CS146 Project 2: Discrete and multi-level models

Below is a data set of GP (general practitioner) doctor visits in England*. The data set is stratified by age and geographical region. For each age and region group, 30 people were sampled randomly and asked how often they attended an appointment with a GP during the past year.

You will note that there are some missing values in the data set. Your task is to fit an appropriate discrete count model for this data set and to estimate the counts in the missing cells.

*Note: This data set is made up even though it is based on real statistics from the National Health Service Digital data service.

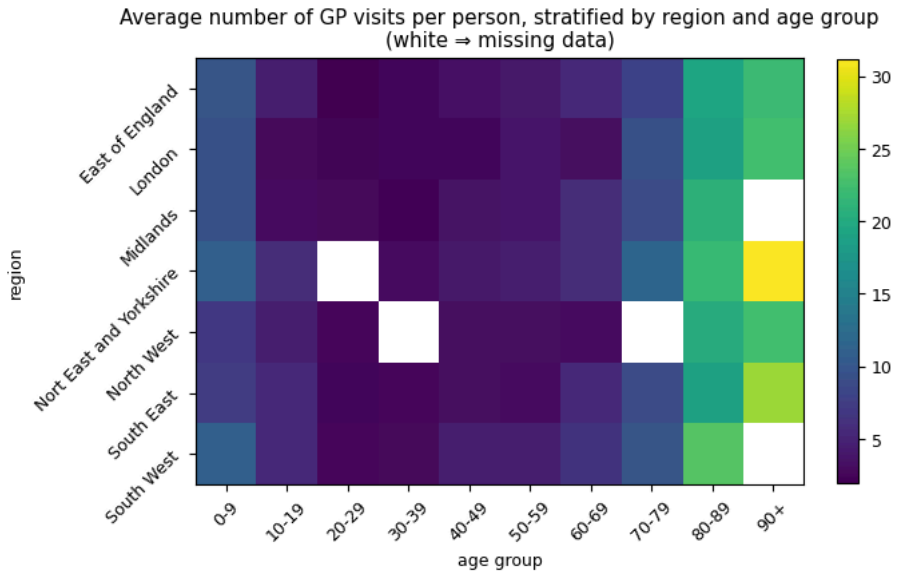
Instructions

Complete all the required tasks below. There are also some optional tasks for extra credit. Create a Python notebook with all your code, results, and the interpretation of your results. This Python notebook is your main deliverable for the assignment. Write text in addition to your code! It is important to explain what you are doing and what the results tell you. Keep it professional! Imagine you are writing your code and text for a client.

Data

The data set is in a 7 × 10 × 30 array for the 7 geographical regions, 10 age groups, and 30 samples per group. The cell below loads the data and plots the average number of GP visits for each group. You need to use all the data (not just the averages) but the averages should help to give you an idea of what the data set looks like.

```
import numpy as np
from numpy import nan  # not-a-number, used to indicated missing data
import matplotlib.pyplot as plt

regions = ['East of England', 'London', 'Midlands', 'Nort East and Yorkshire', 'North West', 'South East', 'South West']
ages = ['0–9', '10–19', '20–29', '30–39', '40–49', '50–59', '60–69', '70–79', '80–89', '90+']
sample_size = 30

raw_data = np.array([
    [[4,11,14,13,12,8,0,8,14,21,17,7,3,9,2,7,23,2,5,12,15,0,14,2,17,7,17,13,11,3],[4,4,10,2,24,1,10,0,0,5,1,3,3,3,0,9,5,0,4,14,3,
    [[6,18,6,5,14,0,7,10,4,12,15,9,2,30,12,0,9,0,14,23,6,5,0,2,11,8,25,0,30,0],[2,0,5,0,13,1,3,0,8,1,0,0,2,6,2,1,0,9,0,0,2,0,6,0,
    [[12,7,0,0,17,9,19,13,9,8,8,8,9,0,21,9,26,0,0,15,0,6,19,0,0,12,9,21,16,12],[2,2,4,0,4,6,7,0,7,0,5,0,0,9,6,2,0,0,4,4,2,6,3,0,0
    [[0,21,14,4,0,17,13,0,14,0,15,32,15,6,0,14,0,17,24,10,16,9,19,9,10,9,17,13,0,19],[12,5,7,2,4,1,0,7,8,11,1,3,6,3,11,0,4,5,4,7,
    [[0,8,0,12,14,3,4,9,0,0,5,0,30,5,0,1,21,8,0,5,0,11,17,3,13,3,18,13,0,0],[6,5,1,0,0,0,12,7,4,7,0,8,0,7,4,4,0,2,12,7,3,8,4,4,5,
    [[6,16,0,10,9,4,24,6,0,10,7,5,5,11,11,12,0,0,0,8,14,5,0,8,7,0,26,0,0,20],[7,0,8,2,3,13,11,5,7,7,0,0,4,0,3,15,3,8,8,5,12,1,4,1
    [[14,0,34,16,0,4,8,22,11,0,13,17,18,14,35,9,10,9,13,10,12,7,6,20,11,12,0,0,7,0],[5,12,0,10,8,0,7,0,6,0,20,0,10,0,3,6,9,8,7,0,

plt.rcParams.update({'font.size': 9})

plt.figure()
plt.title('Average number of GP visits per person, stratified by region and age group\n(white ⇒ missing data)')
plt.xlabel('age group')
plt.ylabel('region')
plt.imshow(raw_data.mean(axis=2))
plt.colorbar(fraction=0.032)
plt.xticks(range(len(ages)), ages, rotation=45)
plt.yticks(range(len(regions)), regions, rotation=45)
plt.show()
```



To make the data easier to work with in PyMC, you should reorganize it into 3 arrays — the region index (an integer indicating which geographical region each count is from), the age index (an integer for each age category), and the counts of the number of visits. You should leave out the missing (NaN) values since PyMC can't use those as observed values.

Model

We will use a Zero-Inflated Poisson likelihood function for this data set. The motivation for making it zero-inflated is that there are many more 0s in the data set than we would expect from a Poisson distribution since a lot of people never visit the doctor. This might be because they don't like doctors, don't take their health seriously, procrastinate, etc. We suspect that those people who do sometimes visit a doctor, follow a Poisson distribution for the number of appointments in a year.

Required tasks

- Come up with a strategy for estimating the values of the missing groups . How can you use the group values you have to estimate/predict the values of the groups you don’t have? Describe your strategy.
- Pre-process the data to make it easier to work with in PyMC by creating a region index array, age index array, and count array.
- Implement two Zero-Inflated Poisson models to produce the predictions for the missing groups.
    - Complete pooling. Assume the counts are generated i.i.d. from a Zero-Inflated Poisson distribution with unknown rate and zero-probability parameters. Compute the posterior distribution over these parameters and make predictions using this posterior distribution.
    - Partial pooling. Assume a hierarchical/multi-level model where each group has a Zero-Inflated Poisson distribution with its own parameters but the parameters come from a common prior with unknown parameters. Choose the prior appropriately. Compute the posterior distribution over all the rate parameters and the parameters of your chosen prior and make predictions using this posterior distribution.

  To simplify this somewhat, assume that the zero-probability parameter is the same for all groups but that the rate parameter depends on the group.
- Explain or demonstrate how you came up with your prior distributions for the two models.
- Show the predictions of the values of each missing group. Explain the differences between the predictions of the complete pooling and partial pooling models. You will find that one model is more confident (has less variance in its predictions) than the other model. Explain which model we should prefer.

Strategy:

In order to predict the data we will have to use a type of mixture model also known as Hierarichal model. This can help us represent similarities and differences between the clusters. We can use the group values we have to create a model and then use posterior predictions from that model to predict the missing values.

For now we are only interested in the raw data since that is what we need for our model to be able to predict. We can start by thinking of each intersection of row and column as a 'tank' almost. There are lots of unmeasured things peculiar to each tank, and these unmeasured factors create variation in visits across tanks, even when all the predictor variables have the same value. These tanks are an example of a cluster variable.

Importing all the neccessary libraries

```
import arviz as az
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pymc as pm
from scipy.special import expit as logistic
import seaborn as sns
```

Pre-processing the data to make it easier to work with in PyMC by creating a region index array, age index array, and observed array.

```
# Extract non-NaN indices
non_nan_indices = np.where(~np.isnan(raw_data))

# Create arrays for region index, age index, and counts
region, age = non_nan_indices[0], non_nan_indices[1]
observed = raw_data[non_nan_indices]
flat_data = raw_data.reshape(-1, raw_data.shape[-1])
Nages = len(np.unique(age))
Nregions = len(np.unique(region))
Nobserved = len(np.unique(observed))
# Create a binary mask for missing data
missing_data_mask = np.isnan(raw_data)
```

Implementing two Zero-Inflated Poisson models to produce the predictions for the missing groups.

Model 1: Complete pooling

I model the data using a gamma distribution for mu and a beta distribution for psi. A Gamma(1, 1) distribution is an uninformative or non-informative prior, often chosen when there is limited prior knowledge about the rate parameter. Similarly, a Beta(1, 1) distribution is uninformative and non-committal. It assigns equal probability density to all values between 0 and 1, reflecting a lack of strong prior beliefs about the zero-inflation parameter.

A zero inflated poisson was used for the likelihood as instructed. Thus we have a Zero-Inflated Poisson distribution with unknown rate and zero-probability parameters.

```
# Model for complete pooling
with pm.Model() as complete_pooling_model:
    data_n = pm.MutableData('data_n', age)
    # Prior for the rate parameter
    rate = pm.Gamma('rate', alpha=1, beta=1)

    # Prior for the zero-inflation parameter
    zero_inflation = pm.Beta('zero_inflation', alpha=1, beta=1)

    # Likelihood
    counts = pm.ZeroInflatedPoisson('counts', mu=rate, psi=zero_inflation,  observed=observed)

    # Sample from the posterior
    trace = pm.sample(1000)

trace
```

100.00% [2000/2000 00:06<00:00 Sampling chain 0, 0 divergences]
100.00% [2000/2000 00:05<00:00 Sampling chain 1, 0 divergences]

arviz.InferenceData

▶ posterior
▶ sample_stats
▶ observed_data
▶ constant_data

Posteriors

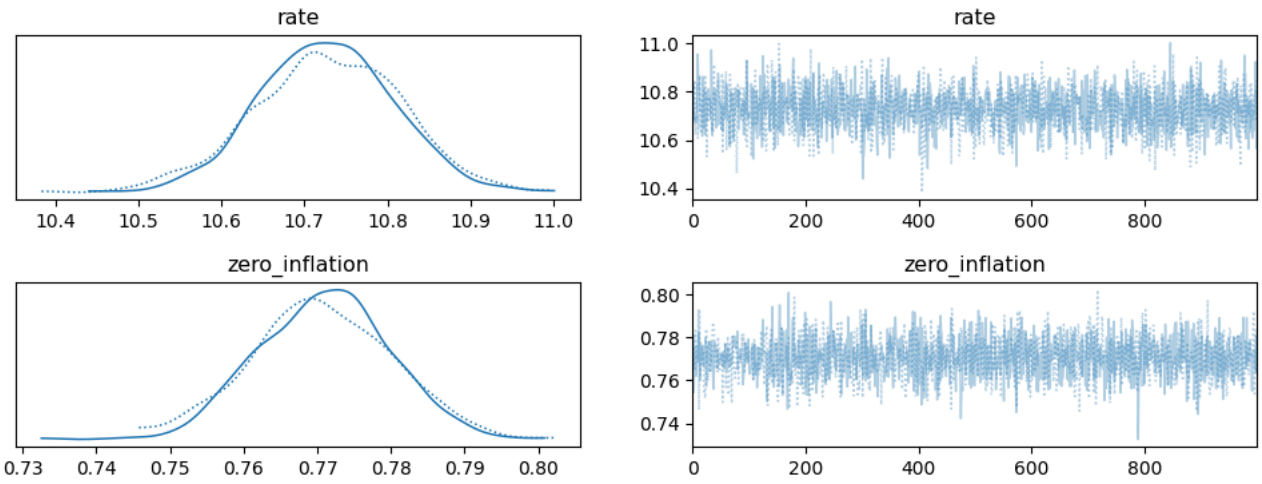
```
inference = {}  # store the results here
with complete_pooling_model:

    # Run MCMC sampling
    inference = pm.sample()
```

100.00% [2000/2000 00:06<00:00 Sampling chain 0, 0 divergences]
100.00% [2000/2000 00:05<00:00 Sampling chain 1, 0 divergences]

```
# Display summary of posterior
print(az.summary(inference))
```

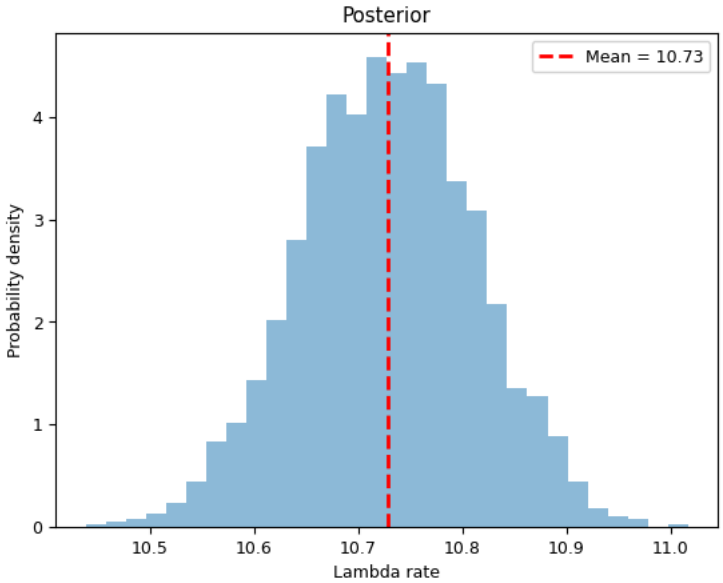
```
                    mean     sd  hdi_3%  hdi_97%  mcse_mean  mcse_sd  ess_bulk  \
rate              10.728  0.083  10.582   10.890      0.002    0.001    2406.0
zero_inflation     0.770  0.009   0.754    0.788      0.000    0.000    2233.0

                  ess_tail  r_hat
rate                1583.0    1.0
zero_inflation      1389.0    1.0
```

```
az.plot_trace(trace)
plt.subplots_adjust(hspace=0.5)

# Show the plot
plt.show()
az.summary(trace, round_to=2)
```



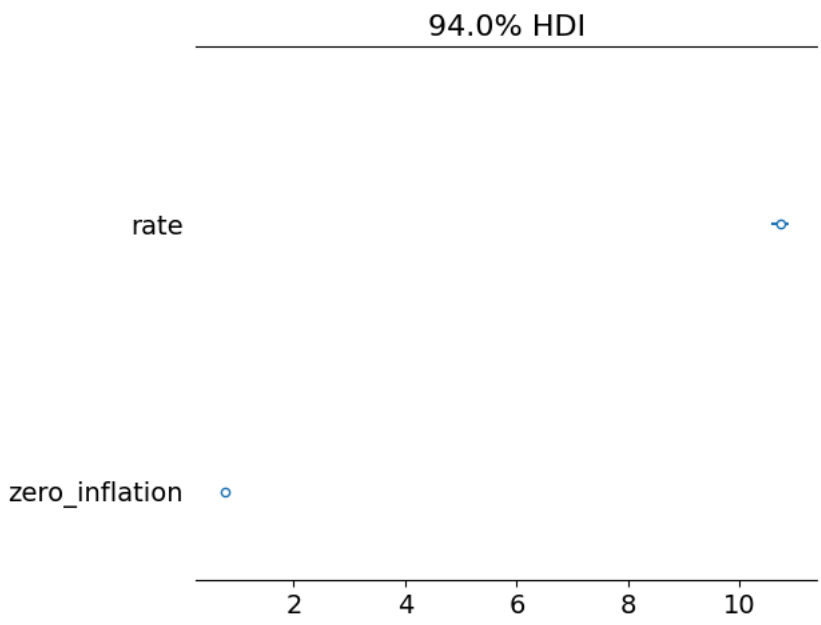
	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
rate	10.73	0.08	10.56	10.88	0.0	0.0	2077.83	1715.30	1.0
zero_inflation	0.77	0.01	0.75	0.79	0.0	0.0	1903.34	1315.64	1.0

```
plt.figure()
plt.title('Posterior')
plt.hist(inference.posterior.rate.values.flat, bins=30,density=True, alpha=0.5)
plt.xlabel('Lambda rate')
plt.ylabel('Probability density')
mean_value = inference.posterior.rate.mean().values
plt.axvline(mean_value, color='red', linestyle='dashed', linewidth=2, label=f'Mean = {mean_value:.2f}')

# Display legend
plt.legend()
plt.show()
```



Each vertical axis represents the probability density of possible values for a specific parameter. The horizontal axis shows the range of possible values for the parameter. A mean of 10.73 tells us that it is the most probable value, the distribution is close to being symmetrical and is not heavily skewed. The sampler seems to be working well and the posterior mean aligns fairly well with the observed data.

```
az.plot_forest(trace, combined=True);
```



Posterior Predictive

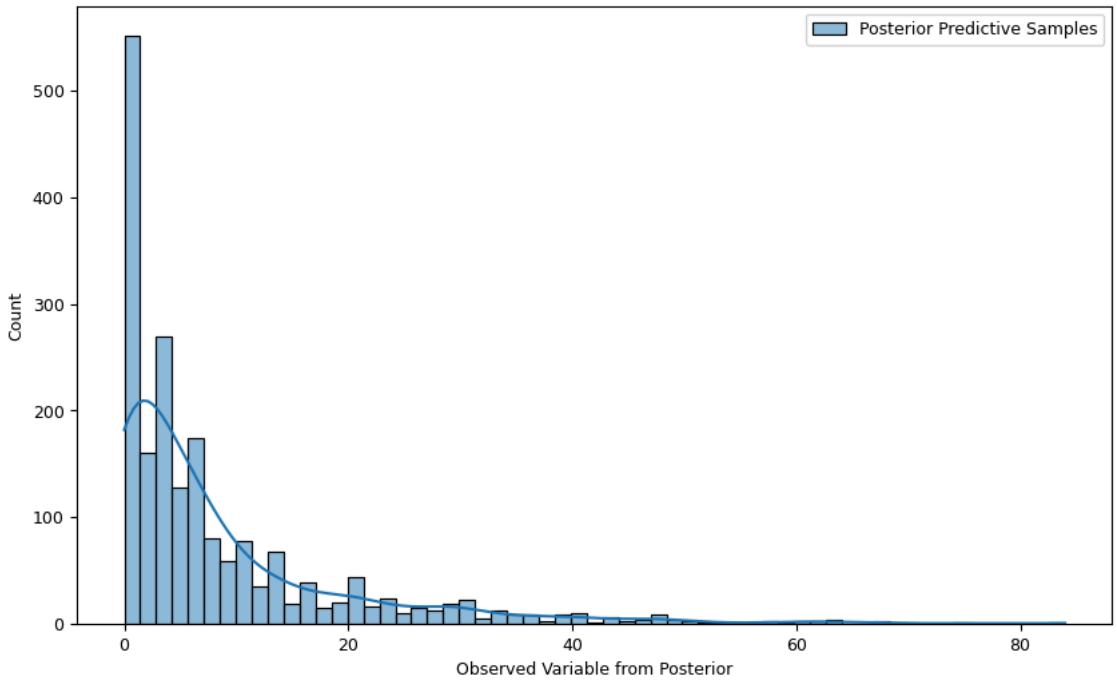
```
posterior_predictive1={}
with complete_pooling_model:
    # Set the data using pm.set_data
    posterior_predictive1 = pm.sample_posterior_predictive(trace)
```

100.00% [2000/2000 00:35<00:00]

```
print(posterior_predictive1.keys())
pm.sample_posterior_predictive
```

```
KeysView(Inference data with groups:
        > posterior_predictive
        > observed_data
        > constant_data)
<function pymc.sampling.forward.sample_posterior_predictive(trace, model: Optional[pymc.model.Model] = None, var_names:
Optional[List[str]] = None, sample_dims: Optional[List[str]] = None, random_seed: Union[int, Sequence[int], numpy.ndarray,
NoneType, numpy.random.mtrand.RandomState, numpy.random._generator.Generator] = None, progressbar: bool = True,
return_inferencedata: bool = True, extend_inferencedata: bool = False, predictions: bool = False, idata_kwargs: dict =
None, compile_kwargs: dict = None) -> Union[arviz.data.inference_data.InferenceData, Dict[str, numpy.ndarray]]>
```

```
# Assuming 'observed_data' is the key for the observed variable
posteriorP1_samples_observed = posterior_predictive1['observed_data']

# Convert the xarray.Dataset to a numpy array
posteriorP1_samples_array = posteriorP1_samples_observed.to_array().values

# 1. Plot Posterior Predictive Samples
plt.figure(figsize=(10, 6))
sns.histplot(posteriorP1_samples_array.flatten(), kde=True, label='Posterior Predictive Samples')
plt.xlabel('Observed Variable from Posterior')
plt.legend()
plt.show()
```

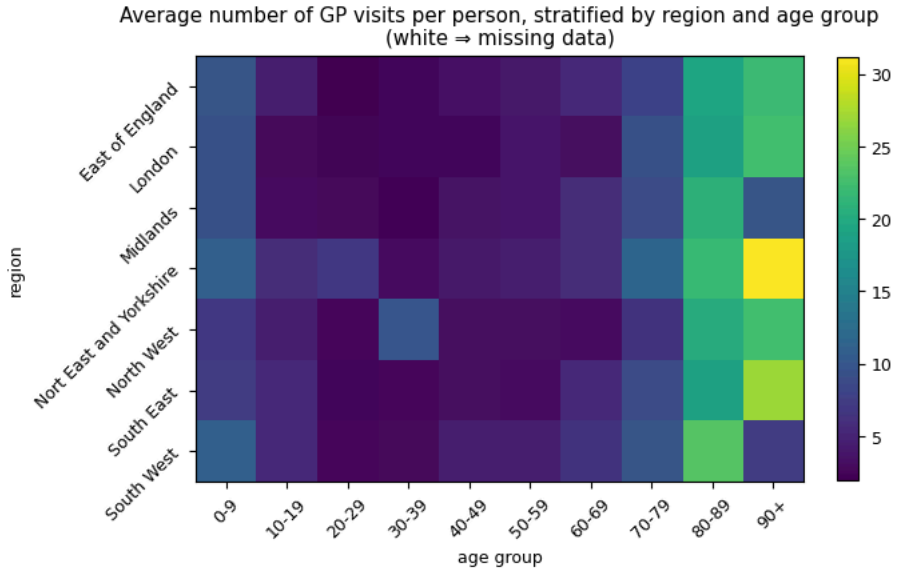


The posterior predictive samples are in-line with what we expected. We have a much higher number of zeros and the density falls as the number of visits increase since we have a little part of the population that goes to the hospital in higher numbers.

Estimating the missing data:

In the code below, I went through a copy of our original dataset and for each value that was missing, I populated the array with new values that were sampled from our posterior predictive. Our predictions are in-line with the mean of the total data but don't blend well since we assume each cluster has the exact same underlying parameters. Thus our predictions are as we would expect.

```
copiedarr = raw_data.copy()
counts_data = posterior_predictive1['observed_data']['counts'].data

    # If any element is NaN, replace all values with samples from the posterior predictive
for i in range(7):
    for j in range(10):
        if np.isnan(copiedarr[i][j]).all():
            # If all values in the block are NaN, replace the entire block with counts_data
            copiedarr[i][j] = np.random.choice(counts_data, size=30, replace=False)
# Now 'copiedarr' contains blocks replaced with samples from posterior predictive
#print(copiedarr)

plt.figure()
plt.title('Average number of GP visits per person, stratified by region and age group\n(white ⇒ missing data)')
plt.xlabel('age group')
plt.ylabel('region')
plt.imshow(copiedarr.mean(axis=2))
plt.colorbar(fraction=0.032)
plt.xticks(range(len(ages)), ages, rotation=45)
plt.yticks(range(len(regions)), regions, rotation=45)
plt.show()
```



Model 2: Partial Pooling

```
with pm.Model() as partial_pooling_model:
    # Hyperprior for the overall mean
    mu_hyperprior = pm.Normal('mu_hyperprior', mu=np.mean(observed), sigma=5)

    # Hyperprior for the overall standard deviation
    sigma_hyperprior = pm.HalfNormal('sigma_hyperprior', sigma=5)

    # Group-specific parameters
    mu = pm.Normal('mu', mu=mu_hyperprior, sigma=sigma_hyperprior)
    sigma = pm.HalfNormal('sigma', sigma=sigma_hyperprior)

    # Group-specific mean rate for each combination of age and geographical region
    lambda_i = pm.Normal('lambda_i', mu=mu, sigma=sigma, shape=(7, 10))

    psi = pm.Beta('psi', alpha=1, beta=1)

    #data
    observed_counts = observed
    age_indices, region_indices = np.meshgrid(np.arange(7), np.arange(10), indexing='ij')
    age = age_indices.flatten()
    region = region_indices.flatten()

    # Likelihood (Zero-Inflated Poisson)
    theta_np = np.exp(mu_hyperprior + lambda_i[i, j])
    theta = theta_np.flatten()
    y_obs = pm.ZeroInflatedPoisson('y_obs', psi=psi, mu=theta, observed=observed_counts)

    partial_trace = pm.sample()
```

```
100.00% [2000/2000 15:18<00:00 Sampling chain 0, 0 divergences]
100.00% [2000/2000 13:59<00:00 Sampling chain 1, 0 divergences]
```

```
partial_trace
```

arviz.InferenceData

▶ posterior

▶ sample_stats

▶ observed_data

Explaining the model: Hyperpriors:

alpha_rate: Represents the mean of the hyperprior for the alpha parameter. beta_rate: Represents the standard deviation of the hyperprior for the alpha parameter. Priors:

alpha: Represents the group-level parameters for the Zero-Inflated Poisson distribution. It has n elements, where n is the number of groups. Each element is drawn from a normal distribution with a mean of alpha_rate and a standard deviation of beta_rate.

mu: Represents the mean of the Zero-Inflated Poisson distribution. It is a deterministic transformation of alpha, ensuring that it is positive. Likelihood:

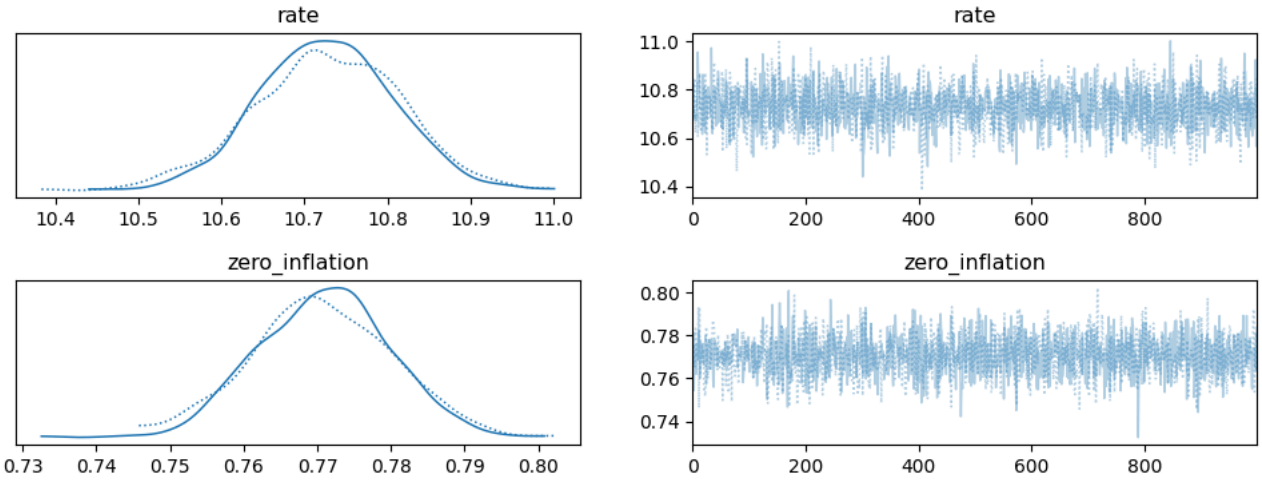
psi: Represents the probability of observing zero in the Zero-Inflated Poisson distribution. It is drawn from a Beta distribution. counts: Represents the observed counts. The likelihood is modeled using a Zero-Inflated Poisson distribution as instructed with a mean given by mu[data_m], where data_m is used to index the appropriate group mean based on the age group.

Posteriors

```
inference_2 = {}  # store the results here
with partial_pooling_model:
    inference_2 = pm.sample()
```

```
100.00% [2000/2000 15:29<00:00 Sampling chain 0, 0 divergences]
100.00% [2000/2000 16:25<00:00 Sampling chain 1, 0 divergences]
```

```
az.plot_trace(trace)
plt.subplots_adjust(hspace=0.5)

# Show the plot
plt.show()
az.summary(partial_trace, round_to=2)
```

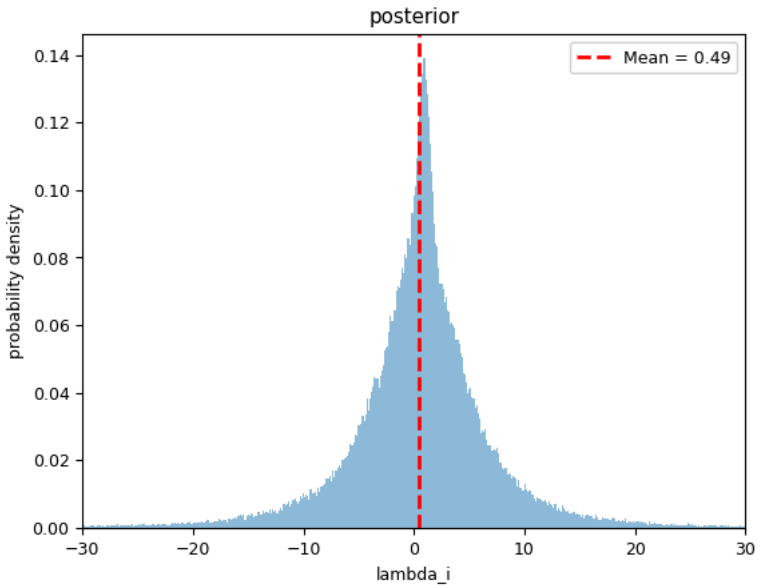


	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
mu_hyperprior	2.82	2.46	-1.12	7.45	0.35	0.25	51.40	45.30	1.03
mu	0.79	2.49	-4.57	5.46	0.83	0.61	8.95	33.85	1.16
lambda_i[0, 0]	0.80	4.33	-6.63	9.72	0.87	0.63	20.54	62.22	1.08
lambda_i[0, 1]	0.57	3.84	-5.95	7.87	0.50	0.48	37.70	52.68	1.06
lambda_i[0, 2]	0.90	4.19	-5.82	8.79	0.89	0.64	20.26	59.11	1.07
...	...	...	...	...	...	...	...	...	...
lambda_i[6, 8]	0.89	4.07	-5.08	8.72	0.94	0.68	16.83	45.06	1.08
lambda_i[6, 9]	-0.45	2.46	-5.08	3.49	0.35	0.25	51.43	45.93	1.03
sigma_hyperprior	3.96	2.50	0.29	8.50	0.36	0.26	42.81	125.29	1.03
sigma	2.28	2.50	0.14	7.57	0.95	0.70	5.79	34.63	1.29
psi	0.77	0.01	0.75	0.79	0.00	0.00	557.23	611.20	1.00

75 rows × 9 columns

```
# Display summary of posterior
print(az.summary(inference_2))
plt.figure()
plt.title('posterior')
plt.hist(inference_2.posterior.lambda_i.values.flat, bins=1000,density=True, alpha=0.5)
plt.xlabel('lambda_i')
plt.ylabel('probability density')
plt.xlim(-30, 30)
mean_value = inference_2.posterior.lambda_i.mean().values
plt.axvline(mean_value, color='red', linestyle='dashed', linewidth=2, label=f'Mean = {mean_value:.2f}')

# Display legend
plt.legend()
plt.show()
```

```
                    mean     sd  hdi_3%  hdi_97%  mcse_mean  mcse_sd  ess_bulk  \
mu_hyperprior      3.239  3.037  -2.274    8.814      0.308    0.218      98.0
mu                 0.507  4.174  -7.997    7.889      0.798    0.570      30.0
lambda_i[0, 0]     0.456  6.792 -14.348   12.815      0.779    0.876      83.0
lambda_i[0, 1]     0.727  6.762 -11.515   14.830      0.508    0.720     117.0
lambda_i[0, 2]     0.554  6.736 -12.284   13.448      0.707    0.643      95.0
...                  ...    ...     ...      ...        ...      ...       ...
lambda_i[6, 8]     0.450  6.632 -12.072   12.866      0.746    0.742      92.0
lambda_i[6, 9]    -0.865  3.037  -6.475    4.616      0.308    0.219      98.0
sigma_hyperprior   4.882  2.945   0.436   10.043      0.872    0.633       9.0
sigma              4.355  3.434   0.403   10.568      1.428    1.065       5.0
psi                0.771  0.009   0.755    0.787      0.000    0.000     500.0

                   ess_tail  r_hat
mu_hyperprior         460.0   1.03
mu                     21.0   1.11
lambda_i[0, 0]         36.0   1.09
lambda_i[0, 1]         55.0   1.10
lambda_i[0, 2]         63.0   1.09
...                     ...    ...
lambda_i[6, 8]         48.0   1.11
lambda_i[6, 9]        460.0   1.03
sigma_hyperprior       21.0   1.17
sigma                  21.0   1.37
psi                  1043.0   1.01

[75 rows x 9 columns]
```



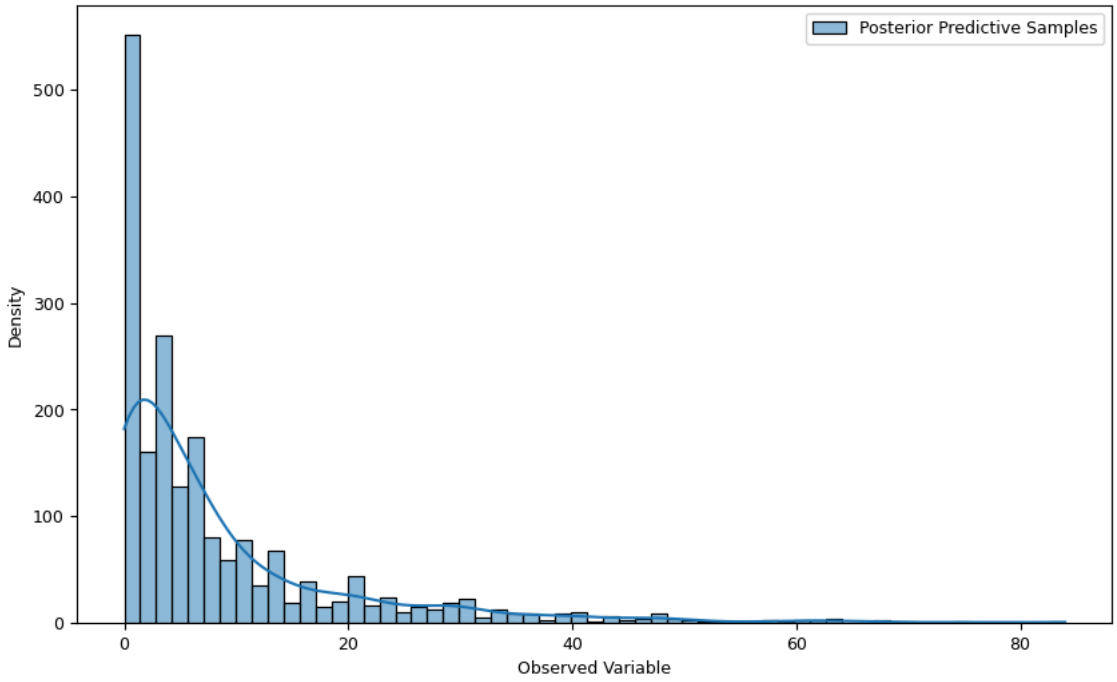
Each vertical axis represents the probability density of possible values for a specific parameter. The horizontal axis shows the range of possible values for the parameter. The distribution is close to being symmetrical and is not heavily skewed. The sampler seems to be working well as well.

Posterior Predictive

```
posterior_predictive2={}
with complete_pooling_model:
    # Set the data using pm.set_data
    #pm.set_data({'data_n': age})
    posterior_predictive2 = pm.sample_posterior_predictive(partial_trace)
```

100.00% [2000/2000 00:28<00:00]

```
# Assuming 'observed_data' is the key for the observed variable
posteriorP2_samples_observed = posterior_predictive2['observed_data']

# Convert the xarray.Dataset to a numpy array
posteriorP2_samples_array = posteriorP2_samples_observed.to_array().values

# 1. Plot Posterior Predictive Samples
plt.figure(figsize=(10, 6))
sns.histplot(posteriorP2_samples_array.flatten(), kde=True, label='Posterior Predictive Samples')
plt.xlabel('Observed Variable')
plt.ylabel('Density')
plt.legend()
plt.show()
```



The posterior predictive samples are in-line with what we expected. We have a much higher number of zeros and the density falls as the number of visits increase since we have a little part of the population that goes to the hospital in higher numbers.

Visualizing the results:

```
copiedarr2 = raw_data.copy()
counts_data2 = posterior_predictive2['observed_data']['counts'].data

    # If any element is NaN, replace all values with samples from the posterior predictive
for i in range(7):
    for j in range(10):
        if np.isnan(copiedarr2[i][j]).all():
            # If all values in the block are NaN, replace the entire block with counts_data
            copiedarr2[i][j] = np.random.choice(counts_data2, size=30, replace=False)
# Now 'copiedarr2' contains blocks replaced with samples from posterior predictive
plt.figure()
plt.title('Average number of GP visits per person, stratified by region and age group\n(white ⇒ missing data)')
plt.xlabel('age group')
plt.ylabel('region')
plt.imshow(copiedarr2.mean(axis=2))
plt.colorbar(fraction=0.032)
plt.xticks(range(len(ages)), ages, rotation=45)
plt.yticks(range(len(regions)), regions, rotation=45)
plt.show()
```

Comparing the results

```
plt.figure()
plt.title('Average number of GP visits per person,complete pooling')
plt.xlabel('age group')
plt.ylabel('region')
plt.imshow(copiedarr.mean(axis=2))
plt.colorbar(fraction=0.032)
plt.xticks(range(len(ages)), ages, rotation=45)
plt.yticks(range(len(regions)), regions, rotation=45)
plt.show()

plt.figure()
plt.title('Average number of GP visits per person, partial pooling')
plt.xlabel('age group')
```