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 slop, 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

.png

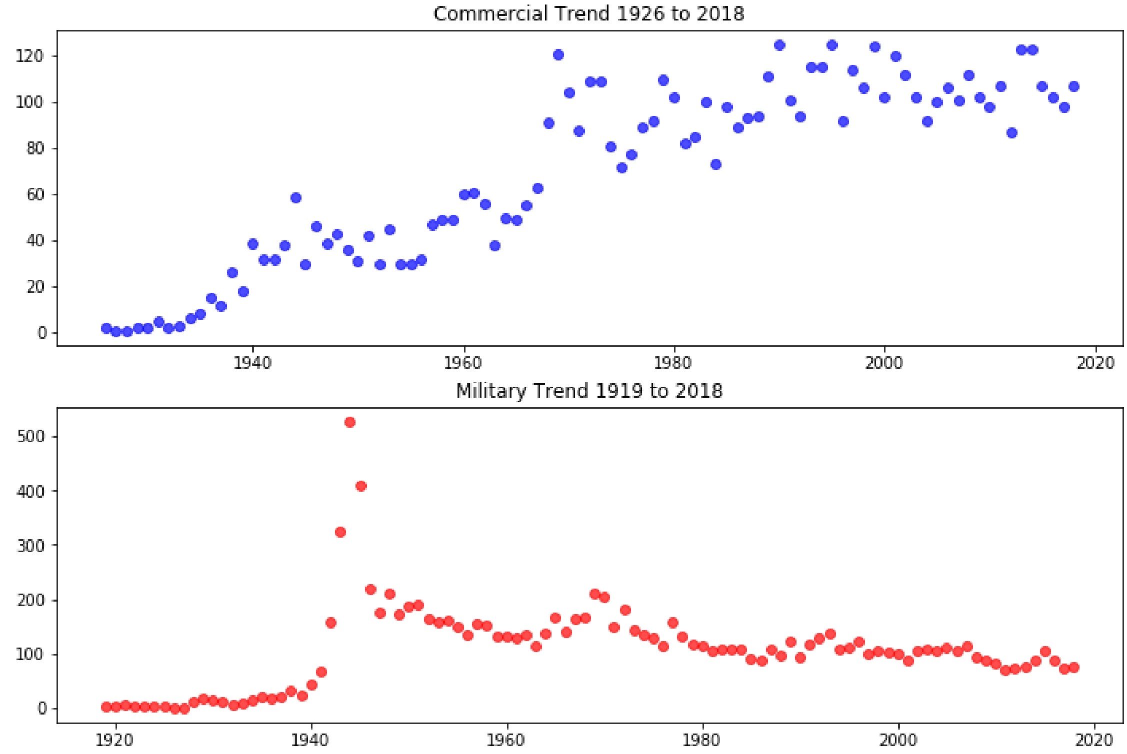
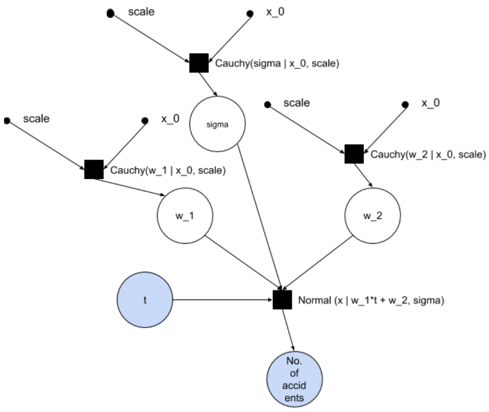


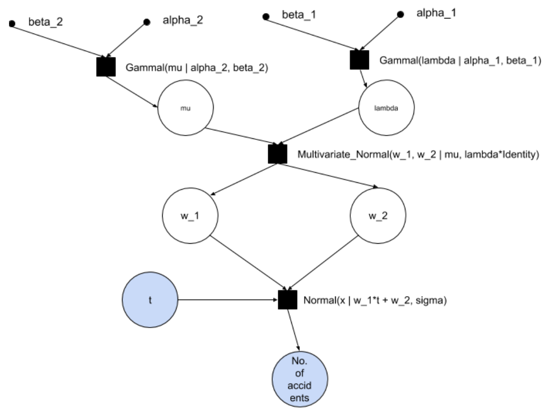
Figure 2: Number Of Accident Per Year, 1919 To 2018

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).

```
model_trend = """

data {
//Data and its length
    int<lower=1> length;
    real<lower=0> data_[length];
//Number of generated values
    int<lower=0> n_gen;
}

parameters {
    //Linear trend model, with 2 coefficients
    real coef_1;
    real coef_2;
    real<lower=0> noise;
}

model {
```



(a) Bayesian Linear Model



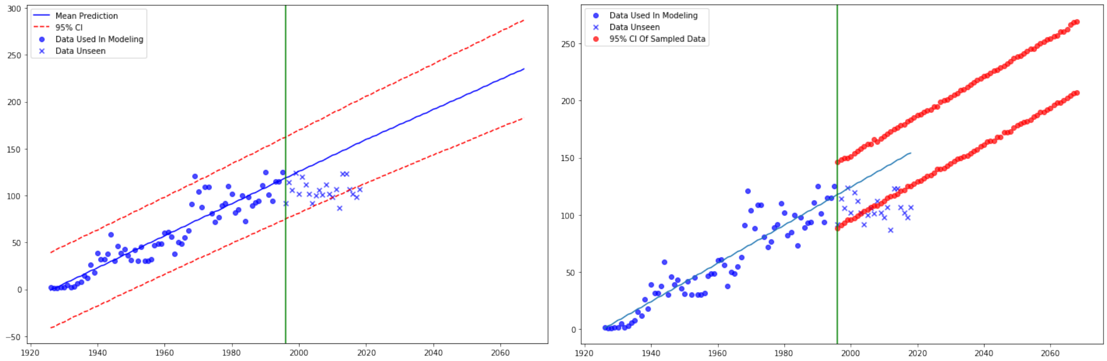
(b) Stan Model

Figure 3: Factor Graph Of Linear Models

```
    #Each coefficient is drawn from a broad Cauchy
    coef_1 ~ cauchy(0,1);
    coef_2 ~ cauchy(0,1);
    noise ~ cauchy(0,1);
    for(i in 1:length) {
        data_[i] ~ normal(coef_1*i + coef_2, noise);
    }
}

generated quantities {
    real data_gen[n_gen];
    for(i in 1:n_gen) {
        data_gen[i] = normal_rng(coef_1*(i+length) + coef_2, noise);
    }
}
"""
stan_model_trend = pystan.StanModel(model_code=model_trend)
```

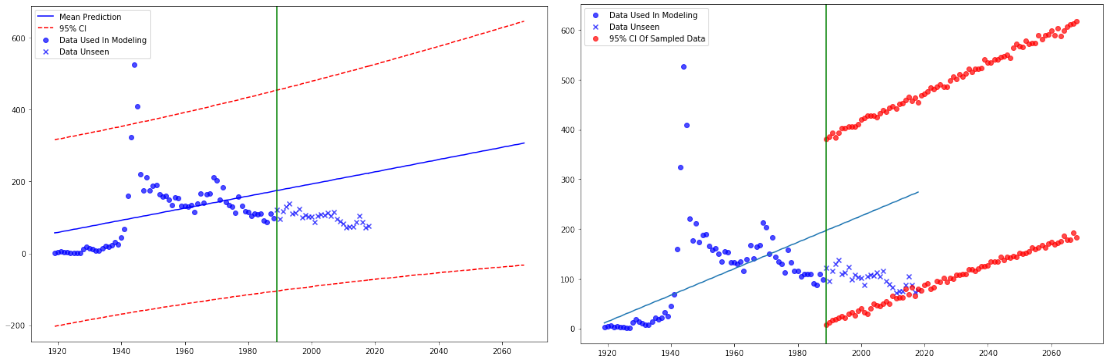
The Stan model (figure 3b), specified by code above, has a different priors. The slope, intercept and noise are generated by three different broad Cauchy since we assume we don't know the ranges of these parameters beforehand, and these Cauchy-s have specified hyperparameters. The estimations of the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



(a) Bayesian Linear Model (b) Stan Model

Figure 4: Predictions Of Linear Models - Commercial



(a) Bayesian Linear Model (b) Stan Model

Figure 5: Predictions Of Linear Models - Military

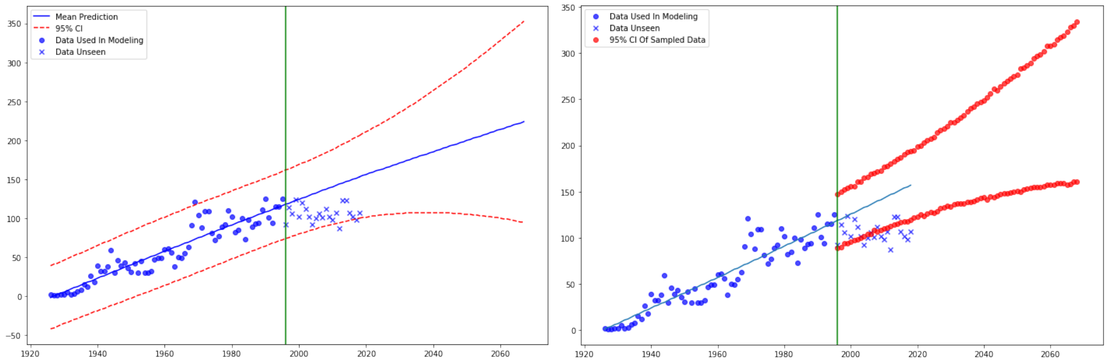
national safety standards in aviation. Interestingly, due to the effect of WWII, the linear trend assumption worked even poorer for the military accidents (figure 5), as shown by the wide confidence intervals of both the Bayesian and the Stan model. Another point worth mentioning is that the estimations of the coefficients by both models with different constructions and different methods of estimations (variational versus MCMC) are the same. This might 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

```
model_trend = """
```

```

data {
//Data and its length
    int<lower=1> length;
    real<lower=0> data_[length];
//Number of generated values
    int<lower=0> n_gen;
}

parameters {
    //Quadratic trend model, with 3 coefficients
    real coef_1;
    real coef_2;
    real coef_3;
    real<lower=0> noise;
}

model {
    #Each coefficient is drawn from a broad Cauchy
    coef_1 ~ cauchy(0,1);
    coef_2 ~ cauchy(0,1);
    coef_3 ~ cauchy(0,1);
    for(i in 1:length) {
        data_[i] ~ normal(coef_1*i*i + coef_2*i + coef_3, noise);
    }
}

generated quantities {
    real data_gen[n_gen];
    for(i in 1:n_gen) {
        data_gen[i] = normal_rng(coef_1*(i+length)*(i+length) + coef_2*(i+
    length) + coef_3, noise);
    }
}
"""
stan_model_trend = pystan.StanModel(model_code=model_trend)
```

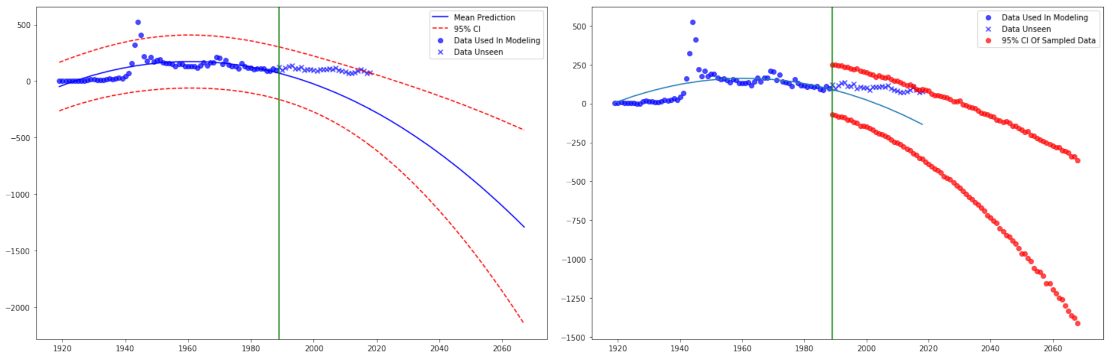
The only different between the quadratic model and the linear model is that we have three coefficients instead of two: the additional one accounts for the quadratic effect of t^2. The factor graphs do not differ much from the linear model, but the predictions do. In figure 6, because up to the point 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



(a) Bayesian Linear Model

(b) Stan Model

Figure 6: Predictions Of Quadratic Models - Commercial



(a) Bayesian Linear Model

(b) Stan Model

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

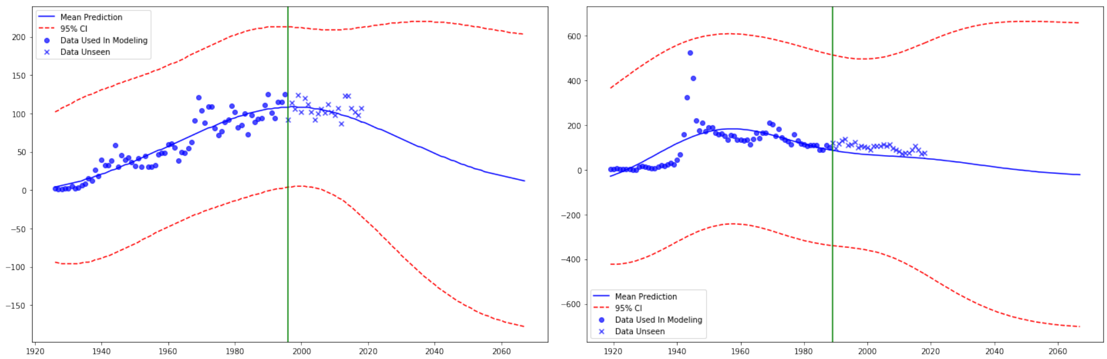
A radial basis function has the form

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

which measures the similarity (by Euclidean distance) between the current point x and a pivot point x_0. This measure of similarity and be used as a feature to indicate the relative position between the data, and since the effect of the pivot point diminish 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.

```
scaler = MinMaxScaler()
N = 70

#Using sklearn rbf kernel
from sklearn.metrics.pairwise import rbf_kernel

#100 pivot points
kernel = np.linspace(0,2,100).reshape(-1,1)
#Rescaling the year to the range 0 to 1, a standard procedure before fitting
year_count_com['index'] = scaler.fit_transform(np.array(year_count_com['index']).reshape(-1,1))

X_train = rbf_kernel(np.array(year_count_com['index'][:N]).reshape(-1,1), kernel, gamma=10)

#Using Bayesian model and plotting the results
reg = BayesianRidge()
reg.fit(X_train, year_count_com['year'].iloc[:N])
plt.figure(figsize=(12,8))
x = year_count_com['abs_year']
plt.scatter(x[:N], year_count_com['year'][:N], color='b', alpha=0.7, label='Data Used In Modeling')
plt.scatter(x[N:], year_count_com['year'][N:], color='b', marker='x', alpha=0.7, label='Data Unseen')

X_test = rbf_kernel(np.array(year_count_com['index']).reshape(-1,1),\
                    kernel, gamma=10)
y_m, y_std = reg.predict(X_test, return_std=True)
plt.plot(x, np.round(y_m), color='b', label='Mean Prediction')
plt.plot(x, np.round(y_m+2.96*y_std), color='r', linestyle='--', label='95% CI')
plt.plot(x, np.round(y_m-2.96*y_std), color='r', linestyle='--')

f_x = np.linspace(1,1.5,50).reshape(-1,1)
K_f_x = rbf_kernel(f_x, kernel, gamma=10)
```

```
f_y_m, f_y_std = reg.predict(K_f_x, return_std=True)
f_x = range(2018, 2018+50)
plt.plot(f_x, np.round(f_y_m), color='b')
plt.plot(f_x, np.round(f_y_m+2.96*f_y_std), color='r', linestyle='--')
plt.plot(f_x, np.round(f_y_m-2.96*f_y_std), color='r', linestyle='--')

plt.axvline(x[N], color='g')
plt.legend()
plt.show()

#Using Stan model
model_trend_main = """

data {
//Data and its length
    int<lower=1> length;
    real<lower=0> data_[length];
//Number of radial basis points or number of coefficients
    int<lower=1> c;
//Kernelized value of the training data
    vector[c] rbf_train[length];
//Number of generated values
    int<lower=0> n_gen;
//Kernelized value of the generated data
    vector[c] rbf_gen[n_gen];
}

parameters {
//RBF model, with c basis points
    row_vector[c] coef_;
//Intercept
    real intercept;
//Gaussian noise
    real<lower=0> noise;
}

model {
//Each coefficient is drawn from a broad Cauchy
    for(j in 1:c) {
        coef_[j] ~ cauchy(0,1);
    }
    for(i in 1:length) {
        data_[i] ~ normal((coef_*rbf_train[i]) + intercept, noise);
    }
}

generated quantities {
    real data_gen[n_gen];
    for(i in 1:n_gen) {
        data_gen[i] = normal_rng((coef_*rbf_gen[i]) + intercept, noise);
    }
}
"""
trend_model = pystan.StanModel(model_code=model_trend_main)
```



(a) Commercial (b) Military

Figure 8: Predictions Of RBF Bayesian Models

```

kernel = np.linspace(0,2,100).reshape(-1,1)
X_train = rbf_kernel(np.array(year_count_com['index'][:N]).reshape(-1,1),
    kernel, gamma=10)
X_gen = rbf_kernel(np.array(year_count_com['index'].iloc[N:]).reshape(-1,1)
    , kernel, gamma=10)
f_X = np.linspace(1,1.5,50).reshape(-1,1)
K_f_X = rbf_kernel(f_X, kernel, gamma=10)
X_gen = np.concatenate((X_gen, K_f_X), axis=0)

stan_data = {
    'length': year_count_com['year'][:N].shape[0],
    'data_': np.array(year_count_com['year'][:N]),
    'c': 100,
    'rbf_train': X_train,
    'n_gen': year_count_com['year'][N:].shape[0] + 50,
    'rbf_gen': X_gen
}
results = trend_model.sampling(data=stan_data)
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 no 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 woth 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).

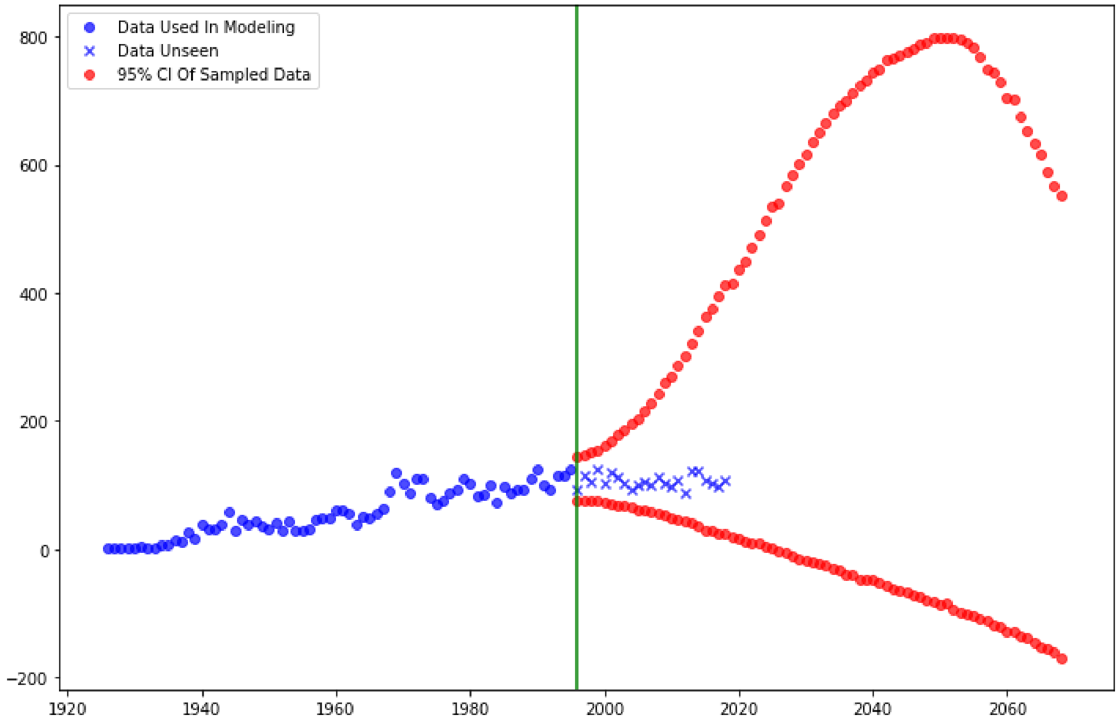


Figure 9: Stan RBF Model

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.

```
stan_model_risk = """

data {
//Number of airline
```

```
    int<lower=1> L;
//Number of aircraft
    int<lower=1> C;
//Number of accident type
    int<lower=1> T;
//Number of datapoint
    int<lower=1> N;

//A vector of 0/1 in each entry expressing which airline, aircraft and
    accident type the datapoint is
    vector<lower=0>[L] airline[N];
    vector<lower=0>[C] aircraft[N];
    vector<lower=0>[T] accident[N];

//A vector expressing the number of passenger on that aircraft
    int<lower=0> passenger[N];

//Number of fatality
    int<lower=0> fat[N];
}


parameters {

//Scaling factor for the airline
    row_vector<lower=0, upper=1>[L] m_al;
//Scaling factor for the aircraft
    row_vector<lower=0, upper=1>[C] m_ac;
//Scaling factor for the accident type
    row_vector<lower=0, upper=1>[T] m_a;

//Mortality rate
    real<lower=0, upper=1> p;
}

model {

    for (l in 1:L){
        m_al[l] ~ lognormal(0,0.25);
    }

    for (c in 1:C){
        m_ac[c] ~ lognormal(0,0.25);
    }

    for (t in 1:T){
        m_a[t] ~ lognormal(0,0.25);
    }

    for(i in 1:N) {
        #Sample the mortality rate from a beta distribution, with scaling
    factor from the airline, aircraft and accident
        p ~ beta(((m_al*airline[i])+(m_ac*aircraft[i])+(m_a*accident[i])),
    1);
```

```
            fat[i] ~ binomial(passenger[i], p);
        }
}

"""
model_risk = pystan.StanModel(model_code=stan_model_risk)
```

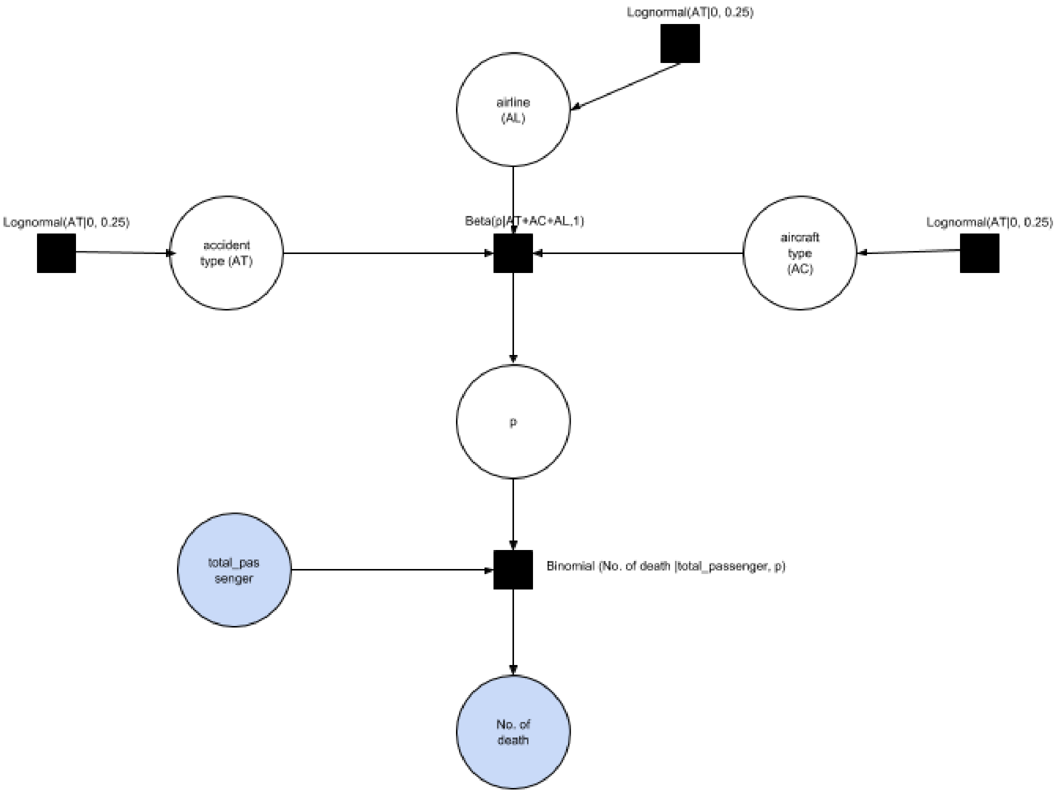
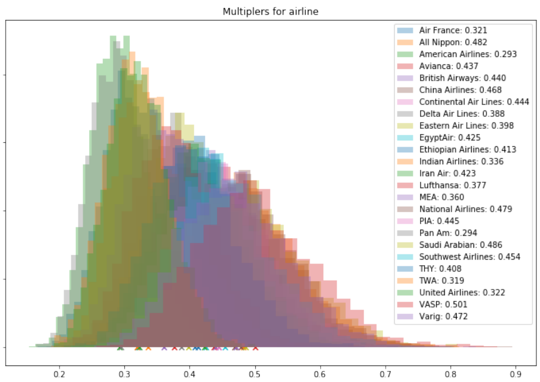


Figure 10: Mortality Risk Model Factor Graph

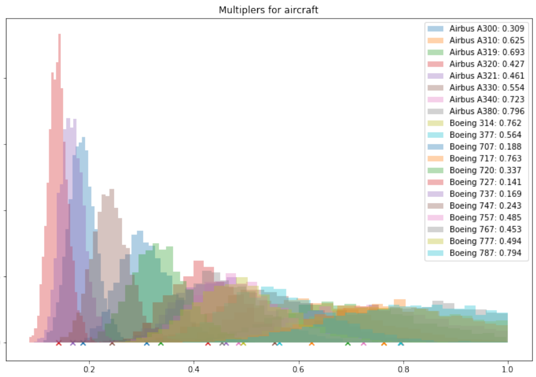
The model is specified in Stan code above and described in figure 10 factor graph. The unobserved variable of the model includes p, the mortality rate; airline scale factor, which is used as a component of the alpha parameter for the beta distribution; aircraft type scale factor and accident type scale 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-

.png

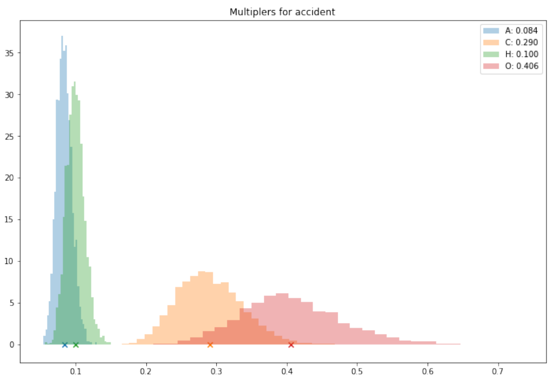


(a) Airline

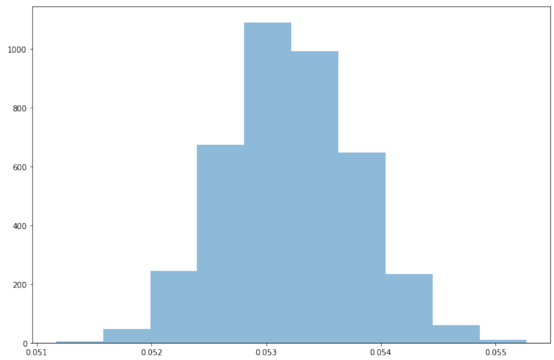


(b) Aircraft Type

Figure 11: Posteriors Of Scaling Factor - Mortatily Risk



(a) Accident Type

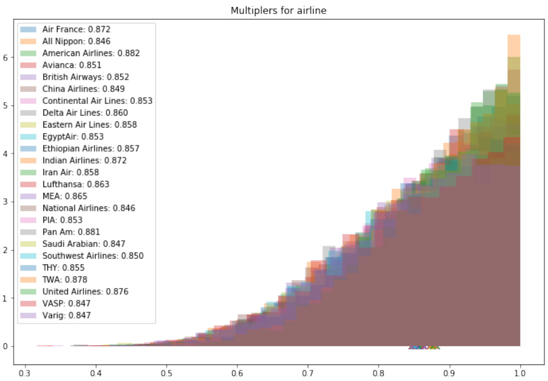


(b) Average Mortality Rate

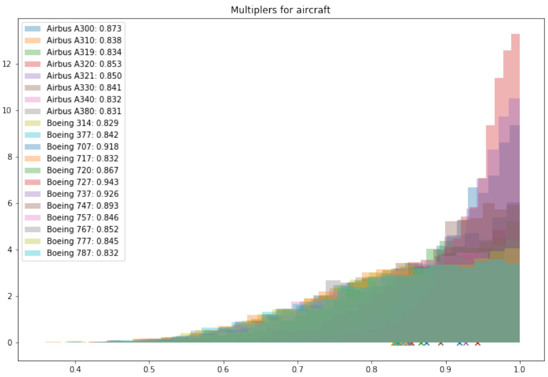
Figure 12: Posteriors Of Scaling Factor - Mortality Risk

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

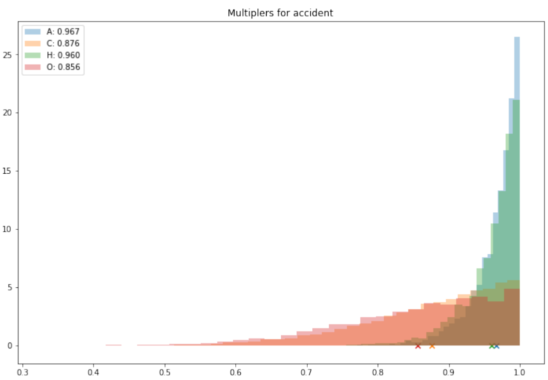


(a) Airline

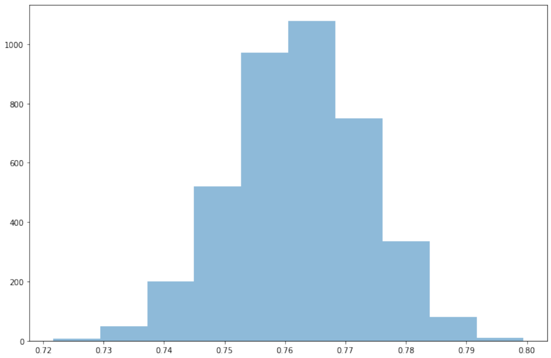


(b) Aircraft Type

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



(a) Accident Type



(b) Average Mortality Rate

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-0ME32gQ2FsOzWbc/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

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import pystan
from sklearn.linear_model import LinearRegression, BayesianRidge
from sklearn.preprocessing import MinMaxScaler

data = pd.read_csv('/Users/ash/Downloads/clean.csv')

def clean_day_year(obj):
    return int(obj)

def clean_month(obj):
    month = str(obj).lower()
    if month == 'jan':
        month = '01'
    elif month == 'feb':
        month = '02'
    elif month == 'mar':
        month = '03'
    elif month == 'apr':
        month = '04'
    elif month == 'may':
        month = '05'
    elif month == 'jun':
        month = '06'
    elif month == 'jul':
        month = '07'
    elif month == 'aug':
        month = '08'
    elif month == 'sep':
        month = '09'
    elif month == 'oct':
        month = '10'
```

```
    elif month == 'nov':
        month = '11'
    elif month == 'dec':
        month = '12'
    return month

def clean_fat(obj):
    fat = str(obj)
    if fat.find('+') == -1:
        return int(fat)
    elif fat.find('+') != "+":
        total = int(fat[0:fat.find('+')]) + int(fat[fat.find('+')+2:])
        return int(total)

data = data.dropna()
data['day'] = data['day'].apply(clean_day_year)
data['year'] = data['year'].apply(clean_day_year)
data['month'] = data['month'].apply(clean_month)
data['fat.'] = data['fat.'].apply(clean_fat)

data['date'] = pd.to_datetime(data[['day', 'month', 'year']])
data.head()

com = data.loc[data['Aircraft Cat'] == 'Commercial']
mil = data.loc[data['Aircraft Cat'] == 'Military']
print('Commercial Size:', com.shape)
print('Miliatary Size:', mil.shape)

#general trend by
year_count_com = (com['year'].value_counts()).reset_index()
year_count_com = year_count_com.sort_values(by=['index'], kind='mergesort')
year_count_com = year_count_com.reset_index(drop=True)

year_count_mil = mil['year'].value_counts().reset_index()
year_count_mil = year_count_mil.sort_values(by=['index'], kind='mergesort')
year_count_mil = year_count_mil.reset_index(drop=True)

plt.figure(figsize=(12,8))
plt.subplot(2,1,1)
plt.scatter(range(year_count_com['index'].iloc[0], year_count_com['index'].
    iloc[-1]+1),\
            year_count_com['year'], color='b', alpha=0.7)
plt.title('Commercial Trend {} to {}'.format(year_count_com['index'].iloc
    [0],year_count_com['index'].iloc[-1]))
plt.subplot(2,1,2)
plt.scatter(range(year_count_mil['index'].iloc[0], year_count_mil['index'].
    iloc[-1]+1),\
            year_count_mil['year'], color='r', alpha=0.7)
plt.title('Military Trend {} to {}'.format(year_count_mil['index'].iloc[0],
    year_count_mil['index'].iloc[-1]))
plt.show()

year_count_com['abs_year'] = year_count_com['index']
year_count_mil['abs_year'] = year_count_mil['index']
```

```
def visualize_posterior(samples, names):
    for _ in names:
        plt.figure(figsize=(12,4))
        plt.hist(samples[_])
        plt.title('Posterior samples of '+_)
    plt.show()

scaler = MinMaxScaler()
N = 70

year_count_com['index'] = scaler.fit_transform(np.array(year_count_com['index']).reshape(-1,1))
reg = BayesianRidge()
reg.fit(np.array(year_count_com[['index']].iloc[:N,:]), year_count_com['year'].iloc[:N])
plt.figure(figsize=(12,8))
x = year_count_com['abs_year']
plt.scatter(x[:N], year_count_com['year'][:N], color='b', alpha=0.7, label='Data Used In Modeling')
plt.scatter(x[N:], year_count_com['year'][N:], color='b', marker='x', alpha=0.7, label='Data Unseen')
y_m, y_std = reg.predict(np.array(year_count_com[['index']]), return_std=True)
plt.plot(x, np.round(y_m), color='b', label='Mean Prediction')
plt.plot(x, np.round(y_m+2.96*y_std), color='r', linestyle='--', label='95% CI')
plt.plot(x, np.round(y_m-2.96*y_std), color='r', linestyle='--')

f_x = np.linspace(1,1.5,50)
f_y_m, f_y_std = reg.predict(f_x.reshape(-1,1), return_std=True)
f_x = range(2018, 2018+50)
plt.plot(f_x, np.round(f_y_m), color='b')
plt.plot(f_x, np.round(f_y_m+2.96*f_y_std), color='r', linestyle='--')
plt.plot(f_x, np.round(f_y_m-2.96*f_y_std), color='r', linestyle='--')

plt.axvline(x[N], color='g')
plt.legend()
plt.show()

model_trend = """

data {
//Data and its length
    int<lower=1> length;
    real<lower=0> data_[length];
//Number of generated values
    int<lower=0> n_gen;
}

parameters {
    //Linear trend model, with 2 coefficients
    real coef_1;
    real coef_2;
    real<lower=0> noise;
```

```
}

model {
    #Each coefficient is drawn from a broad Cauchy
    coef_1 ~ cauchy(0,1);
    coef_2 ~ cauchy(0,1);
    noise ~ cauchy(0,1);
    for(i in 1:length) {
        data_[i] ~ normal(coef_1*i + coef_2, noise);
    }
}

generated quantities {
    real data_gen[n_gen];
    for(i in 1:n_gen) {
        data_gen[i] = normal_rng(coef_1*(i+length) + coef_2, noise);
    }
}
"""
stan_model_trend = pystan.StanModel(model_code=model_trend)

stan_data = {
    'length': year_count_com['year'][:N].shape[0],
    'data_': np.array(year_count_com['year'][:N]),
    'n_gen': year_count_com['year'][N:].shape[0] + 50
}
results = stan_model_trend.sampling(data=stan_data)
samples = results.extract()

paras =['coef_1', 'coef_2', 'noise']
print(results.stansummary(pars=paras))

#From CS146 - 14.1 Pre-class, Professor Carl Scheffler
def plot_acf(x):
    from scipy import signal
    plt.acorr(
        x, maxlags=20, detrend=lambda x: signal.detrend(x, type='constant')
    )


for param in paras:
    plt.figure(figsize=(12, 4))
    plot_acf(samples[param])
    plt.title(f'Autocorrelation of {param} samples')

plt.show()

visualize_posterior(samples, paras)

future = samples['data_gen']
f_interval = np.percentile(future, axis=0, q=[2.5, 97.5])
plt.figure(figsize=(12,8))
x = year_count_com['abs_year']
plt.scatter(x[:N], year_count_com['year'][:N], color='b', alpha=0.7, label=
```

```
'Data Used In Modeling')
plt.scatter(x[N:], year_count_com['year'][N:], color='b', marker='x', alpha=0.7, label='Data Unseen')
x_ = range(1, year_count_com['index'].shape[0]+1)
y = np.round([x_[i]*np.mean(samples['coef_1'])+np.mean(samples['coef_2']) for i in range(len(x))])
plt.plot(year_count_com['abs_year'], y)
plt.scatter(x[N]+range(stan_data['n_gen']), np.round(f_interval[0,:]), color='r', alpha=0.7,\
            label='95% CI Of Sampled Data')
plt.scatter(x[N]+range(stan_data['n_gen']), np.round(f_interval[1,:]), color='r', alpha=0.7)
plt.axvline(x[N], color='g')
plt.legend()
plt.show()

scaler = MinMaxScaler()
N = 70

year_count_mil['index'] = scaler.fit_transform(np.array(year_count_mil['index']).reshape(-1,1))
reg = BayesianRidge()
reg.fit(np.array(year_count_mil[['index']].iloc[:N,:]), year_count_mil['year'].iloc[:N])
plt.figure(figsize=(12,8))
x = year_count_mil['abs_year']
plt.scatter(x[:N], year_count_mil['year'][:N], color='b', alpha=0.7, label='Data Used In Modeling')
plt.scatter(x[N:], year_count_mil['year'][N:], color='b', marker='x', alpha=0.7, label='Data Unseen')
y_m, y_std = reg.predict(np.array(year_count_mil[['index']]), return_std=True)
plt.plot(x, np.round(y_m), color='b', label='Mean Prediction')
plt.plot(x, np.round(y_m+2.96*y_std), color='r', linestyle='--', label='95% CI')
plt.plot(x, np.round(y_m-2.96*y_std), color='r', linestyle='--')

f_x = np.linspace(1,1.5,50)
f_y_m, f_y_std = reg.predict(f_x.reshape(-1,1), return_std=True)
f_x = range(2018, 2018+50)
plt.plot(f_x, np.round(f_y_m), color='b')
plt.plot(f_x, np.round(f_y_m+2.96*f_y_std), color='r', linestyle='--')
plt.plot(f_x, np.round(f_y_m-2.96*f_y_std), color='r', linestyle='--')

plt.axvline(x[N], color='g')
plt.legend()
plt.show()

stan_data = {
    'length': year_count_mil['year'][:N].shape[0],
    'data_': np.array(year_count_mil['year'][:N]),
    'n_gen': year_count_mil['year'][N:].shape[0] + 50
}
results = stan_model_trend.sampling(data=stan_data)
```

```
samples = results.extract()

paras =['coef_1', 'coef_2', 'noise']
print(results.stansummary(pars=paras))

for param in paras:
    plt.figure(figsize=(12, 4))
    plot_acf(samples[param])
    plt.title(f'Autocorrelation of {param} samples')

plt.show()

visualize_posterior(samples, paras)

future = samples['data_gen']
f_interval = np.percentile(future, axis=0, q=[2.5, 97.5])
plt.figure(figsize=(12,8))
x = year_count_mil['abs_year']
plt.scatter(x[:N], year_count_mil['year'][:N], color='b', alpha=0.7, label='Data Used In Modeling')
plt.scatter(x[N:], year_count_mil['year'][N:], color='b', marker='x', alpha=0.7, label='Data Unseen')
x_ = range(1, year_count_mil['index'].shape[0]+1)
y = np.round([x_[i]*np.mean(samples['coef_1'])+np.mean(samples['coef_2']) for i in range(len(x))])
plt.plot(year_count_mil['abs_year'], y)
plt.scatter(x[N]+range(stan_data['n_gen']), np.round(f_interval[0,:]), color='r', alpha=0.7,\
            label='95% CI Of Sampled Data')
plt.scatter(x[N]+range(stan_data['n_gen']), np.round(f_interval[1,:]), color='r', alpha=0.7)
plt.axvline(x[N], color='g')
plt.legend()
plt.show()

scaler = MinMaxScaler()
N = 70

year_count_com['index'] = scaler.fit_transform(np.array(year_count_com['index']).reshape(-1,1))
year_count_com['index^2'] = year_count_com['index']**2
reg = BayesianRidge()
reg.fit(np.array(year_count_com[['index', 'index^2']].iloc[:N,:]), year_count_com['year'].iloc[:N])
plt.figure(figsize=(12,8))
x = year_count_com['abs_year']
plt.scatter(x[:N], year_count_com['year'][:N], color='b', alpha=0.7, label='Data Used In Modeling')
plt.scatter(x[N:], year_count_com['year'][N:], color='b', marker='x', alpha=0.7, label='Data Unseen')
y_m, y_std = reg.predict(np.array(year_count_com[['index', 'index^2']]), return_std=True)
plt.plot(x, np.round(y_m), label='Mean Prediction', color='b')
plt.plot(x, np.round(y_m+2.96*y_std), color='r', linestyle='--', label='95%
```

```
    CI')
plt.plot(x, np.round(y_m-2.96*y_std), color='r', linestyle='--')

f_x = np.expand_dims(np.linspace(1,1.5,50), axis=1)
f_x = np.concatenate((f_x, f_x**2), axis=1)
f_y_m, f_y_std = reg.predict(f_x, return_std=True)
f_x = range(2018, 2018+50)
plt.plot(f_x, np.round(f_y_m), color='b')
plt.plot(f_x, np.round(f_y_m+2.96*f_y_std), color='r', linestyle='--')
plt.plot(f_x, np.round(f_y_m -2.96*f_y_std), color='r', linestyle='--')

plt.axvline(x[N], color='g')
plt.legend()
plt.show()

model_trend = """

data {
//Data and its length
    int<lower=1> length;
    real<lower=0> data_[length];
//Number of generated values
    int<lower=0> n_gen;
}

parameters {
    //Quadratic trend model, with 3 coefficients
    real coef_1;
    real coef_2;
    real coef_3;
    real<lower=0> noise;
}

model {
    #Each coefficient is drawn from a broad Cauchy
    coef_1 ~ cauchy(0,1);
    coef_2 ~ cauchy(0,1);
    coef_3 ~ cauchy(0,1);
    for(i in 1:length) {
        data_[i] ~ normal(coef_1*i*i + coef_2*i + coef_3, noise);
    }
}

generated quantities {
    real data_gen[n_gen];
    for(i in 1:n_gen) {
        data_gen[i] = normal_rng(coef_1*(i+length)*(i+length) + coef_2*(i+
    length) + coef_3, noise);
    }
}
"""
stan_model_trend = pystan.StanModel(model_code=model_trend)

stan_data = {
```

```
    'length': year_count_com['year'][:N].shape[0],
    'data_': np.array(year_count_com['year'][:N]),
    'n_gen': year_count_com['year'][N:].shape[0]+50
}
results = stan_model_trend.sampling(data=stan_data)
samples = results.extract()

paras =['coef_1', 'coef_2', 'coef_3', 'noise']
print(results.stansummary(pars=paras))
for param in paras:
    plt.figure(figsize=(12, 4))
    plot_acf(samples[param])
    plt.title(f'Autocorrelation of {param} samples')

plt.show()

visualize_posterior(samples, paras)

future = samples['data_gen']
f_interval = np.percentile(future, axis=0, q=[2.5, 97.5])
plt.figure(figsize=(12,8))
x = year_count_com['abs_year']
plt.scatter(x[:N], year_count_com['year'][:N], color='b', alpha=0.7, label=
    'Data Used In Modeling')
plt.scatter(x[N:], year_count_com['year'][N:], color='b', marker='x', alpha
    =0.7, label='Data Unseen')
x_ = range(1, year_count_com['index'].shape[0]+1)
y = np.round([x_[i]**2*np.mean(samples['coef_1'])+x_[i]*np.mean(samples['
    coef_2']) + np.mean(samples['coef_3'])\
              for i in range(len(x))])
plt.plot(year_count_com['abs_year'], y)
plt.scatter(x[N]+range(stan_data['n_gen']), np.round(f_interval[0,:]),
    color='r', alpha=0.7,\
            label='95% CI Of Sampled Data')
plt.scatter(x[N]+range(stan_data['n_gen']), np.round(f_interval[1,:]),
    color='r', alpha=0.7)
plt.axvline(x[N], color='g')
plt.legend()
plt.show()

scaler = MinMaxScaler()
N = 70

year_count_mil['index'] = scaler.fit_transform(np.array(year_count_mil['
    index']).reshape(-1,1))
year_count_mil['index^2'] = year_count_mil['index']**2
reg = BayesianRidge()
reg.fit(np.array(year_count_mil[['index', 'index^2']].iloc[:N,:]),
    year_count_mil['year'].iloc[:N])
plt.figure(figsize=(12,8))
x = year_count_mil['abs_year']
plt.scatter(x[:N], year_count_mil['year'][:N], color='b', alpha=0.7, label=
    'Data Used In Modeling')
plt.scatter(x[N:], year_count_mil['year'][N:], color='b', marker='x', alpha
```

```python
=0.7, label='Data Unseen')
y_m, y_std = reg.predict(np.array(year_count_mil[['index', 'index^2']]), return_std=True)
plt.plot(x, np.round(y_m), label='Mean Prediction', color='b')
plt.plot(x, np.round(y_m+2.96*y_std), color='r', linestyle='--', label='95% CI')
plt.plot(x, np.round(y_m-2.96*y_std), color='r', linestyle='--')

f_x = np.expand_dims(np.linspace(1,1.5,50), axis=1)
f_x = np.concatenate((f_x, f_x**2), axis=1)
f_y_m, f_y_std = reg.predict(f_x, return_std=True)
f_x = range(2018, 2018+50)
plt.plot(f_x, np.round(f_y_m), color='b')
plt.plot(f_x, np.round(f_y_m+2.96*f_y_std), color='r', linestyle='--')
plt.plot(f_x, np.round(f_y_m -2.96*f_y_std), color='r', linestyle='--')

plt.axvline(x[N], color='g')
plt.legend()
plt.show()

stan_data = {
    'length': year_count_mil['year'][:N].shape[0],
    'data_': np.array(year_count_mil['year'][:N]),
    'n_gen': year_count_mil['year'][N:].shape[0]+50
}
results = stan_model_trend.sampling(data=stan_data)
samples = results.extract()

paras =['coef_1', 'coef_2', 'coef_3', 'noise']
print(results.stansummary(pars=paras))

#From CS146 - 14.1 Pre-class, Professor Carl Scheffler
def plot_acf(x):
    from scipy import signal
    plt.acorr(
        x, maxlags=20, detrend=lambda x: signal.detrend(x, type='constant')
    )


for param in paras:
    plt.figure(figsize=(12, 4))
    plot_acf(samples[param])
    plt.title(f'Autocorrelation of {param} samples')

plt.show()

visualize_posterior(samples, paras)

future = samples['data_gen']
f_interval = np.percentile(future, axis=0, q=[2.5, 97.5])
plt.figure(figsize=(12,8))
x = year_count_mil['abs_year']
plt.scatter(x[:N], year_count_mil['year'][:N], color='b', alpha=0.7, label='Data Used In Modeling')
```

```
plt.scatter(x[N:], year_count_mil['year'][N:], color='b', marker='x', alpha
    =0.7, label='Data Unseen')
x_ = range(1, year_count_mil['index'].shape[0]+1)
y = np.round([x_[i]**2*np.mean(samples['coef_1'])+x_[i]*np.mean(samples['
    coef_2'])+np.mean(samples['coef_3'])\
             for i in range(len(x))])
plt.plot(year_count_mil['abs_year'], y)
plt.scatter(x[N]+range(stan_data['n_gen']), np.round(f_interval[0,:]),
    color='r', alpha=0.7,\
            label='95% CI Of Sampled Data')
plt.scatter(x[N]+range(stan_data['n_gen']), np.round(f_interval[1,:]),
    color='r', alpha=0.7)
plt.axvline(x[N], color='g')
plt.legend()
plt.show()

scaler = MinMaxScaler()
N = 70

#Using sklearn rbf kernel
from sklearn.metrics.pairwise import rbf_kernel

#100 pivot points
kernel = np.linspace(0,2,100).reshape(-1,1)
#Rescaling the year to the range 0 to 1, a standard procedure before
    fitting
year_count_com['index'] = scaler.fit_transform(np.array(year_count_com['
    index']).reshape(-1,1))

X_train = rbf_kernel(np.array(year_count_com['index'][:N]).reshape(-1,1),
    kernel, gamma=10)

#Using Bayesian model and plotting the results
reg = BayesianRidge()
reg.fit(X_train, year_count_com['year'].iloc[:N])
plt.figure(figsize=(12,8))
x = year_count_com['abs_year']
plt.scatter(x[:N], year_count_com['year'][:N], color='b', alpha=0.7, label=
    'Data Used In Modeling')
plt.scatter(x[N:], year_count_com['year'][N:], color='b', marker='x', alpha
    =0.7, label='Data Unseen')

X_test = rbf_kernel(np.array(year_count_com['index']).reshape(-1,1),\
                    kernel, gamma=10)
y_m, y_std = reg.predict(X_test, return_std=True)
plt.plot(x, np.round(y_m), color='b', label='Mean Prediction')
plt.plot(x, np.round(y_m+2.96*y_std), color='r', linestyle='--', label='95%
    CI')
plt.plot(x, np.round(y_m-2.96*y_std), color='r', linestyle='--')

f_x = np.linspace(1,1.5,50).reshape(-1,1)
K_f_x = rbf_kernel(f_x, kernel, gamma=10)
f_y_m, f_y_std = reg.predict(K_f_x, return_std=True)
f_x = range(2018, 2018+50)
```

```
plt.plot(f_x, np.round(f_y_m), color='b')
plt.plot(f_x, np.round(f_y_m+2.96*f_y_std), color='r', linestyle='--')
plt.plot(f_x, np.round(f_y_m-2.96*f_y_std), color='r', linestyle='--')

plt.axvline(x[N], color='g')
plt.legend()
plt.show()

scaler = MinMaxScaler()
N = 70

from sklearn.metrics.pairwise import rbf_kernel

kernel = np.linspace(0,2,100).reshape(-1,1)
year_count_mil['index'] = scaler.fit_transform(np.array(year_count_mil['index']).reshape(-1,1))

X_train = rbf_kernel(np.array(year_count_mil['index'][:N]).reshape(-1,1), kernel, gamma=10)

reg = BayesianRidge()
reg.fit(X_train, year_count_mil['year'].iloc[:N])
plt.figure(figsize=(12,8))
x = year_count_mil['abs_year']
plt.scatter(x[:N], year_count_mil['year'][:N], color='b', alpha=0.7, label='Data Used In Modeling')
plt.scatter(x[N:], year_count_mil['year'][N:], color='b', marker='x', alpha=0.7, label='Data Unseen')

X_test = rbf_kernel(np.array(year_count_mil['index']).reshape(-1,1),\
                    kernel, gamma=10)
y_m, y_std = reg.predict(X_test, return_std=True)
plt.plot(x, np.round(y_m), color='b', label='Mean Prediction')
plt.plot(x, np.round(y_m+2.96*y_std), color='r', linestyle='--', label='95% CI')
plt.plot(x, np.round(y_m-2.96*y_std), color='r', linestyle='--')

f_x = np.linspace(1,1.5,50).reshape(-1,1)
K_f_x = rbf_kernel(f_x, kernel, gamma=10)
f_y_m, f_y_std = reg.predict(K_f_x, return_std=True)
f_x = range(2018, 2018+50)
plt.plot(f_x, np.round(f_y_m), color='b')
plt.plot(f_x, np.round(f_y_m+2.96*f_y_std), color='r', linestyle='--')
plt.plot(f_x, np.round(f_y_m-2.96*f_y_std), color='r', linestyle='--')

plt.axvline(x[N], color='g')
plt.legend()
plt.show()

model_trend_main = """

data {
//Data and its length
    int<lower=1> length;
```

```
    real<lower=0> data_[length];
//Number of radial basis points or number of coefficients
    int<lower=1> c;
//Kernelized value of the training data
    vector[c] rbf_train[length];
//Number of generated values
    int<lower=0> n_gen;
//Kernelized value of the generated data
    vector[c] rbf_gen[n_gen];
}

parameters {
//RBF model, with c basis points
    row_vector[c] coef_;
//Intercept
    real intercept;
//Gaussian noise
    real<lower=0> noise;
}

model {
//Each coefficient is drawn from a broad Cauchy
    for(j in 1:c) {
        coef_[j] ~ cauchy(0,1);
    }
    for(i in 1:length) {
        data_[i] ~ normal((coef_*rbf_train[i]) + intercept, noise);
    }
}

generated quantities {
    real data_gen[n_gen];
    for(i in 1:n_gen) {
        data_gen[i] = normal_rng((coef_*rbf_gen[i]) + intercept, noise);
    }
}
"""
trend_model = pystan.StanModel(model_code=model_trend_main)

kernel = np.linspace(0,2,100).reshape(-1,1)
X_train = rbf_kernel(np.array(year_count_com['index'][:N]).reshape(-1,1),
    kernel, gamma=10)
X_gen = rbf_kernel(np.array(year_count_com['index'].iloc[N:]).reshape(-1,1)
    , kernel, gamma=10)
f_X = np.linspace(1,1.5,50).reshape(-1,1)
K_f_X = rbf_kernel(f_X, kernel, gamma=10)
X_gen = np.concatenate((X_gen, K_f_X), axis=0)

stan_data = {
    'length': year_count_com['year'][:N].shape[0],
    'data_': np.array(year_count_com['year'][:N]),
    'c': 100,
    'rbf_train': X_train,
    'n_gen': year_count_com['year'][N:].shape[0] + 50,
```

```
    'rbf_gen': X_gen
}
results = trend_model.sampling(data=stan_data)
samples = results.extract()

paras = ['coef_', 'intercept', 'noise']
print(results.stansummary(pars=paras))

future = samples['data_gen']
f_interval = np.percentile(future, axis=0, q=[2.5, 97.5])
plt.figure(figsize=(12,8))
x = year_count_com['abs_year']
plt.scatter(x[:N], year_count_com['year'][:N], color='b', alpha=0.7, label=
    'Data Used In Modeling')
plt.scatter(x[N:], year_count_com['year'][N:], color='b', marker='x', alpha
    =0.7, label='Data Unseen')
x_ = range(1, year_count_com['index'].shape[0]+1)
# y = np.round([x_[i]**2*np.mean(samples['coef_1'])+x_[i]*np.mean(samples['
    coef_2']) + np.mean(samples['coef_3'])\
#                 for i in range(len(x))])
# plt.plot(year_count_com['abs_year'], y)
plt.scatter(x[N]+range(stan_data['n_gen']), np.round(f_interval[0,:]),
    color='r', alpha=0.7,\
            label='95% CI Of Sampled Data')
plt.scatter(x[N]+range(stan_data['n_gen']), np.round(f_interval[1,:]),
    color='r', alpha=0.7)
plt.axvline(x[N], color='g')
plt.legend()
plt.show()

risk_com = com[['year', 'type', 'operator', 'fat.', 'cat']].reset_index(
    drop=True)

def clean_cat_1(obj):
    obj = str(obj)
    return obj[0]

def clean_cat_2(obj):
    obj = str(obj)
    return obj[1]

risk_com['Accident Type'] = risk_com['cat']
risk_com['Loss Type'] = risk_com['cat']
risk_com = risk_com.drop(['cat'], axis=1)
risk_com['Accident Type'] = risk_com['Accident Type'].apply(clean_cat_1)
risk_com['Loss Type'] = risk_com['Loss Type'].apply(clean_cat_2)
risk_com = risk_com.sort_values(by=['year'], kind='mergesort').reset_index(
    drop=True)

def Boeing_Airbus(data):
    df = pd.DataFrame()
    for _ in range(data.shape[0]):
        if ('Boeing' in data['type'].iloc[_]):
            data.iloc[_, 1] = data.iloc[_, 1][:10]
```

```
            if data.iloc[_, 1] == 'Boeing':
                pass
            else:
                df = pd.concat((df, data.iloc[_,:]), axis=1, sort=False)
        elif ('Airbus' in data['type'].iloc[_]):
            data.iloc[_, 1] = data.iloc[_, 1][:11]
            if data.iloc[_, 1] == 'Airbus':
                pass
            else:
                df = pd.concat((df, data.iloc[_,:]), axis=1, sort=False)
    return df.transpose().reset_index(drop=True)

def Top_Airline(data, top_airlines):
    df = pd.DataFrame()
    for _ in range(data.shape[0]):
        if data['operator'].iloc[_] in top_airlines:
            df = pd.concat((df, data.iloc[_,:]), axis=1, sort=False)
    return df.transpose().reset_index(drop=True)

def how_many_passenger(data):
    data['total'] = np.nan
    for _ in range(data.shape[0]):
        if data['type'].iloc[_] == 'Boeing 727':
            data['total'].iloc[_] = 189
        elif data['type'].iloc[_] == 'Boeing 737':
            data['total'].iloc[_] = 143
        elif data['type'].iloc[_] == 'Boeing 707':
            data['total'].iloc[_] = 189
        elif data['type'].iloc[_] == 'Boeing 747':
            data['total'].iloc[_] = 660
        elif data['type'].iloc[_] == 'Boeing 720':
            data['total'].iloc[_] = 219
        elif data['type'].iloc[_] == 'Boeing 757':
            data['total'].iloc[_] = 295
        elif data['type'].iloc[_] == 'Boeing 767':
            data['total'].iloc[_] = 351
        elif data['type'].iloc[_] == 'Boeing 777':
            data['total'].iloc[_] = 451
        elif data['type'].iloc[_] == 'Boeing 377':
            data['total'].iloc[_] = 114
        elif data['type'].iloc[_] == 'Boeing 717':
            data['total'].iloc[_] = 134
        elif data['type'].iloc[_] == 'Boeing 787':
            data['total'].iloc[_] = 335
        elif data['type'].iloc[_] == 'Boeing 314':
            data['total'].iloc[_] = 77
        elif data['type'].iloc[_] == 'Airbus A300':
            data['total'].iloc[_] = 300
        elif data['type'].iloc[_] == 'Airbus A320':
            data['total'].iloc[_] = 186
        elif data['type'].iloc[_] == 'Airbus A321':
            data['total'].iloc[_] = 240
        elif data['type'].iloc[_] == 'Airbus A330':
            data['total'].iloc[_] = 335
```

```
        elif data['type'].iloc[_] == 'Airbus A310':
            data['total'].iloc[_] = 220
        elif data['type'].iloc[_] == 'Airbus A319':
            data['total'].iloc[_] = 160
        elif data['type'].iloc[_] == 'Airbus A340':
            data['total'].iloc[_] = 475
        elif data['type'].iloc[_] == 'Airbus A380':
            data['total'].iloc[_] = 853
    return data

risk_com_BA = Boeing_Airbus(risk_com)
top_25_airlines = risk_com_BA['operator'].value_counts()[:25].index
risk_data = Top_Airline(risk_com_BA, top_25_airlines)
risk_data = how_many_passenger(risk_data)

risk_data = risk_data.drop([505, 504], axis=0).reset_index(drop=True)

stan_model_risk = """

data {
//Number of airline
    int<lower=1> L;
//Number of aircraft
    int<lower=1> C;
//Number of accident type
    int<lower=1> T;
//Number of datapoint
    int<lower=1> N;

//A vector of 0/1 in each entry expressing which airline, aircraft and
    accident type the datapoint is
    vector<lower=0>[L] airline[N];
    vector<lower=0>[C] aircraft[N];
    vector<lower=0>[T] accident[N];

//A vector expressing the number of passenger on that aircraft
    int<lower=0> passenger[N];

//Number of fatality
    int<lower=0> fat[N];
}


parameters {

//Scaling factor for the airline
    row_vector<lower=0, upper=1>[L] m_al;
//Scaling factor for the aircraft
    row_vector<lower=0, upper=1>[C] m_ac;
//Scaling factor for the accident type
    row_vector<lower=0, upper=1>[T] m_a;

//Mortality rate
    real<lower=0, upper=1> p;
```

```
}

model {

    for (l in 1:L){
        m_al[l] ~ lognormal(0,0.25);
    }

    for (c in 1:C){
        m_ac[c] ~ lognormal(0,0.25);
    }

    for (t in 1:T){
        m_a[t] ~ lognormal(0,0.25);
    }

    for(i in 1:N) {
        #Sample the mortality rate from a beta distribution, with scaling factor from the airline, aircraft and accident
        p ~ beta(((m_al*airline[i])+(m_ac*aircraft[i])+(m_a*accident[i])),1);
        fat[i] ~ binomial(passenger[i], p);
    }
}

"""
model_risk = pystan.StanModel(model_code=stan_model_risk)

def produce_stan_data(raw):
    airline = pd.get_dummies(raw['operator'])
    name_airline = airline.columns
    airline = np.array(airline, dtype=int)

    aircraft = pd.get_dummies(raw['type'])
    name_aircraft = aircraft.columns
    aircraft = np.array(aircraft, dtype=int)

    accident = pd.get_dummies(raw['Accident Type'])
    name_accident = accident.columns
    accident = np.array(accident, dtype=int)

    passenger = np.array(raw['total'], dtype=int)

    fat = np.array(raw['fat.'], dtype=int)

    hull = pd.get_dummies(raw['Loss Type'])
    hull = np.array(hull.iloc[:,0], dtype=int)

    data_dict = {
        'L': airline.shape[1],
        'C': aircraft.shape[1],
        'T': accident.shape[1],
        'N': airline.shape[0],

```

```
        'airline': airline,
        'aircraft': aircraft,
        'accident': accident,

        'passenger': passenger,

        'fat': fat,
        'hull': hull

    }

    name_dict = {
        'airline': name_airline,
        'aircraft': name_aircraft,
        'accident': name_accident
    }
    return data_dict, name_dict

def plot_posterior(results, name, what):
    if what == 'airline':
        what_ = 'm_al'
    elif what == 'aircraft':
        what_ = 'm_ac'
    else:
        what_ = 'm_a'
    mul = results[what_]
    plt.figure(figsize=(12,8))
    for _ in range(mul.shape[1]):
        plt.hist(mul[:,_], alpha=0.35, bins=30, density=True, label=str(
    name[what][_])+': {0:.3f}'.format(mul[:,_].mean()))
        plt.scatter(mul[:,_].mean(),0, marker='x', s=40)
    plt.legend()
    plt.title('Multiplers for {}'.format(what))
    mul = mul.reshape(-1,1)
    plt.figure(figsize=(12,8))
    plt.hist(mul, alpha=0.35, bins=30, density=True, label='var: {0:.3f}'.
    format(mul.var()))
    plt.scatter(mul.mean(), 0, marker='x', s=40, label='mean: {0:.3f}'.
    format(mul.mean()))
    plt.title('All multipliers')
    plt.legend()
    plt.show()
    print('95% CI for all {} multipliers is:'.format(what), (np.percentile(
    mul,2.5),np.percentile(mul,97.5)))

stan_data, name = produce_stan_data(risk_data)
results = model_risk.sampling(data=stan_data, n_jobs=1)
results

plot_posterior(results, name, 'airline')
plot_posterior(results, name, 'aircraft')
plot_posterior(results, name, 'accident')

plt.figure(figsize=(12,8))
```

```
plt.hist(results['p'], alpha=0.5)
plt.show()

stan_model_risk_hull = """

data {
//Number of airline
    int<lower=1> L;
//Number of aircraft
    int<lower=1> C;
//Number of accident type
    int<lower=1> T;
//Number of datapoint
    int<lower=1> N;

//A vector of 0/1 in each entry expressing which airline, aircraft and
    accident type the datapoint is
    vector<lower=0>[L] airline[N];
    vector<lower=0>[C] aircraft[N];
    vector<lower=0>[T] accident[N];

// Binary indicator of whether it's a hull loss or not
    int<lower=0> hull[N];
}


parameters {

//Scaling factor for the airline
    row_vector<lower=0, upper=1>[L] m_al;
//Scaling factor for the aircraft
    row_vector<lower=0, upper=1>[C] m_ac;
//Scaling factor for the accident type
    row_vector<lower=0, upper=1>[T] m_a;

//Mortality rate
    real<lower=0, upper=1> p;
}

model {

    for (l in 1:L){
        m_al[l] ~ lognormal(0,0.25);
    }

    for (c in 1:C){
        m_ac[c] ~ lognormal(0,0.25);
    }

    for (t in 1:T){
        m_a[t] ~ lognormal(0,0.25);
    }

    for(i in 1:N) {
```

```
        p ~ beta(((m_al*airline[i])+(m_ac*aircraft[i])+(m_a*accident[i])), 1);
        hull[i] ~ bernoulli(p);
    }
}

"""
model_risk_hull = pystan.StanModel(model_code=stan_model_risk_hull)

stan_data, name = produce_stan_data(risk_data)
results = model_risk_hull.sampling(data=stan_data, n_jobs=1)
results
plot_posterior(results, name, 'airline')
plot_posterior(results, name, 'aircraft')
plot_posterior(results, name, 'accident')
plt.figure(figsize=(12,8))
plt.hist(results['p'], alpha=0.5)
plt.show()
```