:  4
## min treedepth for draws:  1
## mean treedepth for draws:  2.8935
## treedepth quantiles:
##  0.00    1.0
## 0.25    2.0
## 0.50    3.0
## 0.75    3.0
## 1.00    4.0
## Name: treedepth__, dtype: float64
## divergent transitions:  False
```

```python
get_treedepth(hier_results)
```

```
## max treedepth for draws:  6
## min treedepth for draws:  2
## mean treedepth for draws:  3.9645
## treedepth quantiles:
##  0.00    2.0
## 0.25    4.0
## 0.50    4.0
## 0.75    4.0
## 1.00    6.0
## Name: treedepth__, dtype: float64
```

```
## divergent transitions:  False
```

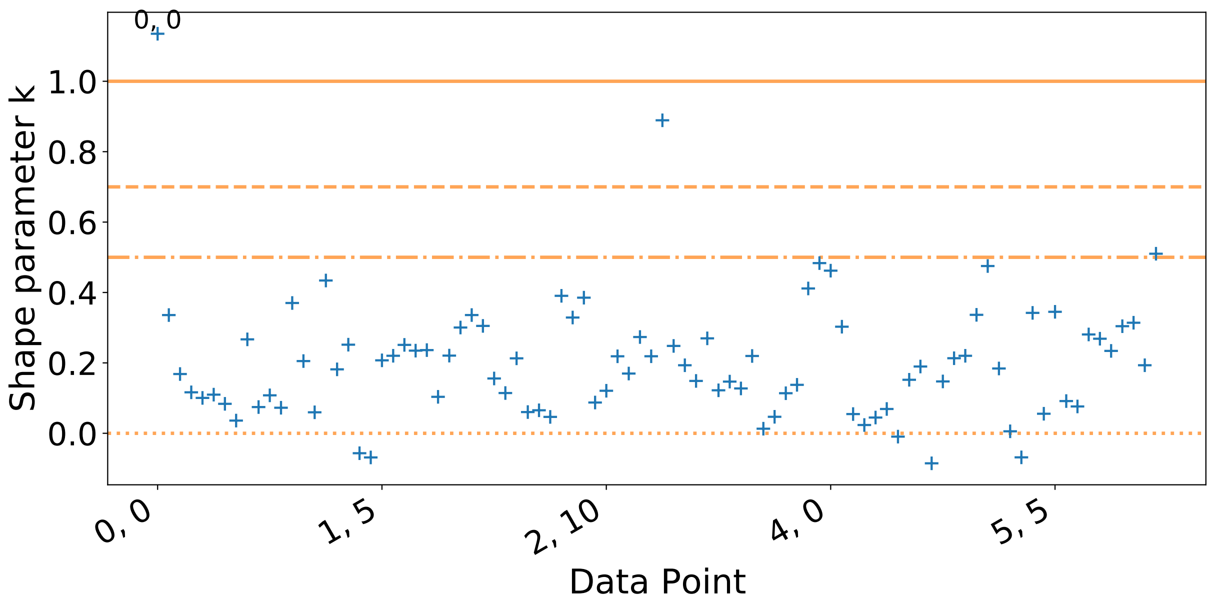
4.2 Cross-Validation with PSIS-LOO

In order to assess the model robustness of the separate, pooled and hierarchical model, we examine the behavior of the three models when a part of the data is excluded using leave-one-out cross-validation.

```
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()
```

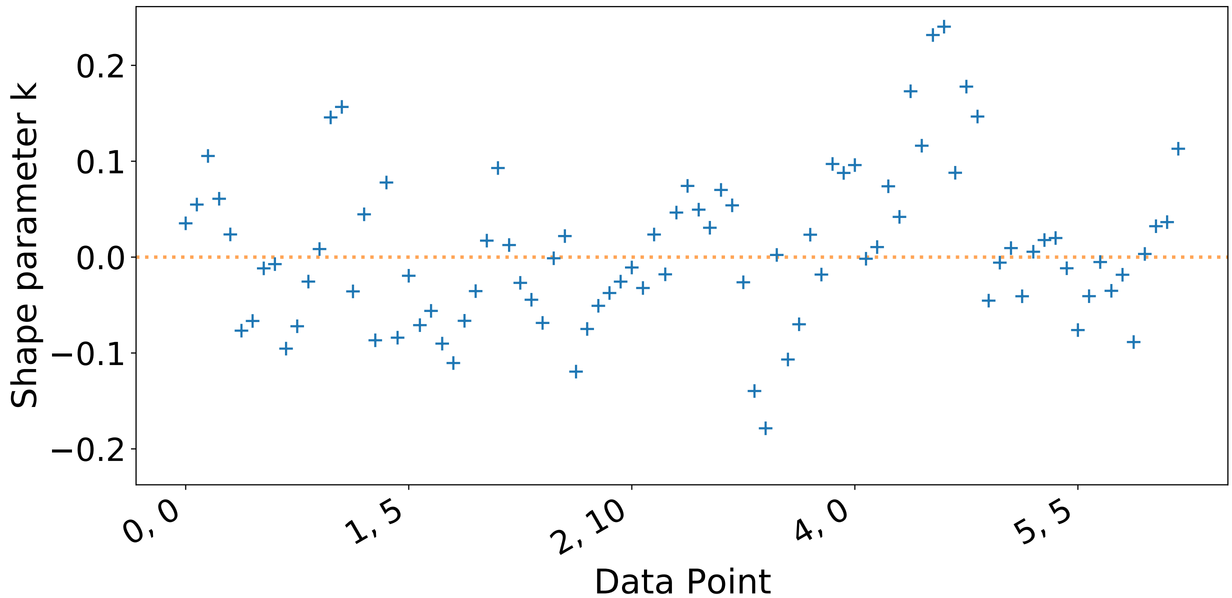
```
get_psis_loo_result(separate_results)
```

```
## Computed from 4000 by 90 log-likelihood matrix
##
##          Estimate       SE
## elpd_loo  -186.76     8.03
## p_loo       16.61        -
##
## 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)          1    1.1%
##    (0.7, 1]   (bad)         1    1.1%
##    (1, Inf)   (very bad)    1    1.1%
##
##
## /home/user/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 "
```



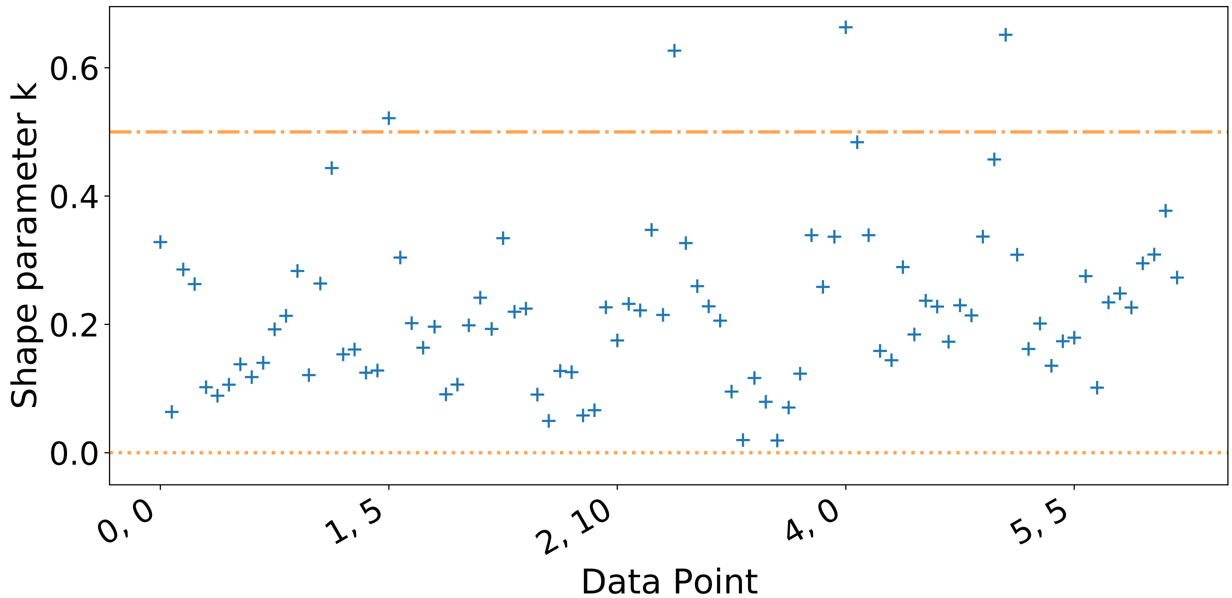

```
get_psis_loo_result(pooled_results)
```

```
## Computed from 4000 by 90 log-likelihood matrix
##
##          Estimate     SE
## elpd_loo  -267.77   6.32
## p_loo        2.79      -
## ------
##
## 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  -196.75    6.94
## p_loo       13.23       -
## ------
##
## 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%
```



As the results shown above, separate model is the best model since it has the largest elpd_loo value. By contrast, the pooled model with the smallest elpd_loo estimate is the worst model. The hierarchical model has a slightly lower elpd_loo value than separate model.

However, for the separate model, there are few pareto k diagnostic values that are very bad, which marginally increase the unreliability of the separate model. This mainly results from some highly influential observations in Great Manchester. As for the hierarchical model, with a little sacrifice on elpd_loo value, all its k diagnostic values are lower than 0.7.

In this section, we can conclude the pooled model is the worst model. But there is a trade-off between performance and reliability for separate model and hierarchical model. We will further compare these two models in the following sections.

4.3 Posterior Predictive Check

In this subsection, we will check posterior predictive distribution for 2020. We first define a utility function for plotting figures and then generate the plots to analyze predictive performance.

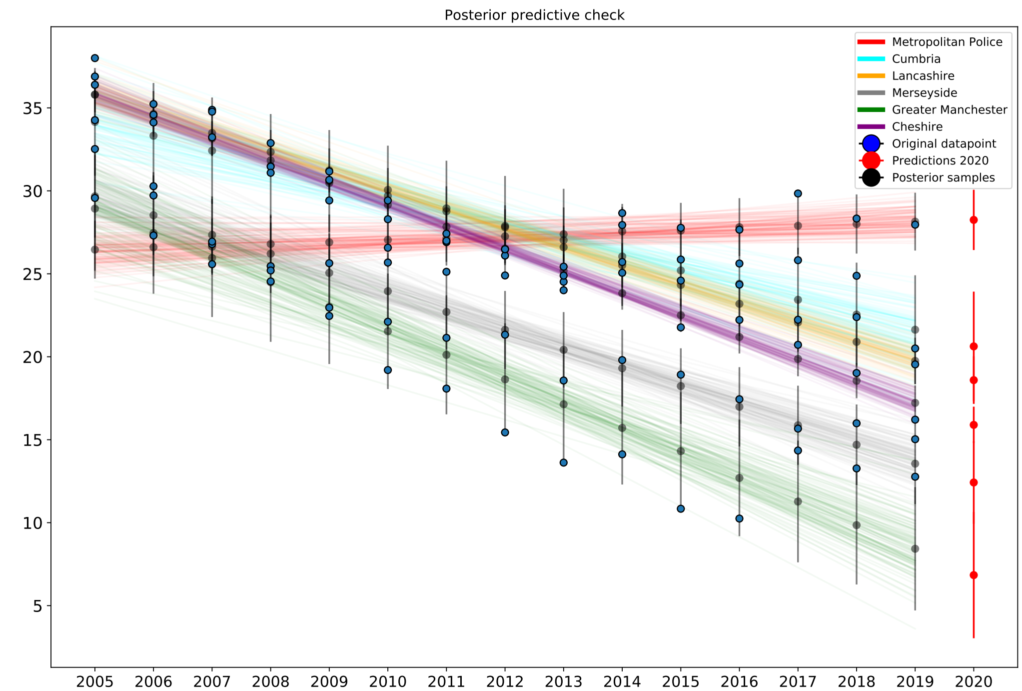
```
def plot_posterior_draws(stan_results, accident_data, pooled=False):
    plt.figure(figsize=(15,10))
    year_idx = np.arange(accident_data.shape[1])+1
    actual_years = year_idx + 2004

    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
```

```python
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.title("Posterior predictive check")
_ = plt.legend(bbox_to_anchor=(1.05, 1), loc='lower left', borderaxespad=0.)

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)
plt.show()
```

In the figures shown below, lines are drawn with different slopes and intercepts based on posterior distributions. Lines in different colors represent different police force areas. Blue, red and black points denote original data points, prediction number in 2020 and posterior samples respectively. At last, vertical lines indicate posterior and prediction variance. We can visually compare three models based on these three plots. For the pooled model, posterior samples in each area are extremely close and far away from original data points. In addition, the variance of prediction is large. For the separate model and hierarchical model, their performance is hard to visually distinguish. However, we can conclude that they are promising because their posterior samples in each area are close to original data points and the variance of prediction is small. To further compare these two models, we need quantitative evaluation, which will be introduced in the following section.

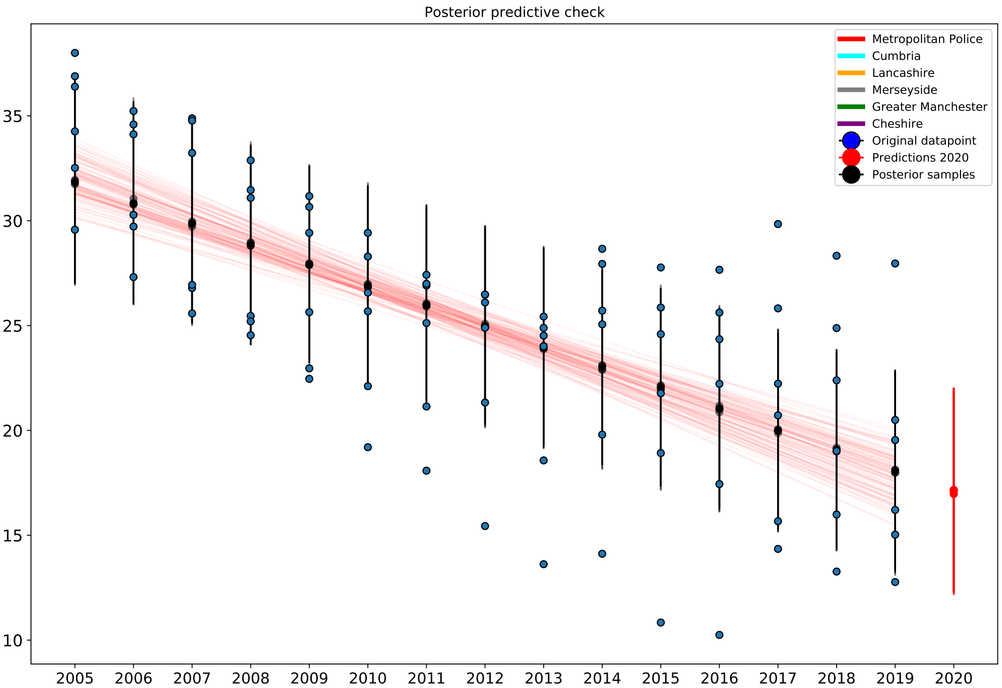
```
plot_posterior_draws(separate_results, accident_data)
```

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



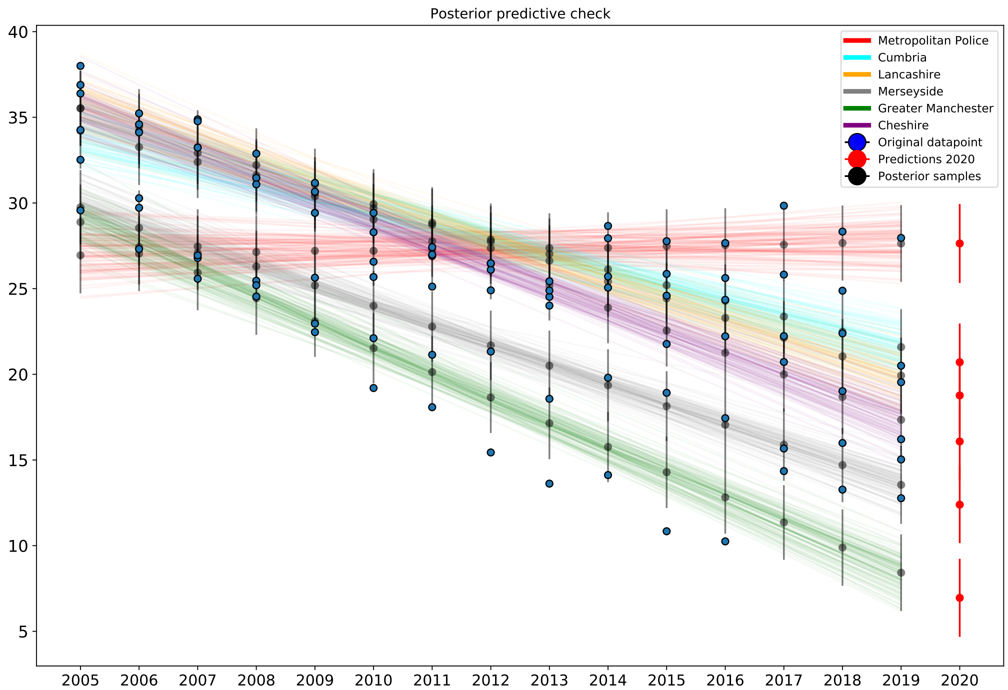

```
plot_posterior_draws(pooled_results, accident_data, pooled=True)
```

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




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

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



4.4 Predictive Performance

Also, we use R^2 6 as a metric to measure the performance of our models.

R^2 is defined as the variance of the predicted values divided by the variance of predicted values plus the expected variance of the errors:

$$R^2 = \frac{Var_\mu}{Var_\mu + Var_{res}},$$

where Var_μ is variance of modelled predictive means, and Var_{res} is the modelled residual variance. Therefore, the R^2 value should be close to 1 for a good fit. As we can see, the pool model has the smallest R^2 score, which is definitely the worst model. And the hierarchical model is slightly better than the separate model, with a higher R^2 score and lower variance.

Now, we prefer the hierarchical model over separate model since it is more reliable (discussed in section 4.2) and has a better predictive performance.

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

```
## r2        0.872656
## r2_std    0.086563
## dtype: float64
```

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

```
## r2        0.442949
## r2_std    0.006499
## dtype: float64
```

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

```
## r2        0.885403
## r2_std    0.005886
## dtype: float64
```

4.5 Prior Sensitivity Test

In this subsection, we manage to test the sensitivity of our models to the different prior distribution. Our models require slope and intercept prior information. Here, we treat the prior distribution described in Section 3 as default prior distribution, which is based on our observation and assumption that the mean accident rates change less than 50 % in one year. Then, we also apply non-informative prior distribution (uniform prior distribution) and another weakly prior similar with default prior, but it has a larger variance.

As can be seen from the forest plots, our models are robust with different prior choices, because there is no clear deviation in the posterior distributions with different prior. In addition, Rhat (all values are equal to 1) and ESS values indicate models can be converged with different prior settings.

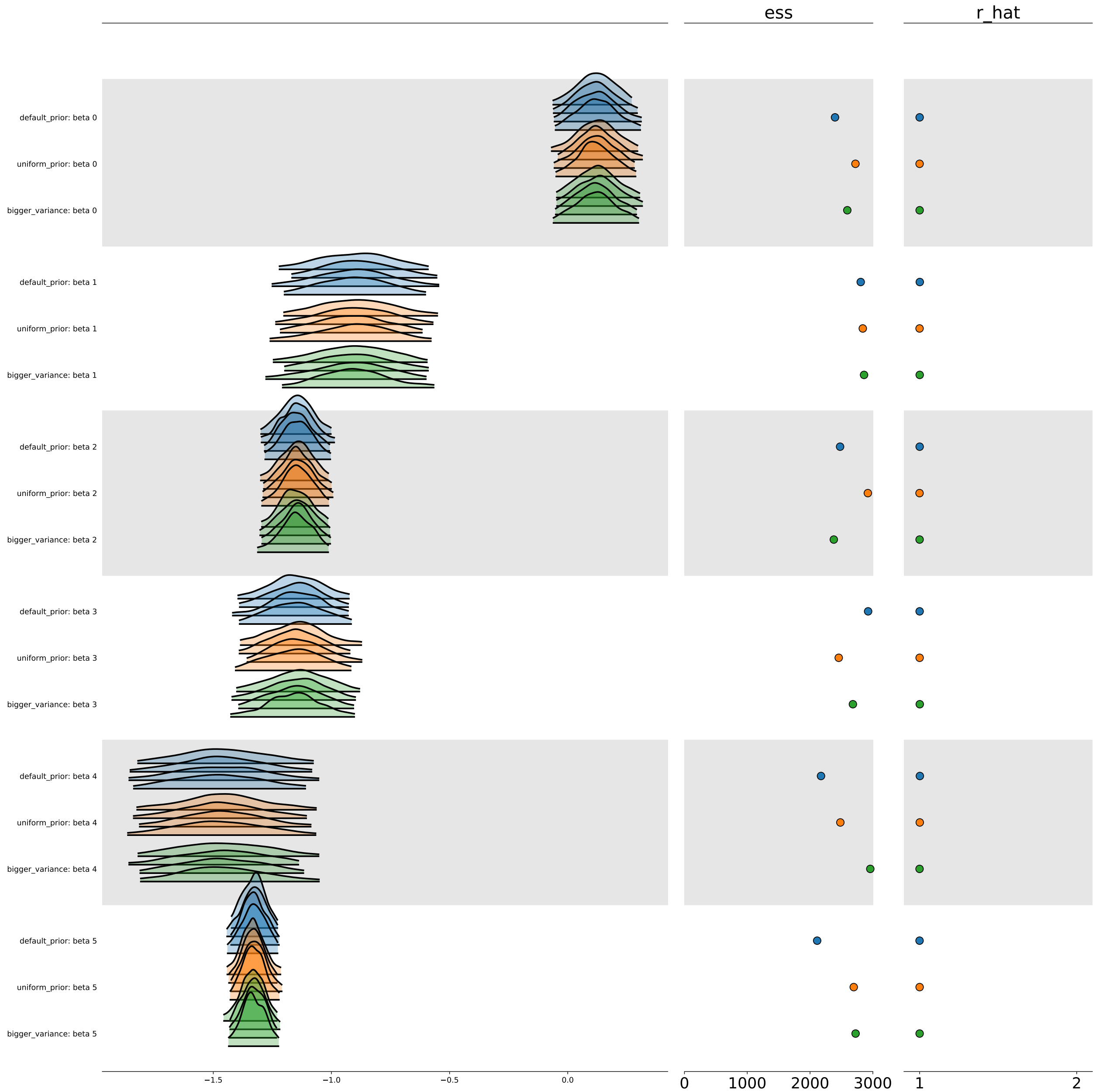
```
data_dict = dict()
names = ["default_prior", "uniform_prior", "bigger_variance"]
for i in range(3):
    current_stan_data = dict(
        N = accident_data.shape[0],
        Y = accident_data.shape[1],
        accidentData = accident_data,
        years = np.arange(1, accident_data.shape[1]+1), #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

```
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()
```

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

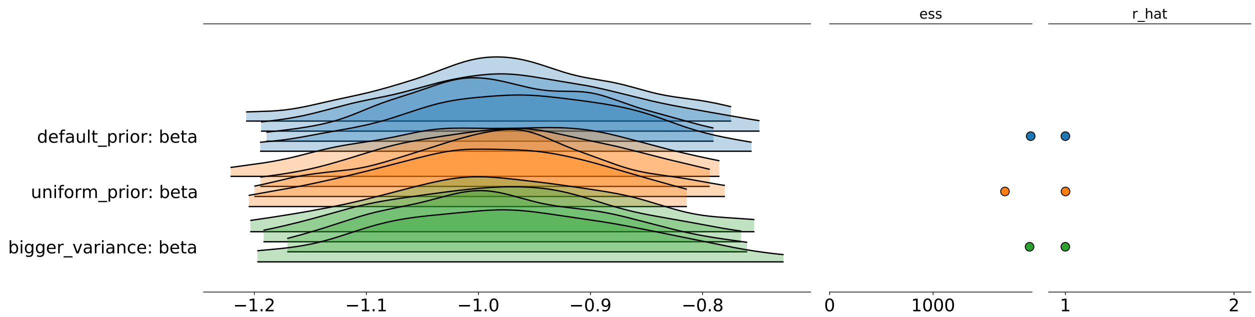
```
## Generating results with prior:1 default_prior
## Generating results with prior:2 uniform_prior
```

```
## Generating results with prior:3 bigger_variance
```



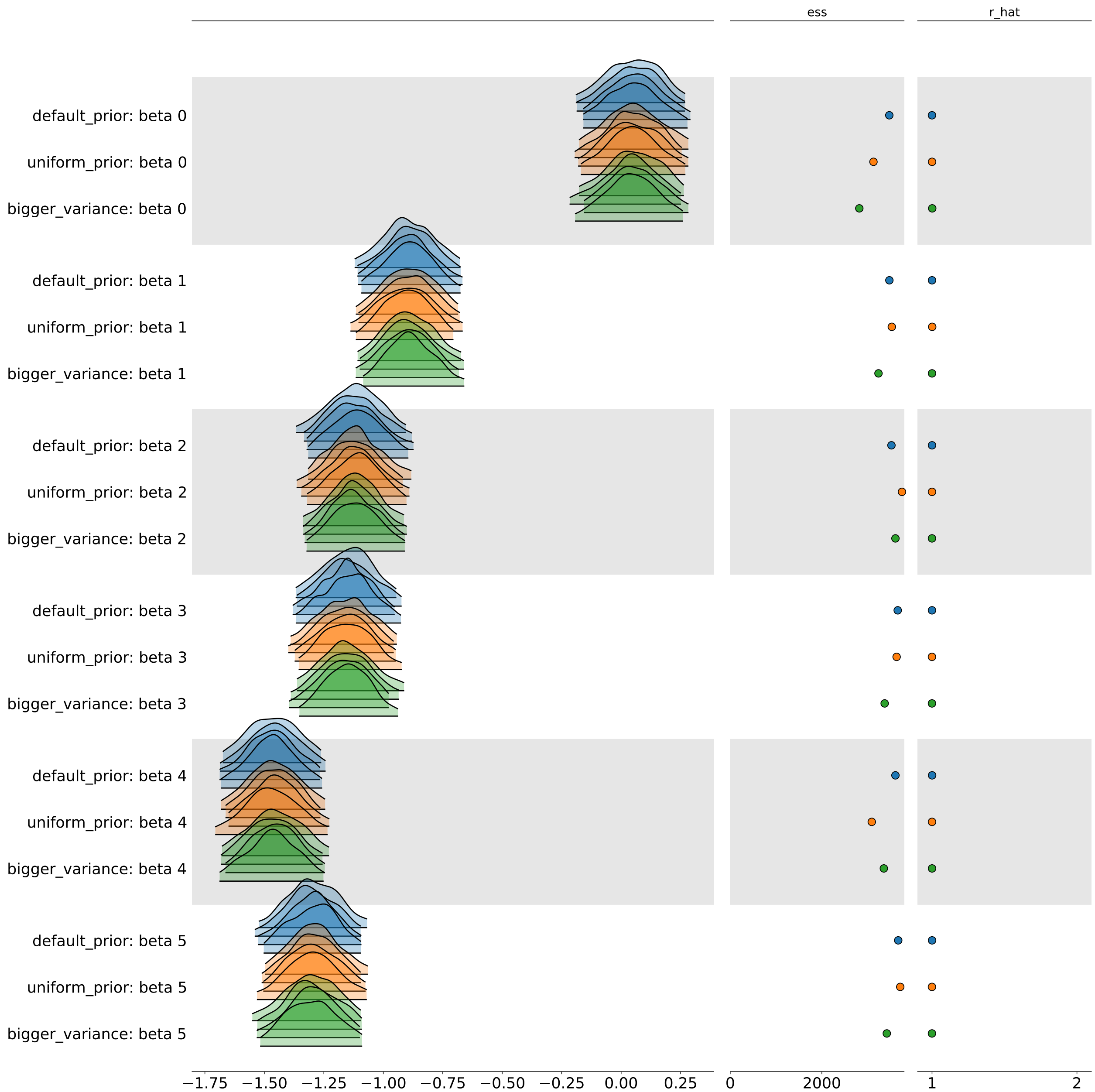

```
get_plot_forest(pooled_stan_model, data_dict, pooled=True)
```

```
## 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
```



5. Conclusion and Future Work

5.1 Project Conclusion

Based on the evaluation in section4, we would like to compare three models in four aspects, including convergence analysis, cross-validation with PSIS-LOO, posterior predictive checking, predictive performance and prior sensitivity test.

For convergence analysis, it is confirmed that all models are converged since 1. all Rhat values are 1.0; 2. ESS is large enough; 3. chains are visually converged. 4.no divergent transitions. From the result of cross-validation, we conclude the pooled model is the worst model. The separate model has a slightly larger elpd_loo value while the hierarchical model is more reliable. After checking posterior predictive distribution, we further confirm that the pooled model is the worst model. The separate model and the hierarchical model fit the data well and give a stable prediction. In the comparison of predictive performance, we find that the hierarchical model has the best predictive performance with the largest R^2 score and the smallest variance. Eventually, our proposed models are robust with a different prior distribution. Based on these results, it is clear that the hierarchical model is reliable and has a good predictive performance. Therefore, we adopt the hierarchical model as the optimal model to analyze our problem, which is to predict the car accident rate in different areas in the UK.

Based on the fitting result of the hierarchical model, the accident rate per 10000 people in Metropolitan Police area will probably increase by 0.05 and the estimated number of cases will reach about 24873 in total. In contrast, the accident rate in all the other five areas would decrease. In 2020, the Great Manchester would probably have 414 traffic accidents less than in 2019. Therefore, the local authority of the Metropolitan area should pay more attention to traffic management and propose effective methods to alleviate the harm from accidents.

5.2 Findings

The accident rate in an area could, to an extend, reflect the efficiency of local traffic management or people's adherence to traffic rules. Based on the fitting results of our model, we found that the separate model and hierarchical model give similar results, but the pooled model is far worse than the other two. We may conclude that the traffic management strength varies among those areas, and they can not be represented uniformly by a single normal distribution, since the pooled model merely demonstrates pool performance. Further, we have sufficient reasons to believe the variation is quite large, because the hierarchical model and the separate model give similar results, which means there is not a distinct hierarchical structure in UK's traffic management system. Nevertheless, we believe the hierarchical model is the best tool to capture the trends in our data, because of its robustness and predictive performance.

5.3 Future Work

1. In this project, we have evaluated three models, including separate model, pooled model and hierarchical model with normal distribution. We should also investigate other distribution to find possible improvement.
2. We have modeled the relationship between accident rate and years. We can further study the influence of weather, lighting condition, festivals and other factors on accident rates, since these studies can provide a more informative prediction for people.
3. Currently, we only analyze six representative areas because of limited computational resources. In the future, we would like to add more areas and even predict accident rates per month to obtain a more comprehensive result.

References

[1] Global status report on road safety 2018 https://www.who.int/publications/i/item/9789241565684 [2] Road Safety Questions and Answers https://www.racfoundation.org/motoring-faqs/safety#a1. [3] Road Safety Data https://data.gov.uk/dataset/cb7ae6f0-4be6-4935-9277-47e5ce24a11f/road-safety-data [4] What is Effective Sample Size https://www.displayr.com/what-is-effective-sample-size/ [5] Brief Guide to Stan's Warnings https://mc-stan.org/misc/warnings.html [6] Bayesian R2 and LOO-R2 https://avehtari.github.io /bayes_R2/bayes_R2.html