

Final Project: Aviation Safety Data Modeling

Ash

2019/01/06

1 Introduction

As the world become more globalized, different regions are becoming more and more interconnected by thousands of flights every day. Since the amount of air travel has been steadily increases, it is interesting to explore the safety of this mode of transportation through statistical analysis. Personally, Minervans are travelling the globe mostly with airplane so some insights on the safety features of airlines and aircraft types are also of interest. This project aims to use statistical methods to investigate two things: the general trend of air accident over the year and the mortality risks involved with air accident, using data from the Aviation Safety Network.

	day	month	year	type	operator	fat.	cat	Aircraft Type	Aircraft Cat	date
0	1	04	1931	Ford 5-AT-C Tri-Motor	Panagra	0	A1	Military_research	Military	1931-04-01
1	1	04	1937	Ford 5-AT-C Tri-Motor	Star Air Service	0	A1	Military_research	Military	1937-04-01
2	1	04	1956	Martin 4-0-4	TWA	22	A1	Commercial_civil	Commercial	1956-04-01
5	1	08	1942	Junkers Ju-52/3mg4e	German AF	0	A1	Commercial_civil	Commercial	1942-08-01
6	1	12	1984	Boeing 720-027	NASA	0	O1	Commercial_civil	Commercial	1984-12-01

Figure 1: Examples Of Cleaned Data

The Aviation Safety Network (ASN) provides a comprehensive record of air accidents for both commercial and military aircrafts carrying at least two people, including cargo and research flights. Unfortunately the data is not accessible in a convenient format, so we transcribed the data from the ASN website to a spreadsheet, with raw information about the date of the accident, the airline operated the flight (e.g., American Airlines, Nippon AIr, etc.), the type of aircraft involved (e.g., Boeing 747, Airbus A300, etc.), the number of fatality and the category of accident (e.g., mechanical issues, hijacking, etc.) The raw data is then cleaned in the following order:

- Three accidents without dates are omitted from analysis
- Aircraft type was matched with a list of commercial and military aircraft on Wikipedia, then classified as such. Accidents with unknown aircraft types were also omitted
- A commercial airline list from Wikipedia were also matched with the airline in the raw data to double-check the initial classification by the aircraft type. Mismatches between the two methods were visually inspected, and aircraft types that were used by both military and commercial purposes were classified as commercial if a known commercial airline employed those types

The code for cleaning data is included in appendix A1 and some examples of the cleaned data is shown in Figure 1. Variable `cat` represents the category of accident (which will be explained in details in section 3), and the number following the category is whether the aircraft was repairable or was a hull loss (1 for hull loss, 2 for repairable). Further data processing is described in section 2 and 3.

2 Accident Trends

First, the cleaned data is categorized into two subsets: commercial and military. This is justified for modeling trends because both subsets are governed by very different events in the real world (for example, military accident peaks when there is a war). Counting the number of accidents in a year, we have roughly 100 datapoints for each subset from 1919 to 2018. The first 70 years will be used in the modeling step and the rest of the data will act as a validation set to confirm our models. A visualization of both subsets is shown in Figure 2.

Two modeling techniques will be considered: Bayesian linear model and statistical sampling using Stan. Both techniques can quantify uncertainty in their predictions but they rely on different assumptions.

2.1 Global trend: Linear Model

We assume two things for both model: 1) the global trend is linear with some fluctuations modeled as random noise and 2) even though the observed quantity is integer, we model the trend as a continuous variable with rounding. A linear model has three unobserved variables: the coefficient for the slope, the intercept and the noise due to unaccounted fluctuation of the data. The Bayesian regression framework that Scikit-learn implements (figure 3a) assumes the number of accident is generated by a normal distribution with the trend modeled as a linear combination between product of the slope and the year and the intercept, plus some random noise (which, in turn, has a Cauchy prior with scale=1 and $x_0 = 1$). Both the slope and the intercept are generated by a multivariate normal distribution with a vector mean μ and a covariance matrix $\lambda \mathbb{I}$. μ

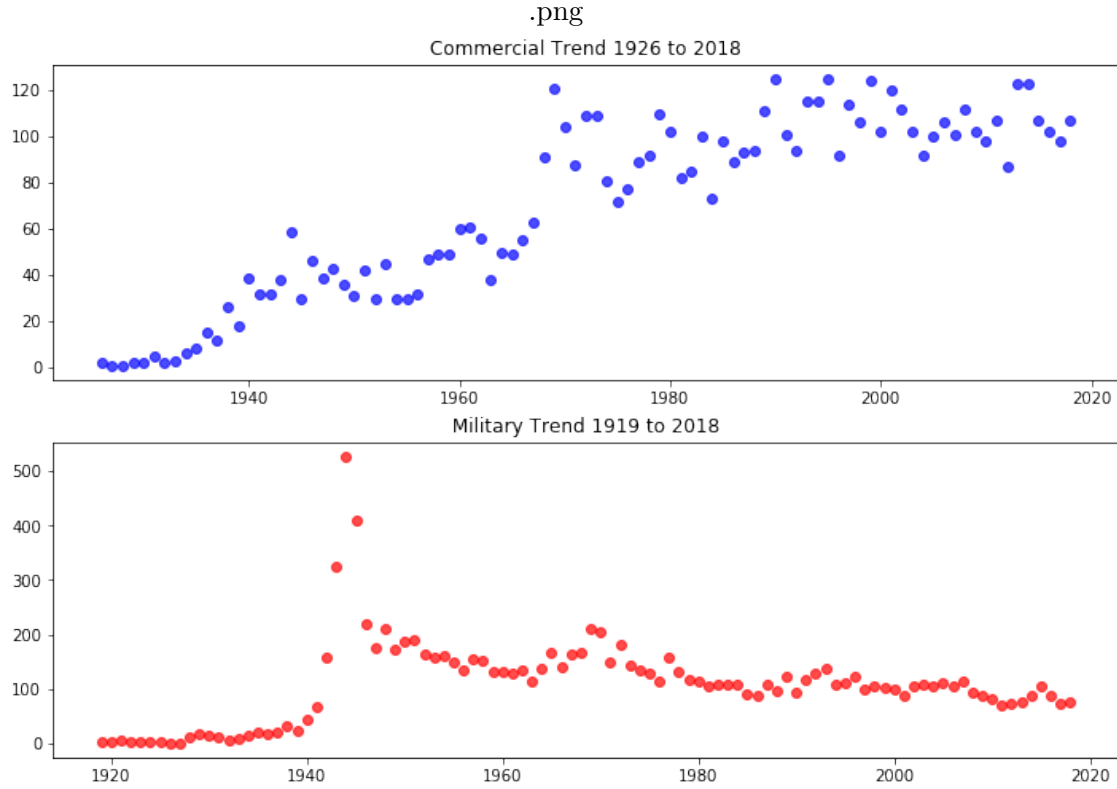


Figure 2: Number Of Accident Per Year, 1919 To 2018

and λ are generated by two Gamma distributions with hyperparameters. The hyperparameters and parameters for this model is jointly estimated by expectation maximization (maximizing the marginal log likelihood).

```

1 model_trend = """
2
3 data {
4 //Data and its length
5   int<lower=1> length;
6   real<lower=0> data_[length];
7 //Number of generated values
8   int<lower=0> n_gen;
9 }
10
11 parameters {
12   //Linear trend model, with 2 coefficients
13   real coef_1;
14   real coef_2;
15   real<lower=0> noise;
16 }
17
18 model {

```

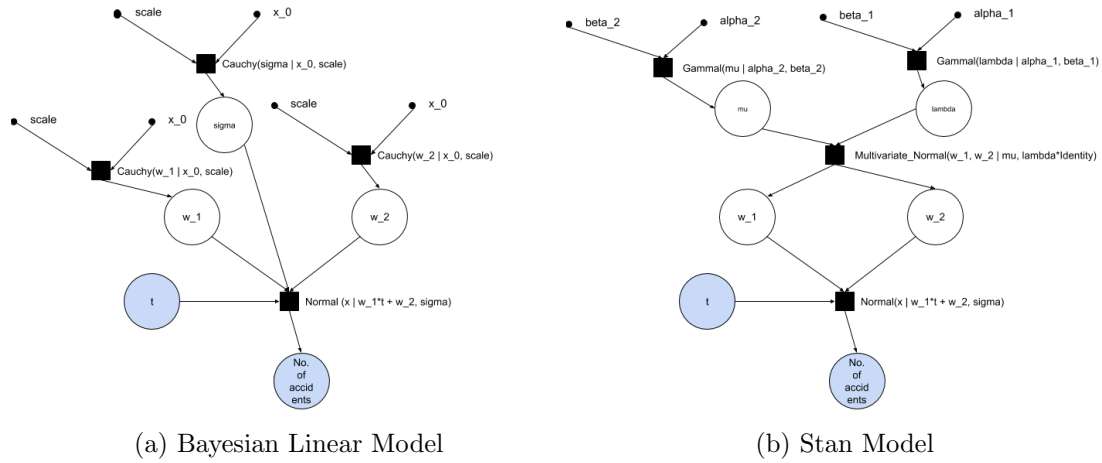


Figure 3: Factor Graph Of Linear Models

```

19  #Each coefficient is drawn from a broad Cauchy
20  coef_1 ~ cauchy(0,1);
21  coef_2 ~ cauchy(0,1);
22  noise ~ cauchy(0,1);
23  for(i in 1:length) {
24      data_[i] ~ normal(coef_1*i + coef_2, noise);
25  }
26 }
27
28 generated quantities {
29     real data_gen[n_gen];
30     for(i in 1:n_gen) {
31         data_gen[i] = normal_rng(coef_1*(i+length) + coef_2, noise);
32     }
33 }
34 """
35 stan_model_trend = pystan.StanModel(model_code=model_trend)

```

The Stan model (figure 3b), specified by code above, has a different priors. The slope, intercept and noise are generated by three different broad Cauchy since we assume we don't know the ranges of these parameters beforehand, and these Cauchy-s have specified hyperparameters. The estimations of the parameters are obtained using MCMC sampling of the posterior.

In figure 4a, we can see that model fitted the linear trend reasonably well, with the confidence interval widening where we do not observed the data. Predictions for the unseen data indicates that because we assumes a linear trend, the model fails to anticipate the plateau right after the training data. A similar failure happens to the generated samples of the Stan model in figure 4b. This situation is to be expected since the assumption of linear trend is a strong one that does not hold in this case due to many factors, such as improvements of technical aspects of the aircrafts or the advent of inter-

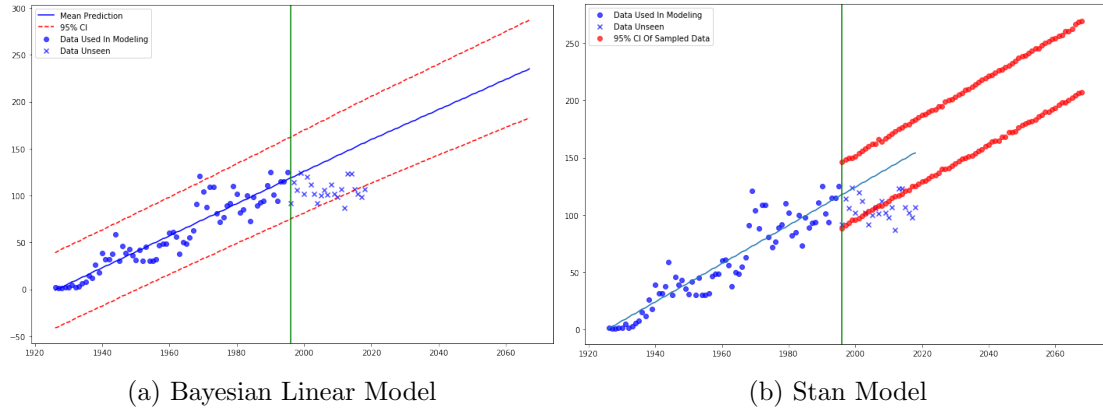


Figure 4: Predictions Of Linear Models - Commercial

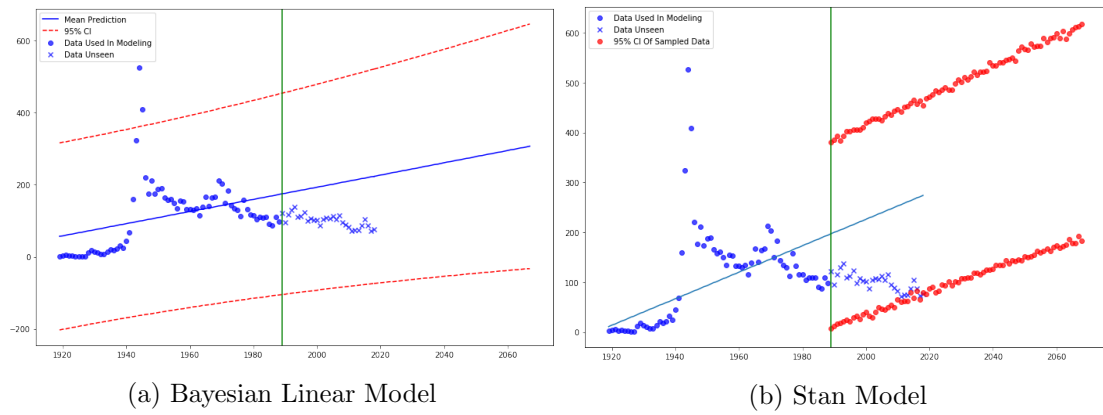


Figure 5: Predictions Of Linear Models - Military

national safety standards in aviation. Interestingly, due to the effect of WWII, the linear trend assumption worked even poorer for the military accidents (figure 5), as shown by the wide confidence intervals of both the Bayesian and the Stan model. Another point worth mentioning is that the estimations of the coefficients by both models with different constructions and different methods of estimations (variational versus MCMC) are the same. This might be because in a simple linear model with only two parameters, especially when one of them is the intercept which depends only on the chosen coordinate system, the parameters are independent of each other so estimations using a multivariate normal (Bayesian model) do not differ much from estimations using independent Cauchy-s (Stan model).

2.2 Global trend: Quadratic Model

```
1 model_trend = " " "
```

```

2
3 data {
4 //Data and its length
5   int<lower=1> length;
6   real<lower=0> data_[length];
7 //Number of generated values
8   int<lower=0> n_gen;
9 }
10
11 parameters {
12   //Quadratic trend model, with 3 coefficients
13   real coef_1;
14   real coef_2;
15   real coef_3;
16   real<lower=0> noise;
17 }
18
19 model {
20   #Each coefficient is drawn from a broad Cauchy
21   coef_1 ~ cauchy(0,1);
22   coef_2 ~ cauchy(0,1);
23   coef_3 ~ cauchy(0,1);
24   for(i in 1:length) {
25     data_[i] ~ normal(coef_1*i*i + coef_2*i + coef_3, noise);
26   }
27 }
28
29 generated quantities {
30   real data_gen[n_gen];
31   for(i in 1:n_gen) {
32     data_gen[i] = normal_rng(coef_1*(i+length)*(i+length) + coef_2*(i+
33     length) + coef_3, noise);
34   }
35 }
36 """
37 stan_model_trend = pystan.StanModel(model_code=model_trend)

```

The only different between the quadratic model and the linear model is that we have three coefficients instead of two: the additional one accounts for the quadratic effect of t^2 . The factor graphs do not differ much from the linear model, but the predictions do. In figure 6, because up to the point where the training data is fed to the model, the trend seems relatively linear so the coefficient of the squared term is small. However, this small term contributes to the rapid widening of the confidence interval since the model assumes a quadratic trend ought to happen at some point in the future, diverging the predictions. Similar observations can be made with the Stan model, where the generated values are also diverging quickly, albeit not as much as the Bayesian model. This is because the interaction between t and t^2 expressed by the estimated covariance matrix of the Bayesian model (since it uses a multivariate normal and the coefficients are generated jointly from them) magnifies the non-linear trend. The performance of the quadratic model is even worse in the military subset in figure 7. As we can see, the model mistakenly assumes a downward trend (since the main assumption is that the

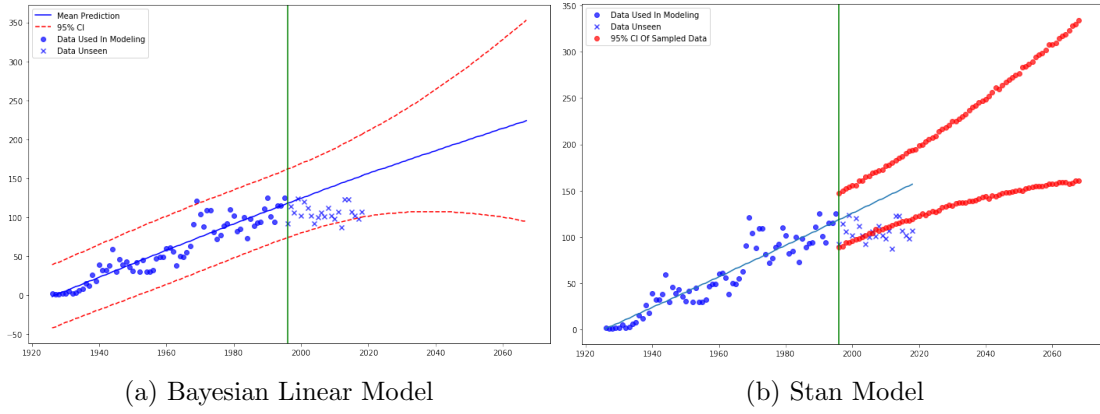


Figure 6: Predictions Of Quadratic Models - Commercial

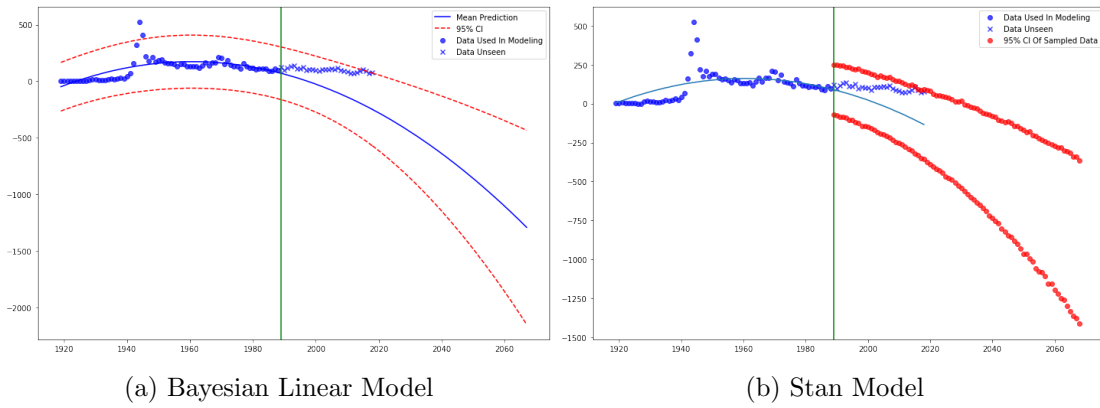


Figure 7: Predictions Of Quadratic Models - Military

trend is quadratic) from WWII on, making the prediction seriously off. Furthermore, since we make no explicit specification for the sign of the observed data (which should be positive) the model predicts negative values.

Clearly, modeling the global trend in this case is not good since no obvious trend can be observed by inspection of the data, and if we keep adding polynomial terms we risk overfitting the model. It's natural to consider modeling the local trend instead, since we might suspect the number of accident per year is affected most by the most recent years, and the further away the less effect any event might have. For example, it's unlikely that WWII has much effect in the difference between the number of accident in 1980 and 1981, but it's highly likely that the number of accidents in WWII is a good predictor for the next few years.

2.3 Local trend: Radial Basis Function Model

A radial basis function has the form

$$f(x, x_0) = e^{-\frac{(x-x_0)^2}{\sigma}}$$

which measures the similarity (by Euclidean distance) between the current point x and a pivot point x_0 . This measure of similarity can be used as a feature to indicate the relative position between the data, and since the effect of the pivot point diminishes exponentially with the distance between the points, datapoints separated by a long distance won't have much similar trend, which is exactly what we need to model local trend. Using 100 evenly spaced pivot points between 1919 and 2119 then calculate these similarity measures, we can fit both a Bayesian model and an MCMC model.

```
1 scaler = MinMaxScaler()
2 N = 70
3
4 #Using sklearn rbf kernel
5 from sklearn.metrics.pairwise import rbf_kernel
6
7 #100 pivot points
8 kernel = np.linspace(0,2,100).reshape(-1,1)
9 #Rescaling the year to the range 0 to 1, a standard procedure before
   fitting
10 year_count_com['index'] = scaler.fit_transform(np.array(year_count_com['
   index']).reshape(-1,1))
11
12 X_train = rbf_kernel(np.array(year_count_com['index'][:N]).reshape(-1,1),
   kernel, gamma=10)
13
14 #Using Bayesian model and plotting the results
15 reg = BayesianRidge()
16 reg.fit(X_train, year_count_com['year'].iloc[:N])
17 plt.figure(figsize=(12,8))
18 x = year_count_com['abs_year']
19 plt.scatter(x[:N], year_count_com['year'][:N], color='b', alpha=0.7, label=
   'Data Used In Modeling')
20 plt.scatter(x[N:], year_count_com['year'][N:], color='b', marker='x', alpha
   =0.7, label='Data Unseen')
21
22 X_test = rbf_kernel(np.array(year_count_com['index']).reshape(-1,1),\
23                     kernel, gamma=10)
24 y_m, y_std = reg.predict(X_test, return_std=True)
25 plt.plot(x, np.round(y_m), color='b', label='Mean Prediction')
26 plt.plot(x, np.round(y_m+2.96*y_std), color='r', linestyle='--', label='95%
   CI')
27 plt.plot(x, np.round(y_m-2.96*y_std), color='r', linestyle='--')
28
29 f_x = np.linspace(1,1.5,50).reshape(-1,1)
30 K_f_x = rbf_kernel(f_x, kernel, gamma=10)
```



```

31 f_y_m, f_y_std = reg.predict(K_f_x, return_std=True)
32 f_x = range(2018, 2018+50)
33 plt.plot(f_x, np.round(f_y_m), color='b')
34 plt.plot(f_x, np.round(f_y_m+2.96*f_y_std), color='r', linestyle='—')
35 plt.plot(f_x, np.round(f_y_m-2.96*f_y_std), color='r', linestyle='—')
36
37 plt.axvline(x[N], color='g')
38 plt.legend()
39 plt.show()
40
41 #Using Stan model
42 model_trend_main = """
43
44 data {
45 //Data and its length
46     int<lower=1> length;
47     real<lower=0> data_[length];
48 //Number of radial basis points or number of coefficients
49     int<lower=1> c;
50 //Kernelized value of the training data
51     vector[c] rbf_train[length];
52 //Number of generated values
53     int<lower=0> n_gen;
54 //Kernelized value of the generated data
55     vector[c] rbf_gen[n_gen];
56 }
57
58 parameters {
59 //RBF model, with c basis points
60     row_vector[c] coef_;
61 //Intercept
62     real intercept;
63 //Gaussian noise
64     real<lower=0> noise;
65 }
66
67 model {
68 //Each coefficient is drawn from a broad Cauchy
69     for(j in 1:c) {
70         coef_[j] ~ cauchy(0,1);
71     }
72     for(i in 1:length) {
73         data_[i] ~ normal((coef_*rbf_train[i]) + intercept, noise);
74     }
75 }
76
77 generated quantities {
78     real data_gen[n_gen];
79     for(i in 1:n_gen) {
80         data_gen[i] = normal_rng((coef_*rbf_gen[i]) + intercept, noise);
81     }
82 }
83 """
84 trend_model = pystan.StanModel(model_code=model_trend_main)

```

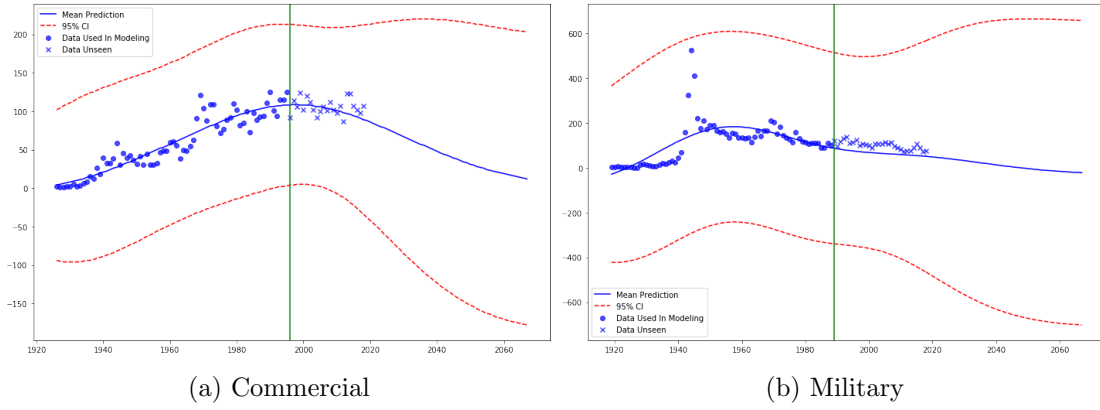


Figure 8: Predictions Of RBF Bayesian Models

```

85
86 kernel = np.linspace(0,2,100).reshape(-1,1)
87 X_train = rbf_kernel(np.array(year_count_com['index'][:N]).reshape(-1,1),
88                        kernel, gamma=10)
89 X_gen = rbf_kernel(np.array(year_count_com['index'].iloc[N:]).reshape(-1,1)
90                    , kernel, gamma=10)
91 f_X = np.linspace(1,1.5,50).reshape(-1,1)
92 K_f_X = rbf_kernel(f_X, kernel, gamma=10)
93 X_gen = np.concatenate((X_gen, K_f_X), axis=0)
94
95 stan_data = {
96     'length': year_count_com['year'][:N].shape[0],
97     'data_': np.array(year_count_com['year'][:N]),
98     'c': 100,
99     'rbf_train': X_train,
100    'n_gen': year_count_com['year'][N:].shape[0] + 50,
101    'rbf_gen': X_gen
102 }
103 results = trend_model.sampling(data=stan_data)
104 samples = results.extract()

```

In Figure 8 we can see the Bayesian models of commercial and military accidents. By modeling the local the prediction for unseen data is reasonably accurate for both case, and as we expect the confidence interval widened for regions we do not observe data. For this type of model it's important to select the number of pivot points appropriately since if we have too many pivot points we will just model random fluctuations in the data which means we are overfitting. Too few pivot points mean we essentially abandon the local trend for the global trend. Another point worth mentioning is that in figure 9 a failure mode of MCMC sampling is observed: because the pivot points are closed, the coefficients are closely dependent and therefore the posterior has multiple separated peaks, which is the type of distribution Stan hates the most. Thus, R-hat shows that the chains did not mix well and we observe the haywire prediction (full results in the code).

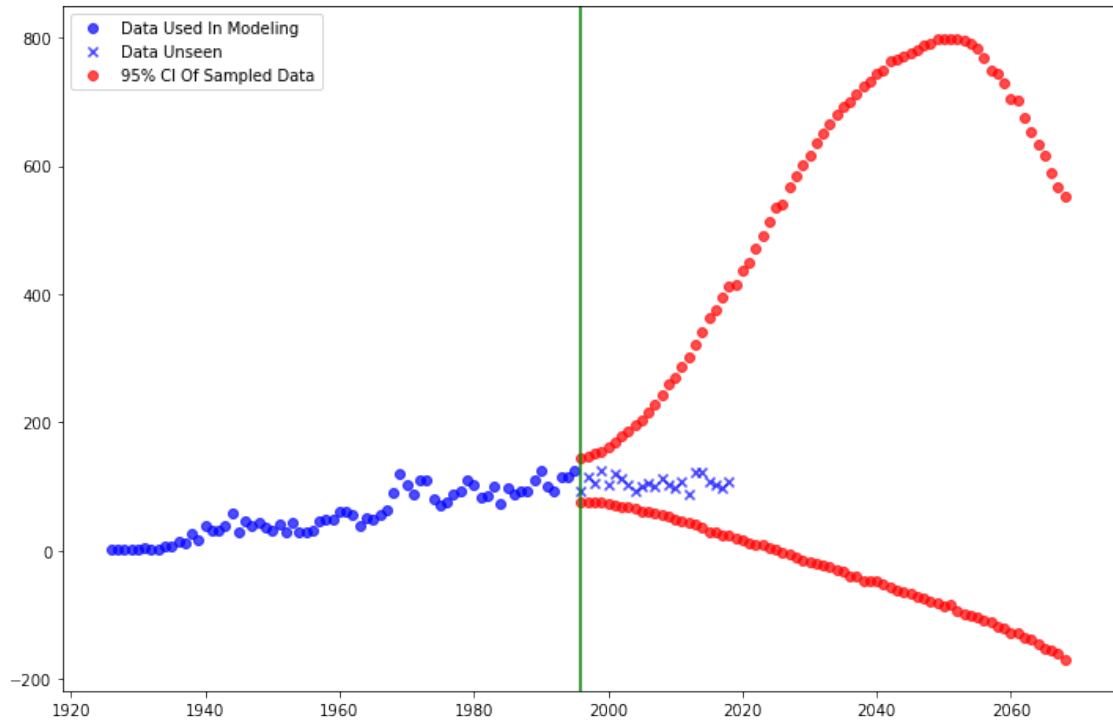


Figure 9: Stan RBF Model

3 Mortality Risks & Hull Loss Risks

In the unfortunate event of an accident, we are interested in the risk of fatality given the type of accident we are in, the type of aircraft and the airline operator. Given that the commercial subset is of particular interest in this scenario, we will focus the analysis on this subset, but the model can also be used for the military subset. First, due to the large amount of data available, and as established from the previous section: more recent accident data is more informative to predict future events, we select only a part of the commercial subset for analysis. Given that most commercial aircraft still available today is of either Boeing or Airbus, we select only these aircraft types. We also select only the top 25 airline with the most recorded accident because sampling for the full list of available airline (around 1300) will take too much time. After further cleaning, we have about 620 datapoints for analysis.

```

1 stan_model_risk = ""
2
3 data {
4   //Number of airline

```

```

5     int<lower=1> L;
6 //Number of aircraft
7     int<lower=1> C;
8 //Number of accident type
9     int<lower=1> T;
10 //Number of datapoint
11     int<lower=1> N;
12
13 //A vector of 0/1 in each entry expressing which airline , aircraft and
    accident type the datapoint is
14     vector<lower=0>[L] airline[N];
15     vector<lower=0>[C] aircraft[N];
16     vector<lower=0>[T] accident[N];
17
18 //A vector expressing the number of passenger on that aircraft
19     int<lower=0> passenger[N];
20
21 //Number of fatality
22     int<lower=0> fat[N];
23 }
24
25
26 parameters {
27
28 //Scaling factor for the airline
29     row_vector<lower=0, upper=1>[L] m_al;
30 //Scaling factor for the aircraft
31     row_vector<lower=0, upper=1>[C] m_ac;
32 //Scaling factor for the accident type
33     row_vector<lower=0, upper=1>[T] m_a;
34
35 //Mortality rate
36     real<lower=0, upper=1> p;
37 }
38
39 model {
40
41     for (l in 1:L){
42         m_al[l] ~ lognormal(0,0.25);
43     }
44
45     for (c in 1:C){
46         m_ac[c] ~ lognormal(0,0.25);
47     }
48
49     for (t in 1:T){
50         m_a[t] ~ lognormal(0,0.25);
51     }
52
53     for(i in 1:N) {
54         #Sample the mortality rate from a beta distribution , with scaling
        factor from the airline , aircraft and accident
55         p ~ beta(((m_al*airline[i])+(m_ac*aircraft[i])+(m_a*accident[i])),
            1);

```

```

56     fat[i] ~ binomial(passenger[i], p);
57   }
58 }
59
60 """
61 model_risk = pystan.StanModel(model_code=stan_model_risk)

```

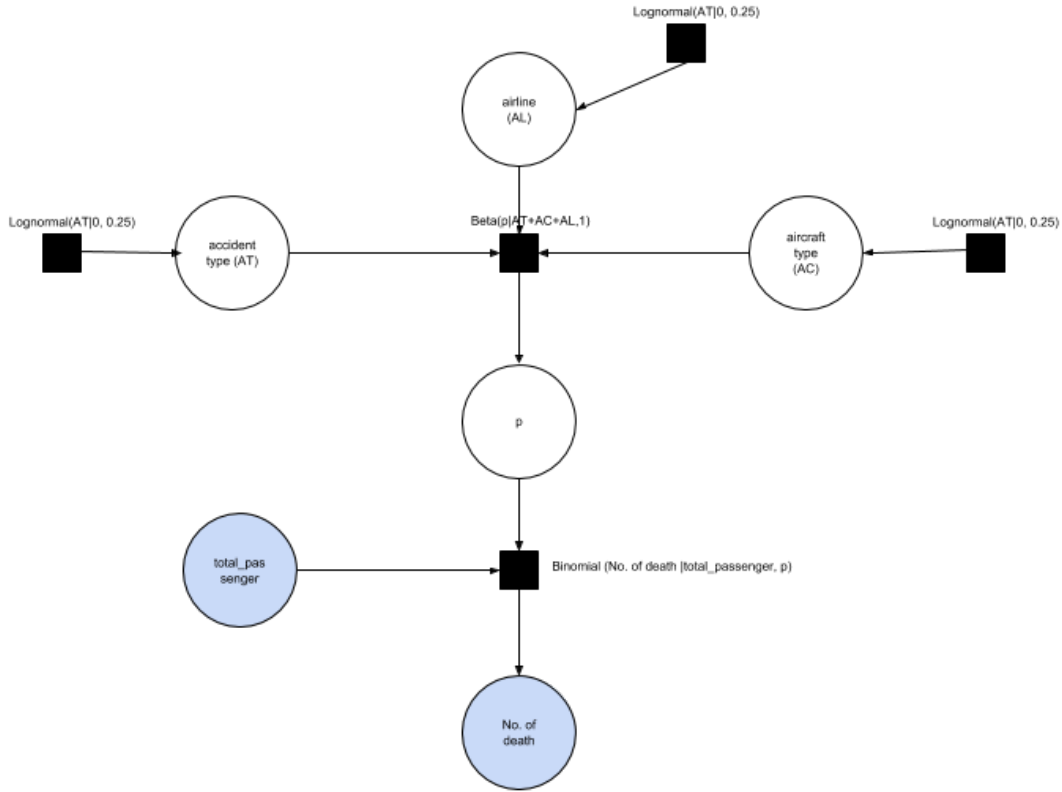


Figure 10: Mortality Risk Model Factor Graph

The model is specified in Stan code above and described in figure 10 factor graph. The unobserved variable of the model includes p , the mortality rate; airline scale factor, which is used as a component of the alpha parameter for the beta distribution; aircraft type scale factor and accident type scale factor. The mortality rate is generated from a beta distribution that skewed to the left if the scale factor sum is large, so larger value of the scale factor means higher chance of death. Finally the number of death is generated from a binomial with the total number of passenger on the plane (depends on the aircraft) and the mortality rate. The results are presented in figure 11 and 12.

In figure 11a we can see that relatively the airlines have quite similar "risk factor". The highest risk belongs to VASP, and the lowest seems to be American Airlines. Com-

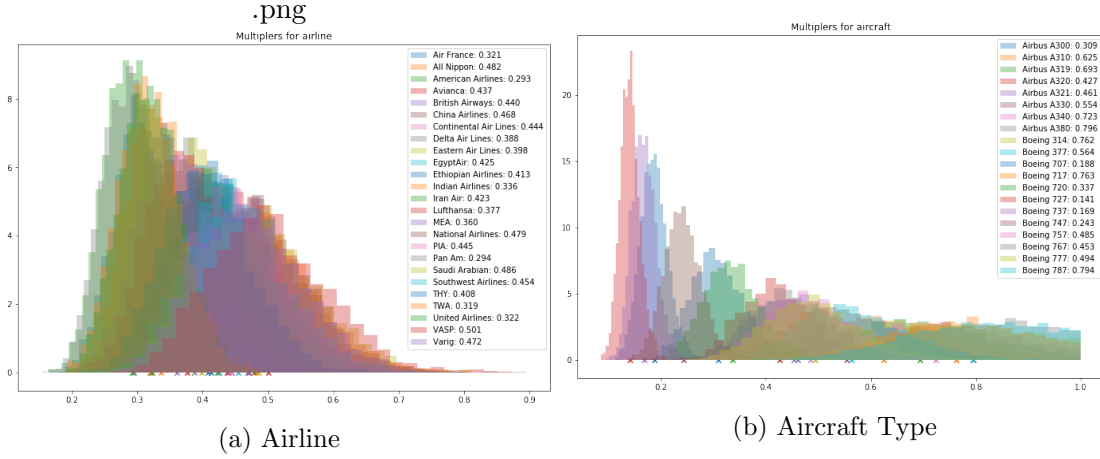


Figure 11: Posteriors Of Scaling Factor - Mortality Risk

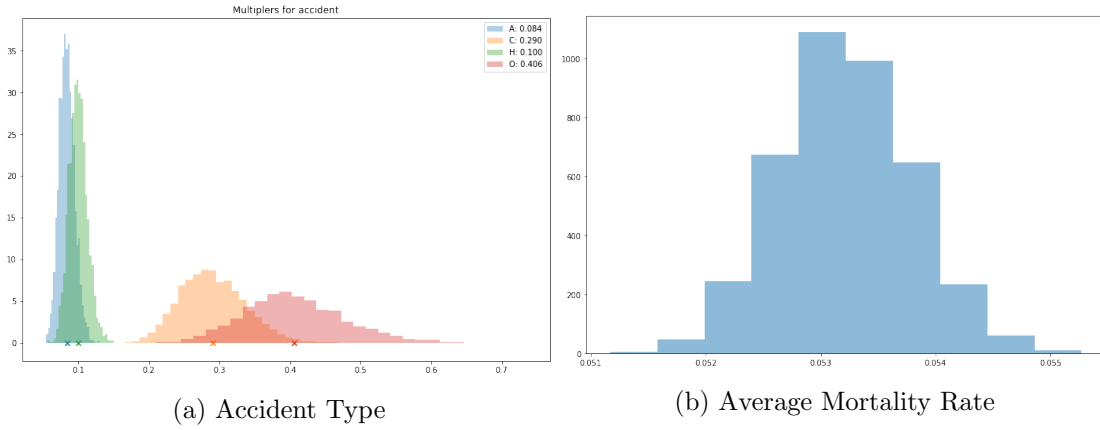


Figure 12: Posteriors Of Scaling Factor - Mortality Risk

pared to the airline, the aircraft type plays a bigger role in the risk of fatality. Bigger airplane (in terms of carrying capacity) seems generally safer, but the biggest airplane in the group, Airbus A380, has the higher scaling factor, which suggests when an accident happen to a big plane it's very usually catastrophic. In figure 12 we can see the most risky type of accident is O type, which is ground fire or sabotage. The mortality rate seems low (0.05 on average, suggesting that usually only 5% of passengers die in an accident), which makes sense intuitively considering how safe aviation travel is.

The same analysis can be made for classification of whether it's a hull loss or not by replacing the binomial distribution as the likelihood by a bernoulli. The results, summarized in figure 13 and 14, suggests that generally the risk for hull-loss is very high (76%) should an accident occur, and all scaling factor concentrates near 1, indicating a high risk of hull-loss in any event of accident (although some type of aircraft or accident seems

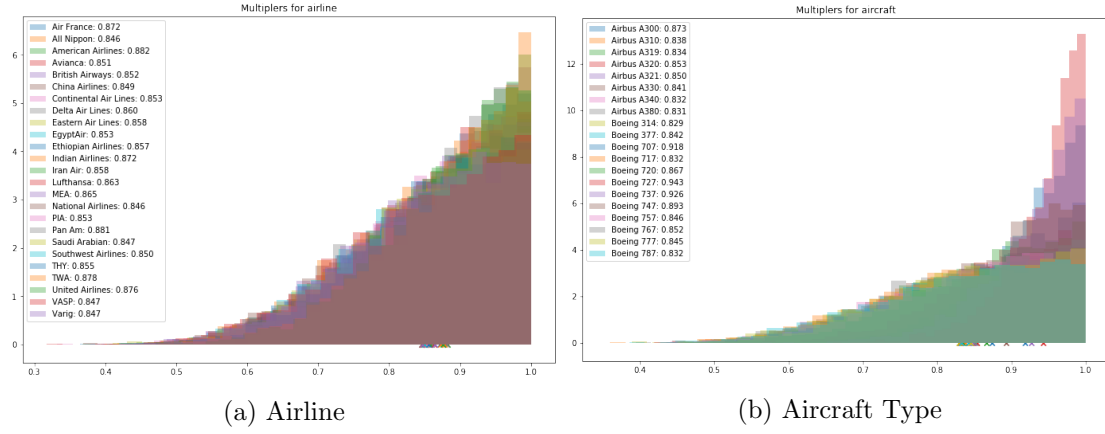


Figure 13: Posteriors Of Scaling Factor - Hull-Loss Risk

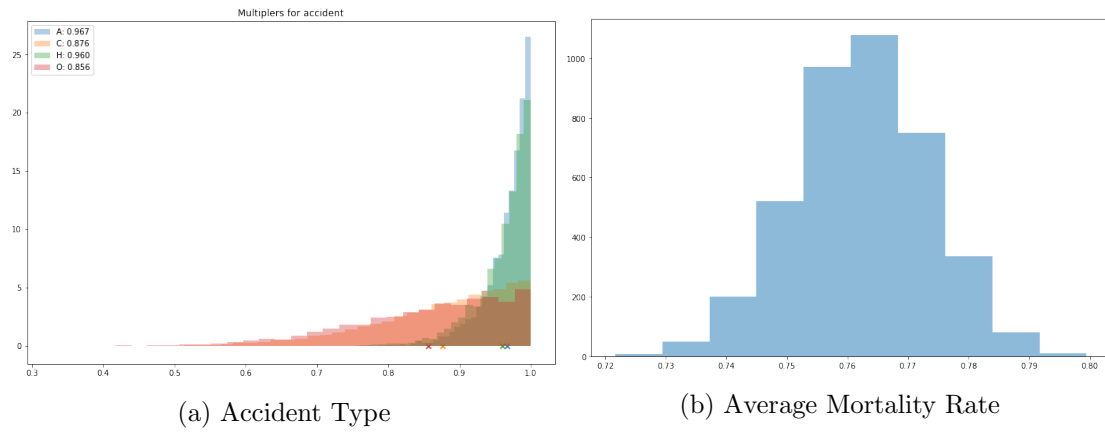


Figure 14: Posteriors Of Scaling Factor - Hull-Loss Risk

to have more variability in terms of being responsible for hull-loss, like C-type accident, which is criminal occurrence, seems to have a high variance, which makes sense since extreme cases like 9/11 is rare).

APPENDIX

Data

Original Data: <https://aviation-safety.net/database/>

Cleaned Data: https://docs.google.com/spreadsheets/d/1KIOE4XdHbQHs5tCCoYuYpMjXw_s-0ME32gQ2Fs0zWbc/edit?usp=sharing

List of aircraft by type: https://en.wikipedia.org/wiki/List_of_aircraft_by_date_and_usage_category

List of commercial airline: https://en.wikipedia.org/wiki/List_of_fighter_aircraft

Full code

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import numpy as np
4 import pystan
5 from sklearn.linear_model import LinearRegression, BayesianRidge
6 from sklearn.preprocessing import MinMaxScaler
7
8 data = pd.read_csv('/Users/ash/Downloads/clean.csv')
9
10 def clean_day_year(obj):
11     return int(obj)
12
13 def clean_month(obj):
14     month = str(obj).lower()
15     if month == 'jan':
16         month = '01'
17     elif month == 'feb':
18         month = '02'
19     elif month == 'mar':
20         month = '03'
21     elif month == 'apr':
22         month = '04'
23     elif month == 'may':
24         month = '05'
25     elif month == 'jun':
26         month = '06'
27     elif month == 'jul':
28         month = '07'
29     elif month == 'aug':
30         month = '08'
31     elif month == 'sep':
32         month = '09'
33     elif month == 'oct':
34         month = '10'
```



```

35     elif month == 'nov':
36         month = '11'
37     elif month == 'dec':
38         month = '12'
39     return month
40
41 def clean_fat(obj):
42     fat = str(obj)
43     if fat.find('+') == -1:
44         return int(fat)
45     elif fat.find('+') != "+":
46         total = int(fat[0:fat.find('+')]) + int(fat[fat.find('+')+2:])
47         return int(total)
48
49 data = data.dropna()
50 data['day'] = data['day'].apply(clean_day_year)
51 data['year'] = data['year'].apply(clean_day_year)
52 data['month'] = data['month'].apply(clean_month)
53 data['fat.'] = data['fat.'].apply(clean_fat)
54
55 data['date'] = pd.to_datetime(data[['day', 'month', 'year']])
56 data.head()
57
58 com = data.loc[data['Aircraft Cat'] == 'Commercial']
59 mil = data.loc[data['Aircraft Cat'] == 'Military']
60 print('Commercial Size:', com.shape)
61 print('Miliatary Size:', mil.shape)
62
63 #general trend by
64 year_count_com = (com['year'].value_counts()).reset_index()
65 year_count_com = year_count_com.sort_values(by=['index'], kind='mergesort')
66 year_count_com = year_count_com.reset_index(drop=True)
67
68 year_count_mil = mil['year'].value_counts().reset_index()
69 year_count_mil = year_count_mil.sort_values(by=['index'], kind='mergesort')
70 year_count_mil = year_count_mil.reset_index(drop=True)
71
72 plt.figure(figsize=(12,8))
73 plt.subplot(2,1,1)
74 plt.scatter(range(year_count_com['index'].iloc[0], year_count_com['index'].
75                 iloc[-1]+1),\
76             year_count_com['year'], color='b', alpha=0.7)
77 plt.title('Commercial Trend {} to {}'.format(year_count_com['index'].iloc
78         [0], year_count_com['index'].iloc[-1]))
79 plt.subplot(2,1,2)
80 plt.scatter(range(year_count_mil['index'].iloc[0], year_count_mil['index'].
81                 iloc[-1]+1),\
82             year_count_mil['year'], color='r', alpha=0.7)
83 plt.title('Military Trend {} to {}'.format(year_count_mil['index'].iloc[0],
84         year_count_mil['index'].iloc[-1]))
85 plt.show()
86
87 year_count_com['abs_year'] = year_count_com['index']
88 year_count_mil['abs_year'] = year_count_mil['index']

```

```

85 def visualize_posterior(samples, names):
86     for _ in names:
87         plt.figure(figsize=(12,4))
88         plt.hist(samples[_])
89         plt.title('Posterior samples of '+_)
90     plt.show()
91
92 scaler = MinMaxScaler()
93 N = 70
94
95 year_count_com['index'] = scaler.fit_transform(np.array(year_count_com['
    index']).reshape(-1,1))
96 reg = BayesianRidge()
97 reg.fit(np.array(year_count_com[['index']].iloc[:N,:]), year_count_com['
    year'].iloc[:N])
98 plt.figure(figsize=(12,8))
99 x = year_count_com['abs_year']
100 plt.scatter(x[:N], year_count_com['year'][:N], color='b', alpha=0.7, label=
    'Data Used In Modeling')
101 plt.scatter(x[N:], year_count_com['year'][N:], color='b', marker='x', alpha
    =0.7, label='Data Unseen')
102 y_m, y_std = reg.predict(np.array(year_count_com[['index']] ), return_std=
    True)
103 plt.plot(x, np.round(y_m), color='b', label='Mean Prediction')
104 plt.plot(x, np.round(y_m+2.96*y_std), color='r', linestyle='--', label='95%
    CI')
105 plt.plot(x, np.round(y_m-2.96*y_std), color='r', linestyle='--')
106
107 f_x = np.linspace(1,1.5,50)
108 f_y_m, f_y_std = reg.predict(f_x.reshape(-1,1), return_std=True)
109 f_x = range(2018, 2018+50)
110 plt.plot(f_x, np.round(f_y_m), color='b')
111 plt.plot(f_x, np.round(f_y_m+2.96*f_y_std), color='r', linestyle='--')
112 plt.plot(f_x, np.round(f_y_m-2.96*f_y_std), color='r', linestyle='--')
113
114 plt.axvline(x[N], color='g')
115 plt.legend()
116 plt.show()
117
118 model_trend = """
119
120 data {
121     //Data and its length
122     int<lower=1> length;
123     real<lower=0> data_[length];
124     //Number of generated values
125     int<lower=0> n_gen;
126 }
127
128 parameters {
129     //Linear trend model, with 2 coefficients
130     real coef_1;
131     real coef_2;
132     real<lower=0> noise;

```

```

133 }
134
135 model {
136     #Each coefficient is drawn from a broad Cauchy
137     coef_1 ~ cauchy(0,1);
138     coef_2 ~ cauchy(0,1);
139     noise ~ cauchy(0,1);
140     for(i in 1:length) {
141         data_[i] ~ normal(coef_1*i + coef_2, noise);
142     }
143 }
144
145 generated quantities {
146     real data_gen[n_gen];
147     for(i in 1:n_gen) {
148         data_gen[i] = normal_rng(coef_1*(i+length) + coef_2, noise);
149     }
150 }
151 """
152 stan_model_trend = pystan.StanModel(model_code=model_trend)
153
154 stan_data = {
155     'length': year_count_com['year'][:N].shape[0],
156     'data_': np.array(year_count_com['year'][:N]),
157     'n_gen': year_count_com['year'][N:].shape[0] + 50
158 }
159 results = stan_model_trend.sampling(data=stan_data)
160 samples = results.extract()
161
162 paras = ['coef_1', 'coef_2', 'noise']
163 print(results.stansummary(pars=paras))
164
165 #From CS146 – 14.1 Pre-class, Professor Carl Scheffler
166 def plot_acf(x):
167     from scipy import signal
168     plt.acorr(
169         x, maxlags=20, detrend=lambda x: signal.detrend(x, type='constant')
170     )
171
172 for param in paras:
173     plt.figure(figsize=(12, 4))
174     plot_acf(samples[param])
175     plt.title(f'Autocorrelation of {param} samples')
176
177 plt.show()
178
179 visualize_posterior(samples, paras)
180
181 future = samples['data_gen']
182 f_interval = np.percentile(future, axis=0, q=[2.5, 97.5])
183 plt.figure(figsize=(12,8))
184 x = year_count_com['abs_year']
185 plt.scatter(x[:N], year_count_com['year'][:N], color='b', alpha=0.7, label=

```

```

    'Data Used In Modeling')
186 plt.scatter(x[N:], year_count_com['year'][N:], color='b', marker='x', alpha
    =0.7, label='Data Unseen')
187 x_ = range(1, year_count_com['index'].shape[0]+1)
188 y = np.round([x_[i]*np.mean(samples['coef_1'])+np.mean(samples['coef_2'])
    for i in range(len(x))])
189 plt.plot(year_count_com['abs_year'], y)
190 plt.scatter(x[N]+range(stan_data['n_gen']), np.round(f_interval[0,:]),
    color='r', alpha=0.7,\
191             label='95% CI Of Sampled Data')
192 plt.scatter(x[N]+range(stan_data['n_gen']), np.round(f_interval[1,:]),
    color='r', alpha=0.7)
193 plt.axvline(x[N], color='g')
194 plt.legend()
195 plt.show()
196
197 scaler = MinMaxScaler()
198 N = 70
199
200 year_count_mil['index'] = scaler.fit_transform(np.array(year_count_mil['
    index']).reshape(-1,1))
201 reg = BayesianRidge()
202 reg.fit(np.array(year_count_mil[['index']].iloc[:N,:]), year_count_mil['
    year'].iloc[:N])
203 plt.figure(figsize=(12,8))
204 x = year_count_mil['abs_year']
205 plt.scatter(x[:N], year_count_mil['year'][:N], color='b', alpha=0.7, label=
    'Data Used In Modeling')
206 plt.scatter(x[N:], year_count_mil['year'][N:], color='b', marker='x', alpha
    =0.7, label='Data Unseen')
207 y_m, y_std = reg.predict(np.array(year_count_mil[['index']] ), return_std=
    True)
208 plt.plot(x, np.round(y_m), color='b', label='Mean Prediction')
209 plt.plot(x, np.round(y_m+2.96*y_std), color='r', linestyle='--', label='95%
    CI')
210 plt.plot(x, np.round(y_m-2.96*y_std), color='r', linestyle='--')
211
212 f_x = np.linspace(1,1.5,50)
213 f_y_m, f_y_std = reg.predict(f_x.reshape(-1,1), return_std=True)
214 f_x = range(2018, 2018+50)
215 plt.plot(f_x, np.round(f_y_m), color='b')
216 plt.plot(f_x, np.round(f_y_m+2.96*f_y_std), color='r', linestyle='--')
217 plt.plot(f_x, np.round(f_y_m-2.96*f_y_std), color='r', linestyle='--')
218
219 plt.axvline(x[N], color='g')
220 plt.legend()
221 plt.show()
222
223 stan_data = {
224     'length': year_count_mil['year'][:N].shape[0],
225     'data_': np.array(year_count_mil['year'][:N]),
226     'n_gen': year_count_mil['year'][N:].shape[0] + 50
227 }
228 results = stan_model_trend.sampling(data=stan_data)

```

```

229 samples = results.extract()
230
231 paras =['coef_1', 'coef_2', 'noise']
232 print(results.stansummary(pars=paras))
233
234 for param in paras:
235     plt.figure(figsize=(12, 4))
236     plot_acf(samples[param])
237     plt.title(f'Autocorrelation of {param} samples')
238
239 plt.show()
240
241 visualize_posterior(samples, paras)
242
243 future = samples['data_gen']
244 f_interval = np.percentile(future, axis=0, q=[2.5, 97.5])
245 plt.figure(figsize=(12,8))
246 x = year_count_mil['abs_year']
247 plt.scatter(x[:N], year_count_mil['year'][:N], color='b', alpha=0.7, label=
    'Data Used In Modeling')
248 plt.scatter(x[N:], year_count_mil['year'][N:], color='b', marker='x', alpha
    =0.7, label='Data Unseen')
249 x_ = range(1, year_count_mil['index'].shape[0]+1)
250 y = np.round([x_[i]*np.mean(samples['coef_1'])+np.mean(samples['coef_2'])
    for i in range(len(x))])
251 plt.plot(year_count_mil['abs_year'], y)
252 plt.scatter(x[N]+range(stan_data['n_gen']), np.round(f_interval[0,:]),
    color='r', alpha=0.7,\
253     label='95% CI Of Sampled Data')
254 plt.scatter(x[N]+range(stan_data['n_gen']), np.round(f_interval[1,:]),
    color='r', alpha=0.7)
255 plt.axvline(x[N], color='g')
256 plt.legend()
257 plt.show()
258
259 scaler = MinMaxScaler()
260 N = 70
261
262 year_count_com['index'] = scaler.fit_transform(np.array(year_count_com['
    index']).reshape(-1,1))
263 year_count_com['index^2'] = year_count_com['index']**2
264 reg = BayesianRidge()
265 reg.fit(np.array(year_count_com[['index', 'index^2']].iloc[:N,:]),
    year_count_com['year'].iloc[:N])
266 plt.figure(figsize=(12,8))
267 x = year_count_com['abs_year']
268 plt.scatter(x[:N], year_count_com['year'][:N], color='b', alpha=0.7, label=
    'Data Used In Modeling')
269 plt.scatter(x[N:], year_count_com['year'][N:], color='b', marker='x', alpha
    =0.7, label='Data Unseen')
270 y_m, y_std = reg.predict(np.array(year_count_com[['index', 'index^2']]),
    return_std=True)
271 plt.plot(x, np.round(y_m), label='Mean Prediction', color='b')
272 plt.plot(x, np.round(y_m+2.96*y_std), color='r', linestyle='—', label='95%

```

```

    CI')
273 plt.plot(x, np.round(y_m-2.96*y_std), color='r', linestyle='--')
274
275 f_x = np.expand_dims(np.linspace(1,1.5,50), axis=1)
276 f_x = np.concatenate((f_x, f_x**2), axis=1)
277 f_y_m, f_y_std = reg.predict(f_x, return_std=True)
278 f_x = range(2018, 2018+50)
279 plt.plot(f_x, np.round(f_y_m), color='b')
280 plt.plot(f_x, np.round(f_y_m+2.96*f_y_std), color='r', linestyle='--')
281 plt.plot(f_x, np.round(f_y_m-2.96*f_y_std), color='r', linestyle='--')
282
283 plt.axvline(x[N], color='g')
284 plt.legend()
285 plt.show()
286
287 model_trend = """
288
289 data {
290   //Data and its length
291   int<lower=1> length;
292   real<lower=0> data_[length];
293   //Number of generated values
294   int<lower=0> n_gen;
295 }
296
297 parameters {
298   //Quadratic trend model, with 3 coefficients
299   real coef_1;
300   real coef_2;
301   real coef_3;
302   real<lower=0> noise;
303 }
304
305 model {
306   #Each coefficient is drawn from a broad Cauchy
307   coef_1 ~ cauchy(0,1);
308   coef_2 ~ cauchy(0,1);
309   coef_3 ~ cauchy(0,1);
310   for(i in 1:length) {
311     data_[i] ~ normal(coef_1*i*i + coef_2*i + coef_3, noise);
312   }
313 }
314
315 generated quantities {
316   real data_gen[n_gen];
317   for(i in 1:n_gen) {
318     data_gen[i] = normal_rng(coef_1*(i+length)*(i+length) + coef_2*(i+
length) + coef_3, noise);
319   }
320 }
321 """
322 stan_model_trend = pystan.StanModel(model_code=model_trend)
323
324 stan_data = {

```

```

325     'length': year_count_com['year'][:N].shape[0],
326     'data_': np.array(year_count_com['year'][:N]),
327     'n_gen': year_count_com['year'][N:].shape[0]+50
328 }
329 results = stan_model.trend.sampling(data=stan_data)
330 samples = results.extract()
331
332 paras = ['coef_1', 'coef_2', 'coef_3', 'noise']
333 print(results.stansummary(pars=paras))
334 for param in paras:
335     plt.figure(figsize=(12, 4))
336     plot_acf(samples[param])
337     plt.title(f'Autocorrelation of {param} samples')
338
339 plt.show()
340
341 visualize_posterior(samples, paras)
342
343 future = samples['data_gen']
344 f_interval = np.percentile(future, axis=0, q=[2.5, 97.5])
345 plt.figure(figsize=(12,8))
346 x = year_count_com['abs_year']
347 plt.scatter(x[:N], year_count_com['year'][:N], color='b', alpha=0.7, label=
    'Data Used In Modeling')
348 plt.scatter(x[N:], year_count_com['year'][N:], color='b', marker='x', alpha
    =0.7, label='Data Unseen')
349 x_ = range(1, year_count_com['index'].shape[0]+1)
350 y = np.round((x_[i]**2*np.mean(samples['coef_1'])+x_[i]*np.mean(samples['
    coef_2']) + np.mean(samples['coef_3']))\
351             for i in range(len(x)))
352 plt.plot(year_count_com['abs_year'], y)
353 plt.scatter(x[N]+range(stan_data['n_gen']), np.round(f_interval[0,:]),
    color='r', alpha=0.7,\
354             label='95% CI Of Sampled Data')
355 plt.scatter(x[N]+range(stan_data['n_gen']), np.round(f_interval[1,:]),
    color='r', alpha=0.7)
356 plt.axvline(x[N], color='g')
357 plt.legend()
358 plt.show()
359
360 scaler = MinMaxScaler()
361 N = 70
362
363 year_count_mil['index'] = scaler.fit_transform(np.array(year_count_mil['
    index']).reshape(-1,1))
364 year_count_mil['index^2'] = year_count_mil['index']**2
365 reg = BayesianRidge()
366 reg.fit(np.array(year_count_mil[['index', 'index^2']].iloc[:N,:]),
    year_count_mil['year'].iloc[:N])
367 plt.figure(figsize=(12,8))
368 x = year_count_mil['abs_year']
369 plt.scatter(x[:N], year_count_mil['year'][:N], color='b', alpha=0.7, label=
    'Data Used In Modeling')
370 plt.scatter(x[N:], year_count_mil['year'][N:], color='b', marker='x', alpha

```

```

    =0.7, label='Data Unseen')
371 y_m, y_std = reg.predict(np.array(year_count_mil[['index', 'index^2']]),
    return_std=True)
372 plt.plot(x, np.round(y_m), label='Mean Prediction', color='b')
373 plt.plot(x, np.round(y_m+2.96*y_std), color='r', linestyle='--', label='95%
    CI')
374 plt.plot(x, np.round(y_m-2.96*y_std), color='r', linestyle='--')
375
376 f_x = np.expand_dims(np.linspace(1,1.5,50), axis=1)
377 f_x = np.concatenate((f_x, f_x**2), axis=1)
378 f_y_m, f_y_std = reg.predict(f_x, return_std=True)
379 f_x = range(2018, 2018+50)
380 plt.plot(f_x, np.round(f_y_m), color='b')
381 plt.plot(f_x, np.round(f_y_m+2.96*f_y_std), color='r', linestyle='--')
382 plt.plot(f_x, np.round(f_y_m-2.96*f_y_std), color='r', linestyle='--')
383
384 plt.axvline(x[N], color='g')
385 plt.legend()
386 plt.show()
387
388 stan_data = {
389     'length': year_count_mil['year'][:N].shape[0],
390     'data_': np.array(year_count_mil['year'][:N]),
391     'n_gen': year_count_mil['year'][N:].shape[0]+50
392 }
393 results = stan_model.trend.sampling(data=stan_data)
394 samples = results.extract()
395
396 paras =['coef_1', 'coef_2', 'coef_3', 'noise']
397 print(results.stansummary(pars=paras))
398
399 #From CS146 - 14.1 Pre-class, Professor Carl Scheffler
400 def plot_acf(x):
401     from scipy import signal
402     plt.acorr(
403         x, maxlags=20, detrend=lambda x: signal.detrend(x, type='constant')
404     )
405
406 for param in paras:
407     plt.figure(figsize=(12, 4))
408     plot_acf(samples[param])
409     plt.title(f'Autocorrelation of {param} samples')
410
411 plt.show()
412
413 visualize_posterior(samples, paras)
414
415 future = samples['data_gen']
416 f_interval = np.percentile(future, axis=0, q=[2.5, 97.5])
417 plt.figure(figsize=(12,8))
418 x = year_count_mil['abs_year']
419 plt.scatter(x[:N], year_count_mil['year'][:N], color='b', alpha=0.7, label=
    'Data Used In Modeling')

```



```

420 plt.scatter(x[N:], year_count_mil['year'][N:], color='b', marker='x', alpha
    =0.7, label='Data Unseen')
421 x_ = range(1, year_count_mil['index'].shape[0]+1)
422 y = np.round([x_[i]**2*np.mean(samples['coef_1'])+x_[i]*np.mean(samples['
    coef_2'])+np.mean(samples['coef_3'])\
423               for i in range(len(x))])
424 plt.plot(year_count_mil['abs_year'], y)
425 plt.scatter(x[N+range(stan_data['n_gen'])], np.round(f_interval[0,:]),
    color='r', alpha=0.7,\
426               label='95% CI Of Sampled Data')
427 plt.scatter(x[N+range(stan_data['n_gen'])], np.round(f_interval[1,:]),
    color='r', alpha=0.7)
428 plt.axvline(x[N], color='g')
429 plt.legend()
430 plt.show()
431
432 scaler = MinMaxScaler()
433 N = 70
434
435 #Using sklearn rbf kernel
436 from sklearn.metrics.pairwise import rbf_kernel
437
438 #100 pivot points
439 kernel = np.linspace(0,2,100).reshape(-1,1)
440 #Rescaling the year to the range 0 to 1, a standard procedure before
    fitting
441 year_count_com['index'] = scaler.fit_transform(np.array(year_count_com['
    index']).reshape(-1,1))
442
443 X_train = rbf_kernel(np.array(year_count_com['index'][:N]).reshape(-1,1),
    kernel, gamma=10)
444
445 #Using Bayesian model and plotting the results
446 reg = BayesianRidge()
447 reg.fit(X_train, year_count_com['year'].iloc[:N])
448 plt.figure(figsize=(12,8))
449 x = year_count_com['abs_year']
450 plt.scatter(x[:N], year_count_com['year'][:N], color='b', alpha=0.7, label=
    'Data Used In Modeling')
451 plt.scatter(x[N:], year_count_com['year'][N:], color='b', marker='x', alpha
    =0.7, label='Data Unseen')
452
453 X_test = rbf_kernel(np.array(year_count_com['index']).reshape(-1,1),\
454                     kernel, gamma=10)
455 y_m, y_std = reg.predict(X_test, return_std=True)
456 plt.plot(x, np.round(y_m), color='b', label='Mean Prediction')
457 plt.plot(x, np.round(y_m+2.96*y_std), color='r', linestyle='--', label='95%
    CI')
458 plt.plot(x, np.round(y_m-2.96*y_std), color='r', linestyle='--')
459
460 f_x = np.linspace(1,1.5,50).reshape(-1,1)
461 K_f_x = rbf_kernel(f_x, kernel, gamma=10)
462 f_y_m, f_y_std = reg.predict(K_f_x, return_std=True)
463 f_x = range(2018, 2018+50)

```

```

464 plt.plot(f_x, np.round(f_y_m), color='b')
465 plt.plot(f_x, np.round(f_y_m+2.96*f_y_std), color='r', linestyle='—')
466 plt.plot(f_x, np.round(f_y_m-2.96*f_y_std), color='r', linestyle='—')
467
468 plt.axvline(x[N], color='g')
469 plt.legend()
470 plt.show()
471
472 scaler = MinMaxScaler()
473 N = 70
474
475 from sklearn.metrics.pairwise import rbf_kernel
476
477 kernel = np.linspace(0,2,100).reshape(-1,1)
478 year_count_mil['index'] = scaler.fit_transform(np.array(year_count_mil['
index']).reshape(-1,1))
479
480 X_train = rbf_kernel(np.array(year_count_mil['index'][:N]).reshape(-1,1),
kernel, gamma=10)
481
482 reg = BayesianRidge()
483 reg.fit(X_train, year_count_mil['year'].iloc[:N])
484 plt.figure(figsize=(12,8))
485 x = year_count_mil['abs_year']
486 plt.scatter(x[:N], year_count_mil['year'][:N], color='b', alpha=0.7, label=
'Data Used In Modeling')
487 plt.scatter(x[N:], year_count_mil['year'][N:], color='b', marker='x', alpha
=0.7, label='Data Unseen')
488
489 X_test = rbf_kernel(np.array(year_count_mil['index']).reshape(-1,1),\
kernel, gamma=10)
490
491 y_m, y_std = reg.predict(X_test, return_std=True)
492 plt.plot(x, np.round(y_m), color='b', label='Mean Prediction')
493 plt.plot(x, np.round(y_m+2.96*y_std), color='r', linestyle='—', label='95%
CI')
494 plt.plot(x, np.round(y_m-2.96*y_std), color='r', linestyle='—')
495
496 f_x = np.linspace(1,1.5,50).reshape(-1,1)
497 K_f_x = rbf_kernel(f_x, kernel, gamma=10)
498 f_y_m, f_y_std = reg.predict(K_f_x, return_std=True)
499 f_x = range(2018, 2018+50)
500 plt.plot(f_x, np.round(f_y_m), color='b')
501 plt.plot(f_x, np.round(f_y_m+2.96*f_y_std), color='r', linestyle='—')
502 plt.plot(f_x, np.round(f_y_m-2.96*f_y_std), color='r', linestyle='—')
503
504 plt.axvline(x[N], color='g')
505 plt.legend()
506 plt.show()
507
508 model_trend_main = """
509
510 data {
511 //Data and its length
512     int<lower=1> length;

```

```

513     real<lower=0> data_[length];
514 //Number of radial basis points or number of coefficients
515     int<lower=1> c;
516 //Kernelized value of the training data
517     vector[c] rbf_train[length];
518 //Number of generated values
519     int<lower=0> n_gen;
520 //Kernelized value of the generated data
521     vector[c] rbf_gen[n_gen];
522 }
523
524 parameters {
525 //RBF model, with c basis points
526     row_vector[c] coef_;
527 //Intercept
528     real intercept;
529 //Gaussian noise
530     real<lower=0> noise;
531 }
532
533 model {
534 //Each coefficient is drawn from a broad Cauchy
535     for(j in 1:c) {
536         coef_[j] ~ cauchy(0,1);
537     }
538     for(i in 1:length) {
539         data_[i] ~ normal((coef_*rbf_train[i]) + intercept, noise);
540     }
541 }
542
543 generated quantities {
544     real data_gen[n_gen];
545     for(i in 1:n_gen) {
546         data_gen[i] = normal_rng((coef_*rbf_gen[i]) + intercept, noise);
547     }
548 }
549 """
550 trend_model = pystan.StanModel(model_code=model_trend_main)
551
552 kernel = np.linspace(0,2,100).reshape(-1,1)
553 X_train = rbf_kernel(np.array(year_count_com['index'][:N]).reshape(-1,1),
554                       kernel, gamma=10)
555 X_gen = rbf_kernel(np.array(year_count_com['index'].iloc[N:]).reshape(-1,1),
556                    kernel, gamma=10)
557 f_X = np.linspace(1,1.5,50).reshape(-1,1)
558 K_f_X = rbf_kernel(f_X, kernel, gamma=10)
559 X_gen = np.concatenate((X_gen, K_f_X), axis=0)
560
561 stan_data = {
562     'length': year_count_com['year'][:N].shape[0],
563     'data_': np.array(year_count_com['year'][:N]),
564     'c': 100,
565     'rbf_train': X_train,
566     'n_gen': year_count_com['year'][N:].shape[0] + 50,

```

```

565     'rbf_gen': X_gen
566 }
567 results = trend_model.sampling(data=stan_data)
568 samples = results.extract()
569
570 paras = ['coef_', 'intercept', 'noise']
571 print(results.stansummary(pars=paras))
572
573 future = samples['data_gen']
574 f_interval = np.percentile(future, axis=0, q=[2.5, 97.5])
575 plt.figure(figsize=(12,8))
576 x = year_count_com['abs_year']
577 plt.scatter(x[:N], year_count_com['year'][:N], color='b', alpha=0.7, label=
    'Data Used In Modeling')
578 plt.scatter(x[N:], year_count_com['year'][N:], color='b', marker='x', alpha
    =0.7, label='Data Unseen')
579 x_ = range(1, year_count_com['index'].shape[0]+1)
580 # y = np.round([x_[i]**2*np.mean(samples['coef_1'])+x_[i]*np.mean(samples['
    coef_2']) + np.mean(samples['coef_3'])]\
581 #               for i in range(len(x))])
582 # plt.plot(year_count_com['abs_year'], y)
583 plt.scatter(x[N]+range(stan_data['n_gen']), np.round(f_interval[0,:]),
    color='r', alpha=0.7,\
584             label='95% CI Of Sampled Data')
585 plt.scatter(x[N]+range(stan_data['n_gen']), np.round(f_interval[1,:]),
    color='r', alpha=0.7)
586 plt.axvline(x[N], color='g')
587 plt.legend()
588 plt.show()
589
590 risk_com = com[['year', 'type', 'operator', 'fat.', 'cat']].reset_index(
    drop=True)
591
592 def clean_cat_1(obj):
593     obj = str(obj)
594     return obj[0]
595
596 def clean_cat_2(obj):
597     obj = str(obj)
598     return obj[1]
599
600 risk_com['Accident Type'] = risk_com['cat']
601 risk_com['Loss Type'] = risk_com['cat']
602 risk_com = risk_com.drop(['cat'], axis=1)
603 risk_com['Accident Type'] = risk_com['Accident Type'].apply(clean_cat_1)
604 risk_com['Loss Type'] = risk_com['Loss Type'].apply(clean_cat_2)
605 risk_com = risk_com.sort_values(by=['year'], kind='mergesort').reset_index(
    drop=True)
606
607 def Boeing_Airbus(data):
608     df = pd.DataFrame()
609     for _ in range(data.shape[0]):
610         if ('Boeing' in data['type']).iloc[_]:
611             data.iloc[_ , 1] = data.iloc[_ , 1][:10]

```

```

612         if data.iloc[-, 1] == 'Boeing':
613             pass
614         else:
615             df = pd.concat((df, data.iloc[-,:]), axis=1, sort=False)
616     elif ('Airbus' in data['type'].iloc[-]):
617         data.iloc[-, 1] = data.iloc[-, 1][11]
618         if data.iloc[-, 1] == 'Airbus':
619             pass
620         else:
621             df = pd.concat((df, data.iloc[-,:]), axis=1, sort=False)
622     return df.transpose().reset_index(drop=True)
623
624 def Top_Airline(data, top_airlines):
625     df = pd.DataFrame()
626     for _ in range(data.shape[0]):
627         if data['operator'].iloc[_] in top_airlines:
628             df = pd.concat((df, data.iloc[-,:]), axis=1, sort=False)
629     return df.transpose().reset_index(drop=True)
630
631 def how_many_passenger(data):
632     data['total'] = np.nan
633     for _ in range(data.shape[0]):
634         if data['type'].iloc[_] == 'Boeing 727':
635             data['total'].iloc[_] = 189
636         elif data['type'].iloc[_] == 'Boeing 737':
637             data['total'].iloc[_] = 143
638         elif data['type'].iloc[_] == 'Boeing 707':
639             data['total'].iloc[_] = 189
640         elif data['type'].iloc[_] == 'Boeing 747':
641             data['total'].iloc[_] = 660
642         elif data['type'].iloc[_] == 'Boeing 720':
643             data['total'].iloc[_] = 219
644         elif data['type'].iloc[_] == 'Boeing 757':
645             data['total'].iloc[_] = 295
646         elif data['type'].iloc[_] == 'Boeing 767':
647             data['total'].iloc[_] = 351
648         elif data['type'].iloc[_] == 'Boeing 777':
649             data['total'].iloc[_] = 451
650         elif data['type'].iloc[_] == 'Boeing 377':
651             data['total'].iloc[_] = 114
652         elif data['type'].iloc[_] == 'Boeing 717':
653             data['total'].iloc[_] = 134
654         elif data['type'].iloc[_] == 'Boeing 787':
655             data['total'].iloc[_] = 335
656         elif data['type'].iloc[_] == 'Boeing 314':
657             data['total'].iloc[_] = 77
658         elif data['type'].iloc[_] == 'Airbus A300':
659             data['total'].iloc[_] = 300
660         elif data['type'].iloc[_] == 'Airbus A320':
661             data['total'].iloc[_] = 186
662         elif data['type'].iloc[_] == 'Airbus A321':
663             data['total'].iloc[_] = 240
664         elif data['type'].iloc[_] == 'Airbus A330':
665             data['total'].iloc[_] = 335

```

```

666     elif data['type'].iloc[-] == 'Airbus A310':
667         data['total'].iloc[-] = 220
668     elif data['type'].iloc[-] == 'Airbus A319':
669         data['total'].iloc[-] = 160
670     elif data['type'].iloc[-] == 'Airbus A340':
671         data['total'].iloc[-] = 475
672     elif data['type'].iloc[-] == 'Airbus A380':
673         data['total'].iloc[-] = 853
674     return data
675
676 risk_com_BA = Boeing_Airbus(risk_com)
677 top_25_airlines = risk_com_BA['operator'].value_counts()[ :25].index
678 risk_data = Top_Airline(risk_com_BA, top_25_airlines)
679 risk_data = how_many_passenger(risk_data)
680
681 risk_data = risk_data.drop([505, 504], axis=0).reset_index(drop=True)
682
683 stan_model_risk = """
684
685 data {
686     //Number of airline
687     int<lower=1> L;
688     //Number of aircraft
689     int<lower=1> C;
690     //Number of accident type
691     int<lower=1> T;
692     //Number of datapoint
693     int<lower=1> N;
694
695     //A vector of 0/1 in each entry expressing which airline, aircraft and
696     accident type the datapoint is
697     vector<lower=0>[L] airline[N];
698     vector<lower=0>[C] aircraft[N];
699     vector<lower=0>[T] accident[N];
700
701     //A vector expressing the number of passenger on that aircraft
702     int<lower=0> passenger[N];
703
704     //Number of fatality
705     int<lower=0> fat[N];
706 }
707
708 parameters {
709
710     //Scaling factor for the airline
711     row_vector<lower=0, upper=1>[L] m_al;
712     //Scaling factor for the aircraft
713     row_vector<lower=0, upper=1>[C] m_ac;
714     //Scaling factor for the accident type
715     row_vector<lower=0, upper=1>[T] m_a;
716
717     //Mortality rate
718     real<lower=0, upper=1> p;

```

```

719 }
720
721 model {
722
723     for (l in 1:L){
724         m_al[l] ~ lognormal(0,0.25);
725     }
726
727     for (c in 1:C){
728         m_ac[c] ~ lognormal(0,0.25);
729     }
730
731     for (t in 1:T){
732         m_a[t] ~ lognormal(0,0.25);
733     }
734
735     for(i in 1:N) {
736         #Sample the mortality rate from a beta distribution , with scaling
737         #factor from the airline , aircraft and accident
738         p ~ beta(((m_al*airline[i])+(m_ac*aircraft[i])+(m_a*accident[i])),
739         1);
740         fat[i] ~ binomial(passenger[i], p);
741     }
742 }
743
744 """
745 model_risk = pystan.StanModel(model_code=stan_model_risk)
746
747 def produce_stan_data(raw):
748     airline = pd.get_dummies(raw[ 'operator' ])
749     name_airline = airline.columns
750     airline = np.array(airline , dtype=int)
751
752     aircraft = pd.get_dummies(raw[ 'type' ])
753     name_aircraft = aircraft.columns
754     aircraft = np.array(aircraft , dtype=int)
755
756     accident = pd.get_dummies(raw[ 'Accident Type' ])
757     name_accident = accident.columns
758     accident = np.array(accident , dtype=int)
759
760     passenger = np.array(raw[ 'total' ], dtype=int)
761
762     fat = np.array(raw[ 'fat.' ], dtype=int)
763
764     hull = pd.get_dummies(raw[ 'Loss Type' ])
765     hull = np.array(hull.iloc[:,0], dtype=int)
766
767     data_dict = {
768         'L': airline.shape[1],
769         'C': aircraft.shape[1],
770         'T': accident.shape[1],
771         'N': airline.shape[0],

```

```

771     'airline': airline ,
772     'aircraft': aircraft ,
773     'accident': accident ,
774
775     'passenger': passenger ,
776
777     'fat': fat ,
778     'hull': hull
779
780 }
781
782 name_dict = {
783     'airline': name_airline ,
784     'aircraft': name_aircraft ,
785     'accident': name_accident
786 }
787 return data_dict , name_dict
788
789 def plot_posterior(results , name, what):
790     if what == 'airline':
791         what_ = 'm_al'
792     elif what == 'aircraft':
793         what_ = 'm_ac'
794     else:
795         what_ = 'm_a'
796     mul = results[what_]
797     plt.figure(figsize=(12,8))
798     for _ in range(mul.shape[1]):
799         plt.hist(mul[:,_], alpha=0.35, bins=30, density=True, label=str(
800 name[what][_])+': {0:.3f}'.format(mul[:,_].mean()))
801         plt.scatter(mul[:,_].mean(),0, marker='x', s=40)
802     plt.legend()
803     plt.title('Multipliers for {}'.format(what))
804     mul = mul.reshape(-1,1)
805     plt.figure(figsize=(12,8))
806     plt.hist(mul, alpha=0.35, bins=30, density=True, label='var: {0:.3f}'.
807 format(mul.var()))
808     plt.scatter(mul.mean(), 0, marker='x', s=40, label='mean: {0:.3f}'.
809 format(mul.mean()))
810     plt.title('All multipliers')
811     plt.legend()
812     plt.show()
813     print('95% CI for all {} multipliers is:'.format(what), (np.percentile(
814 mul,2.5),np.percentile(mul,97.5)))
815
816 stan_data , name = produce_stan_data(risk_data)
817 results = model_risk.sampling(data=stan_data , n_jobs=1)
818 results
819
820 plot_posterior(results , name, 'airline')
821 plot_posterior(results , name, 'aircraft')
822 plot_posterior(results , name, 'accident')
823
824 plt.figure(figsize=(12,8))

```



```

821 plt.hist(results['p'], alpha=0.5)
822 plt.show()
823
824 stan_model_risk_hull = """
825
826 data {
827 //Number of airline
828   int<lower=1> L;
829 //Number of aircraft
830   int<lower=1> C;
831 //Number of accident type
832   int<lower=1> T;
833 //Number of datapoint
834   int<lower=1> N;
835
836 //A vector of 0/1 in each entry expressing which airline , aircraft and
      accident type the datapoint is
837   vector<lower=0>[L] airline[N];
838   vector<lower=0>[C] aircraft[N];
839   vector<lower=0>[T] accident[N];
840
841 // Binary indicator of whether it's a hull loss or not
842   int<lower=0> hull[N];
843 }
844
845
846 parameters {
847
848 //Scaling factor for the airline
849   row_vector<lower=0, upper=1>[L] m_al;
850 //Scaling factor for the aircraft
851   row_vector<lower=0, upper=1>[C] m_ac;
852 //Scaling factor for the accident type
853   row_vector<lower=0, upper=1>[T] m_a;
854
855 //Mortality rate
856   real<lower=0, upper=1> p;
857 }
858
859 model {
860
861   for (l in 1:L){
862     m_al[l] ~ lognormal(0,0.25);
863   }
864
865   for (c in 1:C){
866     m_ac[c] ~ lognormal(0,0.25);
867   }
868
869   for (t in 1:T){
870     m_a[t] ~ lognormal(0,0.25);
871   }
872
873   for(i in 1:N) {

```

```

874     p ~ beta(((m_al*airline[i])+(m_ac*aircraft[i])+(m_a*accident[i])),
875             1);
876     hull[i] ~ bernoulli(p);
877 }
878
879 """
880 model_risk_hull = pystan.StanModel(model_code=stan_model_risk_hull)
881
882 stan_data, name = produce_stan_data(risk_data)
883 results = model_risk_hull.sampling(data=stan_data, n_jobs=1)
884 results
885 plot_posterior(results, name, 'airline')
886 plot_posterior(results, name, 'aircraft')
887 plot_posterior(results, name, 'accident')
888 plt.figure(figsize=(12,8))
889 plt.hist(results['p'], alpha=0.5)
890 plt.show()

```