glm_roman_exam2

April 28, 2024

1 Second Partial Exam GLM

@roman avj

April 27, 2024

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os

import pymc as pm
import arviz as az
import xarray as xr
```

```
WARNING (pytensor.tensor.blas): Using NumPy C-API based implementation for BLAS
functions.
```

---

2 Q1: Tenis Data

The following table shows the sample size n_i and the average service time $\bar{y}_i$ (in seconds) for six professional tennis players. Suppose that the sample mean for player i and $\bar{y}_i$ is normally distributed with mean μ_i and standard deviation $\sigma/\sqrt{n_i}$ where $\sigma = 5.5$ seconds.

[ ]:
```
# Load the data
df_tenis_players = pd.DataFrame({
    'Player': ['Murray', 'Simon', 'Federer', 'Ferrer', 'Isner', 'Kyrgios'],
    'n': [731, 570, 491, 456, 403, 274],
    'y': [23.56, 18.07, 16.21, 21.7, 22.31, 14.11]
}).set_index('Player')

df_tenis_players
```

[ ]:
```
           n      y
Player
Murray   731  23.56
```

Simon	570	18.07
Federer	491	16.21
Ferrer	456	21.70
Isner	403	22.31
Kyrgios	274	14.11

2.1 Q1.A: Murray's average service time

Is of interest to estimate the average service time of Murray μ_1. Find the posterior distribution of μ_1 and construct a 90% credible interval for μ_1.

[ ]:
```python
# model
with pm.Model() as model:
    # priors
    mu = pm.Normal('mu_murray', mu=20, sigma=10)
    sigma = pm.ConstantData('sigma_murray', 5.5)
    n_obs = pm.ConstantData('n_obs', df_tenis_players.loc['Murray', 'n'])

    # likelihood
    likelihood = pm.Normal(
        'y',
        mu=mu,
        sigma=sigma / pm.math.sqrt(n_obs),
        observed=df_tenis_players.loc['Murray', 'y']
        )

    # sample
    trace = pm.sample(1000, tune=1000, cores=1, chains=4)
    trace = pm.sample_posterior_predictive(
        trace, extend_inferencedata=True, random_seed=42
        )
```

```
Auto-assigning NUTS sampler…
Initializing NUTS using jitter+adapt_diag…
Sequential sampling (4 chains in 1 job)
NUTS: [mu_murray]
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

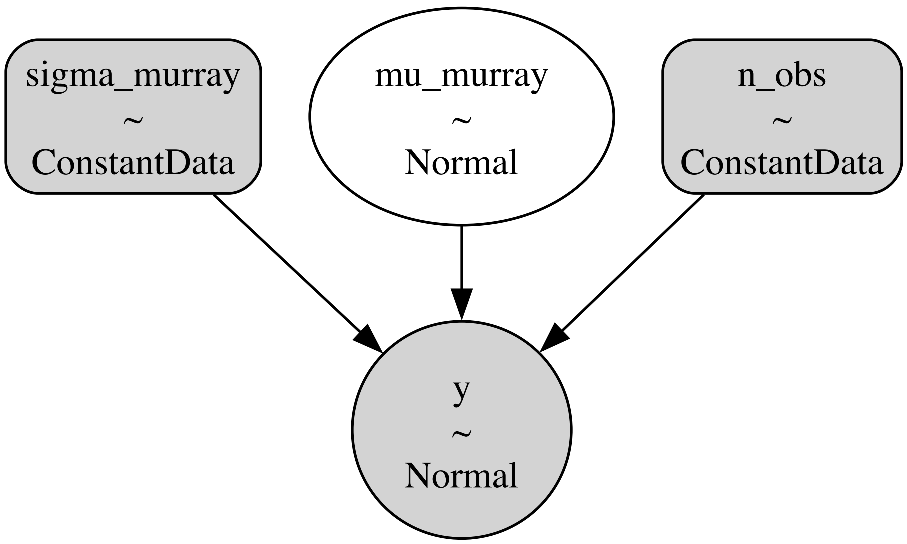
```
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws
total) took 2 seconds.
Sampling: [y]
```

```
<IPython.core.display.HTML object>
```

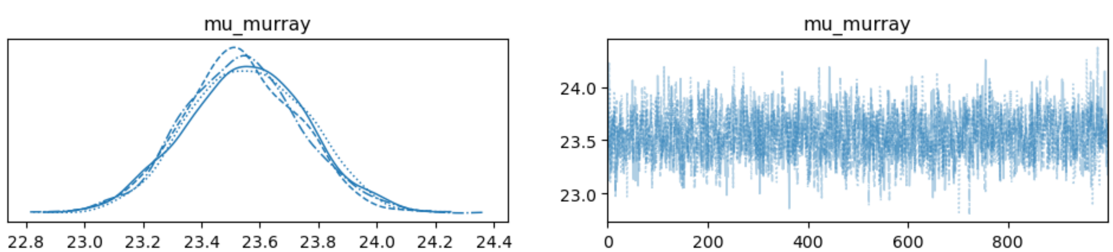
```
<IPython.core.display.HTML object>
```

[ ]:
```
# look causal graph
pm.model_to_graphviz(model)
```

[ ]:



[ ]:
```
# posterior predictive check
az.plot_trace(trace, var_names=["mu_murray"])
plt.show()
```



[ ]:
```
# summary of the trace
az.summary(trace,  var_names=["mu_murray"], hdi_prob=0.9)
```

[ ]:
```
             mean     sd  hdi_5%  hdi_95%  mcse_mean  mcse_sd  ess_bulk  \
mu_murray  23.547  0.203  23.211   23.873      0.005    0.004    1681.0
```

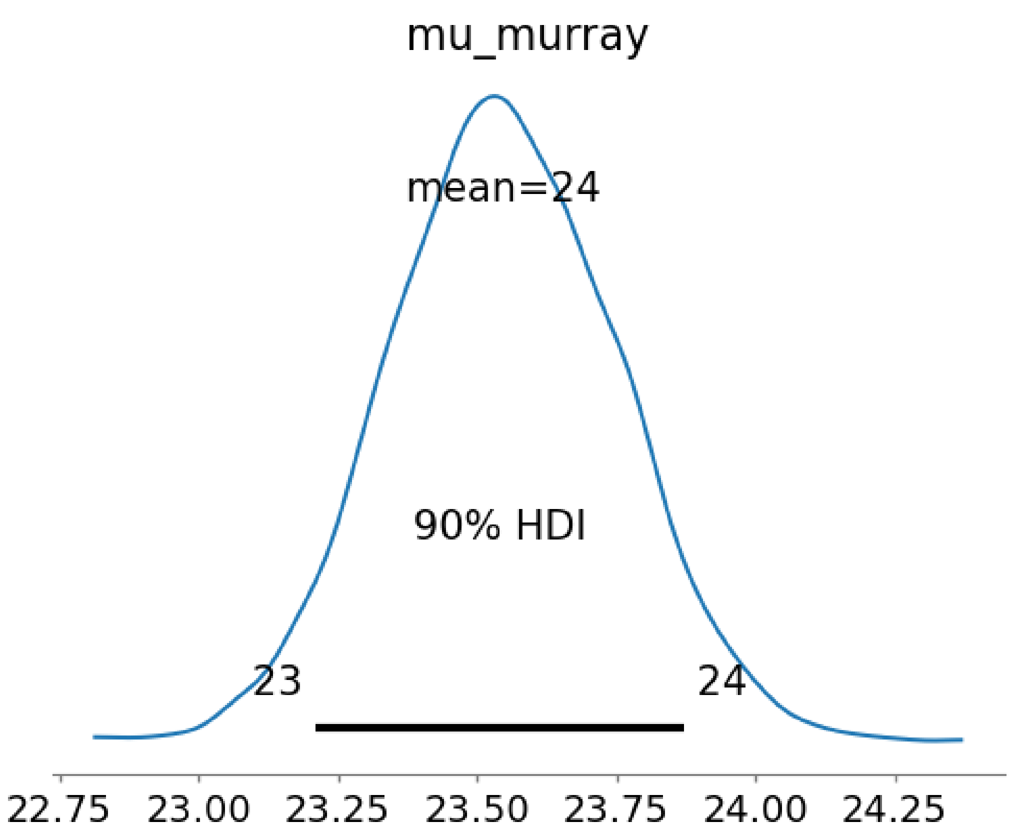
```
            ess_tail  r_hat
mu_murray     2608.0    1.0
```

[ ]:
```
# interval length
23.886-23.222
```

[ ]: 0.6639999999999979

[ ]:
```
# plot posterior distribution for coefficients
az.plot_posterior(trace, var_names=["mu_murray"], hdi_prob=0.9)
```

[ ]: <Axes: title={'center': 'mu_murray'}>



We can observe that inferred Murray's average time is 23.54 seconds with a 90% credible interval of [23.21, 23.87] seconds. The interval length is 0.664 seconds.

2.2 Q1.B: General model for all players

Assume that the average service time for all players is the same, $\mu_1 = \cdots = \mu_6 = \mu$. The average service time for each player is $\bar{y} = 19.9$ with a combined sample size of $n = 2925$. Suppose that

μ has an initial distribution $N(20, 10)$. Find the posterior distribution of μ and construct a 90% credible interval for μ.

[ ]:
```
# get average time of players
avg_all_players_time = np.sum(df_tenis_players['y'] * df_tenis_players['n']) /
 ↪np.sum(df_tenis_players['n'])
n_obs_all_players = np.sum(df_tenis_players['n'])

print(f"Average time of all players: {avg_all_players_time:.2f}")
print(f"Number of observations of all players: {n_obs_all_players}")
```

```
Average time of all players: 19.91
Number of observations of all players: 2925
```

[ ]:
```
# model
with pm.Model() as model:
    # priors
    mu = pm.Normal('mu_players', mu=20, sigma=10)
    sigma = pm.ConstantData('sigma_players', 5.5)
    n_obs = pm.ConstantData('n_obs', n_obs_all_players)

    # likelihood
    likelihood = pm.Normal(
        'y',
        mu=mu,
        sigma=sigma / pm.math.sqrt(n_obs),
        observed=avg_all_players_time
        )

    # sample
    trace = pm.sample(1000, tune=1000, cores=1, chains=4)
    trace = pm.sample_posterior_predictive(
        trace, extend_inferencedata=True, random_seed=42
        )
```

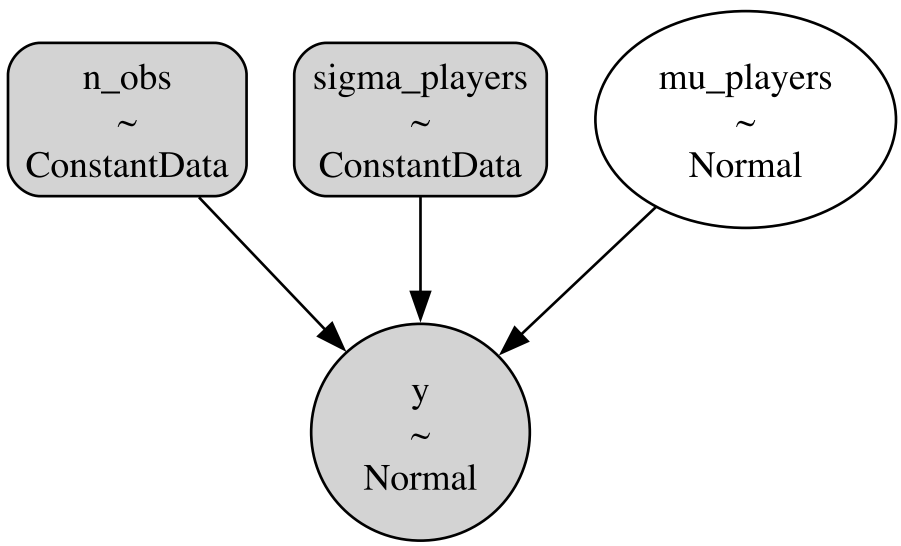
```
Auto-assigning NUTS sampler…
Initializing NUTS using jitter+adapt_diag…
Sequential sampling (4 chains in 1 job)
NUTS: [mu_players]

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws
total) took 2 seconds.
Sampling: [y]

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

[ ]:

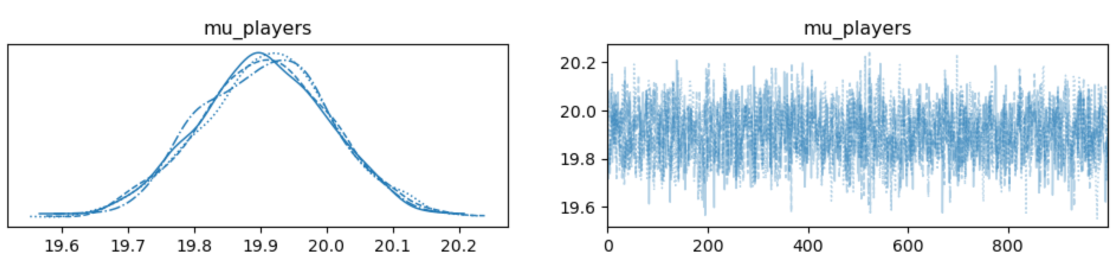
```
# look causal graph
pm.model_to_graphviz(model)
```

[ ]:



[ ]:

```
# posterior predictive check
az.plot_trace(trace, var_names=["mu_players"])
plt.show()
```



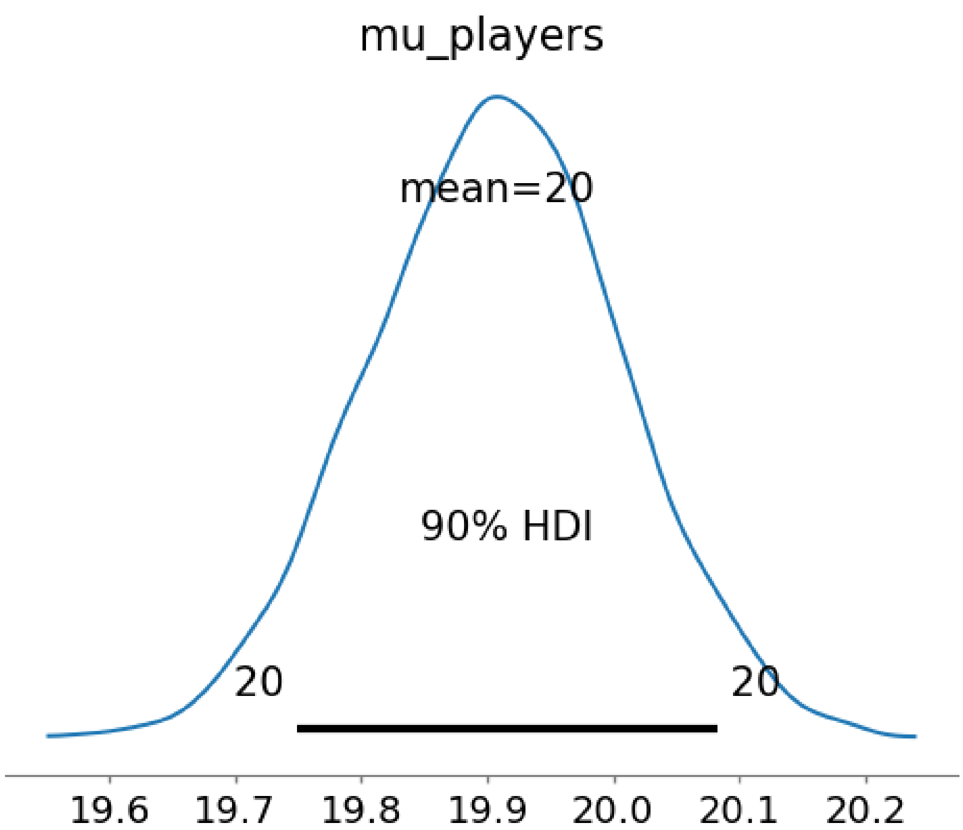
[ ]:

```
# summary of the trace
az.summary(trace,  var_names=["mu_players"], hdi_prob=0.9)
```

```
[ ]:              mean     sd  hdi_5%  hdi_95%  mcse_mean  mcse_sd  ess_bulk  \
mu_players  19.906  0.102  19.749   20.082      0.003    0.002    1574.0

            ess_tail  r_hat
mu_players    2758.0    1.0
```

```python
[ ]: # plot posterior distribution for coefficients
     az.plot_posterior(trace, var_names=["mu_players"], hdi_prob=0.9)
```

```
[ ]: <Axes: title={'center': 'mu_players'}>
```



The average service time of all players is 19.91 with a 90% credible interval of [19.22, 20.61]. The interval length is 1.39 seconds.

2.3 Q1.C: Statement

Which approach, part (a) or part (b), seems more reasonable in this situation?

In my humble opinion, depends on what you want to know. If you want to know the average time of all players, you can use the average time of all players as the observed value (pooled data). If you want to know the average time of a specific player (b), you can use the observed value of that player (a) (unpooled data). Is a bias-variance trade-off.

---

3 Q2: Hierarchical model

Continuing with the previous problem, suppose that you want to estimate the average service time for the six tennis players using a hierarchical model. Recall that $\sigma = 5.5$ seconds.

$$\bar{y}_i \sim N(\mu_i, \sigma/\sqrt{n_i}), \quad i = 1, \dots, 6\mu_i \sim N(\mu, \tau), \quad i = 1, \dots, 6\mu \sim N(20, 1/0.0001), \quad 1/\tau^2 \sim G(0.01, 0.1)$$

3.1 Q2.A: Posterior Distribution

Use pyMC, Stan or JAGS to simulate a sample of size 1000 from the posterior distribution of the hierarchical model

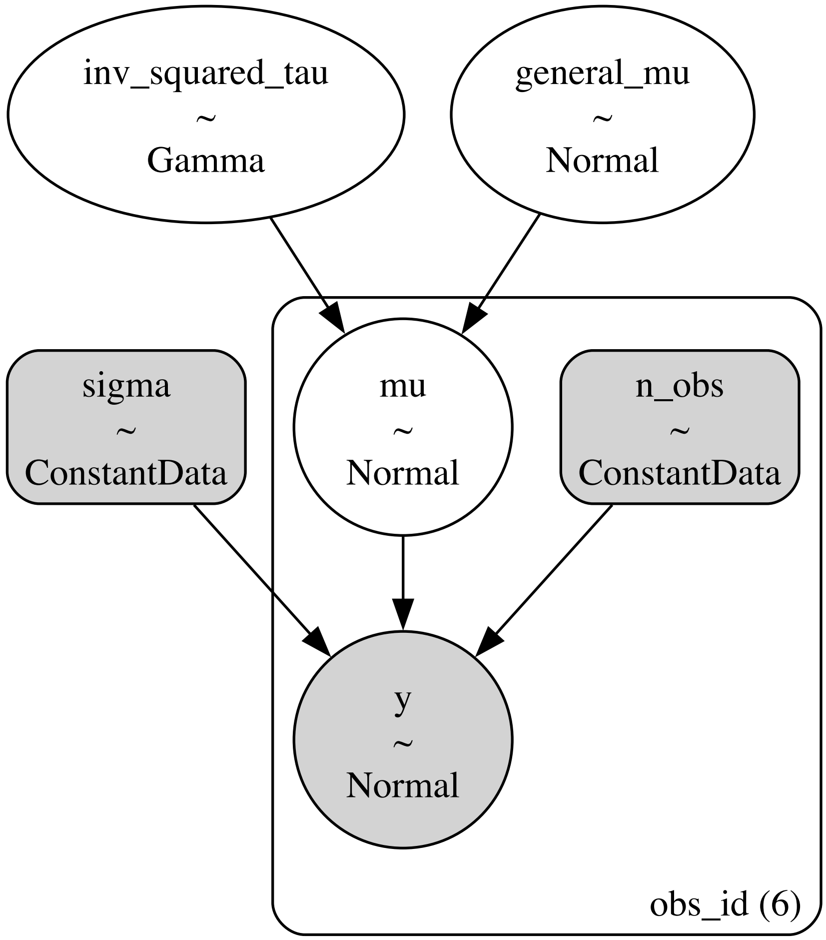
[ ]:
```
# model
with pm.Model(coords={"obs_id": df_tenis_players.index}) as model:
    # data
    n_obs = pm.ConstantData('n_obs', df_tenis_players['n'], dims='obs_id')
    sigma = pm.ConstantData('sigma', 5.5)

    # priors
    general_mu = pm.Normal('general_mu', mu=20, tau=1e-4)  # general mu for all␣
 ↪players
    inv_squared_tau = pm.Gamma('inv_squared_tau', alpha=1e-2, beta=1e-1)
    tau = 1 / pm.math.sqrt(inv_squared_tau)
    mu = pm.Normal('mu', mu=general_mu, tau=tau, dims='obs_id')

    # likelihood
    likelihood = pm.Normal(
        'y',
        mu=mu,
        sigma=sigma / pm.math.sqrt(n_obs),
        dims='obs_id',
        observed=df_tenis_players['y']
        )

    # sample
    trace = pm.sample(1000, tune=1000, cores=1, chains=4)
    trace = pm.sample_posterior_predictive(
        trace, extend_inferencedata=True, random_seed=42
        )
```

```
Auto-assigning NUTS sampler…
Initializing NUTS using jitter+adapt_diag…
Sequential sampling (4 chains in 1 job)
NUTS: [general_mu, inv_squared_tau, mu]
```

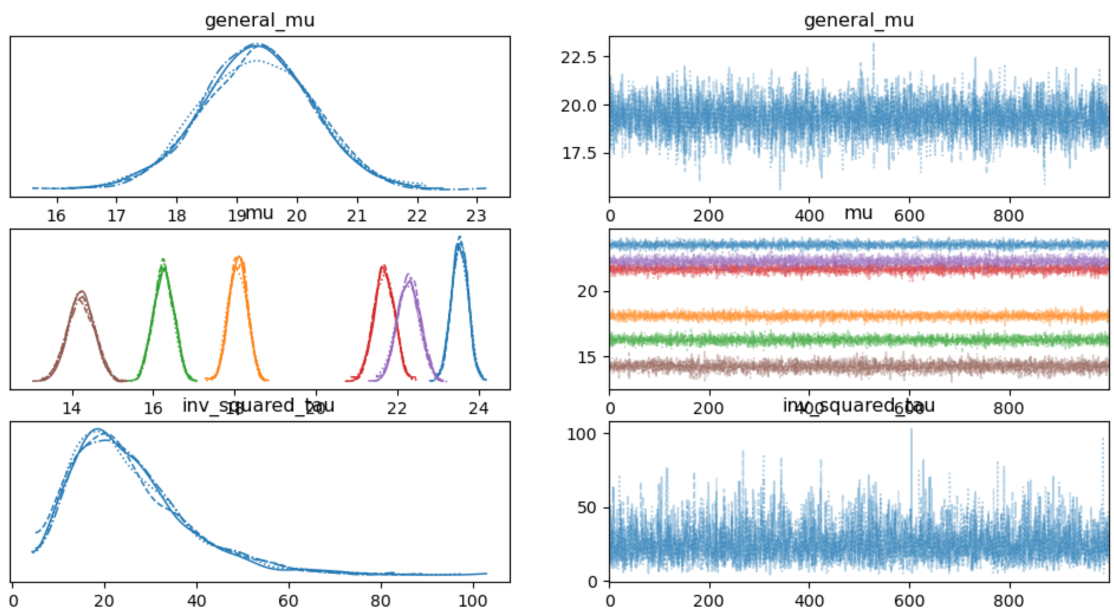
```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws
total) took 5 seconds.
Sampling: [y]

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

[ ]:
```
# look causal graph
pm.model_to_graphviz(model)
```

[ ]:



[ ]:
```
# posterior predictive check
az.plot_trace(trace)
plt.show()
```



The former 1000 simulations were to get the average time of all players and the individual times

3.2 Q2.B: Build Credibility Intervals for the average time of each player

[ ]:
```
# get traze of each player average time
az.summary(trace, var_names=["mu"], hdi_prob=0.9)
```

[ ]:
```
                 mean     sd  hdi_5%  hdi_95%  mcse_mean  mcse_sd  ess_bulk  \
mu[Murray]     23.522  0.202  23.201   23.856      0.003    0.002    4908.0
mu[Simon]      18.081  0.226  17.715   18.452      0.003    0.002    4926.0
mu[Federer]    16.252  0.249  15.814   16.631      0.003    0.002    5131.0
mu[Ferrer]     21.665  0.252  21.264   22.085      0.003    0.002    5424.0
mu[Isner]      22.265  0.277  21.816   22.732      0.004    0.003    5137.0
mu[Kyrgios]    14.226  0.331  13.685   14.777      0.005    0.003    4774.0

             ess_tail  r_hat
mu[Murray]     3467.0    1.0
mu[Simon]      3046.0    1.0
mu[Federer]    3081.0    1.0
mu[Ferrer]     3241.0    1.0
mu[Isner]      3233.0    1.0
mu[Kyrgios]    3035.0    1.0
```

[ ]:
```
# plot mean estimates with its HDP for each player
df_player_means = trace.posterior['mu'].mean(dim=['chain', 'draw']).
 ↪to_dataframe().reset_index()
```

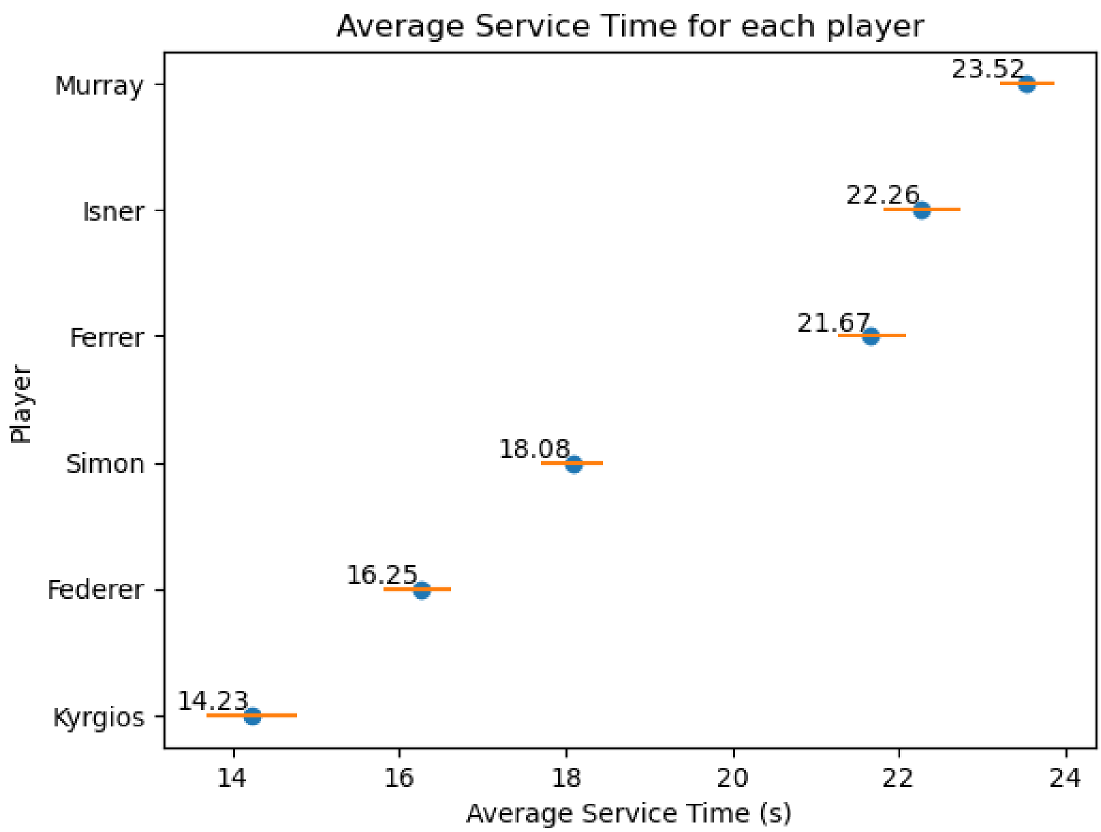
```
df_players_hdi = az.hdi(trace.posterior['mu'], hdi_prob=0.9).to_dataframe().
 ↪reset_index().pivot(index='obs_id', columns='hdi', values='mu')

# join
df_player_means_data = df_player_means.merge(df_players_hdi, on='obs_id').
 ↪sort_values('mu')
df_player_means_data
```

[ ]:

	obs_id	mu	higher	lower
5	Kyrgios	14.225799	14.776800	13.685396
2	Federer	16.252415	16.630786	15.813900
1	Simon	18.080560	18.452169	17.715065
3	Ferrer	21.665195	22.085468	21.263903
4	Isner	22.264748	22.731953	21.815503
0	Murray	23.522058	23.856077	23.201106

[ ]:

```
# plot service time
fig, ax = plt.subplots()

# lines for HDP
ax.hlines(
    y=df_player_means_data['obs_id'],
    xmin=df_player_means_data['lower'],
    xmax=df_player_means_data['higher'],
    color='C1', label='HDP'
    )

# scatter on mu
ax.scatter(df_player_means_data['mu'], df_player_means_data['obs_id'],␣
 ↪label='Mean estimate')

# add time of all players above the point as text
for i, row in df_player_means_data.iterrows():
    ax.text(row['mu'], row['obs_id'], f"{row['mu']:.2f}", ha='right',␣
 ↪va='bottom')

# plot
ax.set_xlabel('Average Service Time (s)')
ax.set_ylabel('Player')

# title
ax.set_title('Average Service Time for each player')

# show
plt.show()
```



As we can see, the average service time for each player is different. Kyrgios is the fastes but with the higher uncertainty, while Murray is the slowest but with the lower uncertainty. The rest of the players are in between.

3.3 Q2.C: Compare Murray's service time

Compare the estimated average time of Murray using the hierarchical model with the results obtained in Q1.A.

[ ]:
```
# summary of the trace
az.summary(trace,  var_names=["mu"], coords={'obs_id': ['Murray']}, hdi_prob=0.
 ↪9)
```

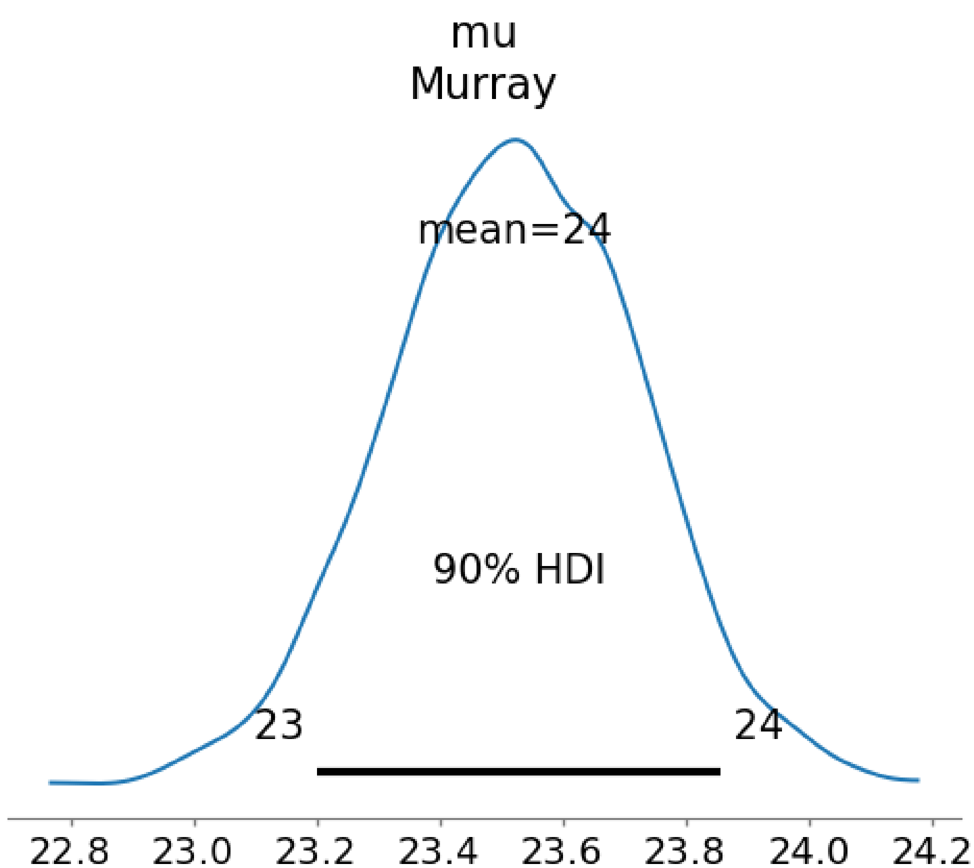
[ ]:
```
               mean     sd  hdi_5%  hdi_95%  mcse_mean  mcse_sd  ess_bulk  \
mu[Murray]  23.522  0.202  23.201   23.856      0.003    0.002    4908.0

            ess_tail  r_hat
mu[Murray]    3467.0    1.0
```

[ ]:
```
# interval length
23.848-23.188
```

[ ]: 0.6600000000000001

[ ]:
```
# plot posterior distribution for coefficients only for the mu with obs_id =␣
 ↪Murray
az.plot_posterior(trace, var_names=["mu"], coords={'obs_id': ['Murray']},␣
 ↪hdi_prob=0.9)
```

[ ]: <Axes: title={'center': 'mu\nMurray'}>



The mean is slightly lower in the hierarchical model (2.a) than in the incomplete model (1.a). The HDI are similar, however the longitude of the hierarchical model (2.a) is slightly shorter, by only 0.003 seconds.

---

4 Q3: Fire Calls in Pennsylvania

The below displays the number of fire calls and the number of building fires for ten counties in Montgomery County, Pennsylvania from 2015 through 2019. This data is currently described as Emergency - 911 Calls" from kaggle.com Suppose that the number of building fires for the j-th zip code is Poisson with mean $n_j \lambda_j$, where n_j and λ_j are respectively the number of fire calls and rate of building fires for the j-th zip code.

[ ]:
```
# data
df_fire_calls = (
    pd.DataFrame({
        'zip_code': [18054, 18103, 19010, 19025, 19040, 19066, 19116, 19406,␣
 ↪19428, 19474],
        'fire_calls': [266, 1, 1470, 246, 1093, 435, 2, 2092, 2025, 4],
        'building_fires': [12, 0, 59, 11, 47, 26, 0, 113, 73, 1]
    })
    .assign(building_fire_rates=lambda x: x['building_fires'] / x['fire_calls'])
    .assign(
        building_fire_rates= lambda x: np.where(
            x['building_fire_rates'].eq(0), 0 + 1e-4, x['building_fire_rates']
        )
    )
)
df_fire_calls
```

[ ]:

	zip_code	fire_calls	building_fires	building_fire_rates
0	18054	266	12	0.045113
1	18103	1	0	0.000100
2	19010	1470	59	0.040136
3	19025	246	11	0.044715
4	19040	1093	47	0.043001
5	19066	435	26	0.059770
6	19116	2	0	0.000100
7	19406	2092	113	0.054015
8	19428	2025	73	0.036049
9	19474	4	1	0.250000

4.1 Q3.A: Posterior Distribution

Suppose that the building fire rates $\lambda_1, \ldots, \lambda_{10}$ follow a common Gamma(α, β) distribution where the hyperparameters α and β follow weakly informative distributions. Use JAGS to simulate a sample of size 5000 from the joint posterior distribution of all parameters of the model.

[ ]:
```
 # multilevel gamma poisson model
 with pm.Model(
    coords_mutable={
        "obs_id": np.arange(df_fire_calls.shape[0]),
        "zip_code": df_fire_calls["zip_code"]
        }
    ) as model:
    # Data
    fire_calls = pm.MutableData('fire_calls', df_fire_calls['fire_calls'].
 ↪values, dims='obs_id')
```

```
    zip_code_idx = pm.MutableData('zip_code_idx', np.arange(df_fire_calls.
↪shape[0]), dims='obs_id')

    # Hyperpriors
    alpha = pm.HalfNormal('alpha', sigma=10)
    beta = pm.HalfNormal('beta', sigma=10)

    # Generated Quantities
    mean_building_fire_rate = pm.Deterministic('mean_building_fire_rate', alpha␣
↪/ beta)

    # Priors
    building_fire_rates = pm.Gamma('building_fire_rates', alpha, beta,␣
↪dims='zip_code')

    # Likelihood
    likelihood = pm.Poisson(
        'building_fires',
        mu=building_fire_rates[zip_code_idx] * fire_calls,  # individual rate␣
↪for each zip code
        observed=df_fire_calls['building_fires'],
        dims='obs_id'
        )

    # Sample
    trace = pm.sample(5000, tune=1000, cores=1, chains=4)
    trace = pm.sample_posterior_predictive(
        trace, extend_inferencedata=True, random_seed=42
        )
```

```
Auto-assigning NUTS sampler…
Initializing NUTS using jitter+adapt_diag…
Sequential sampling (4 chains in 1 job)
NUTS: [alpha, beta, building_fire_rates]
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

```
Sampling 4 chains for 1_000 tune and 5_000 draw iterations (4_000 + 20_000 draws
total) took 22 seconds.
Sampling: [building_fires]
```

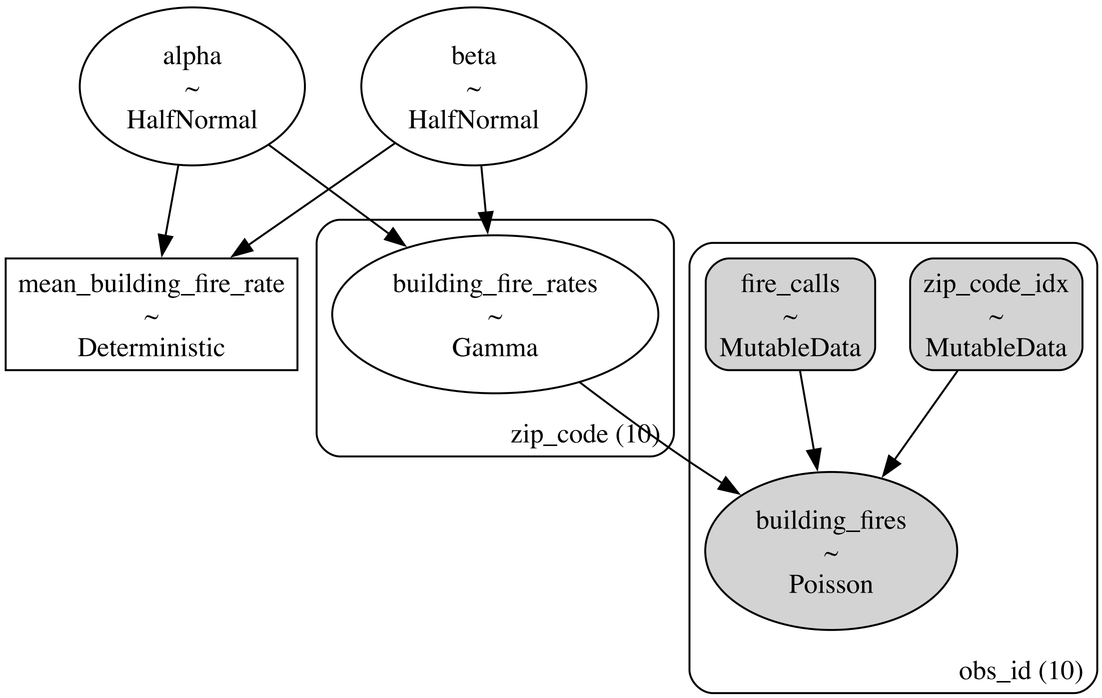
```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

[ ]:

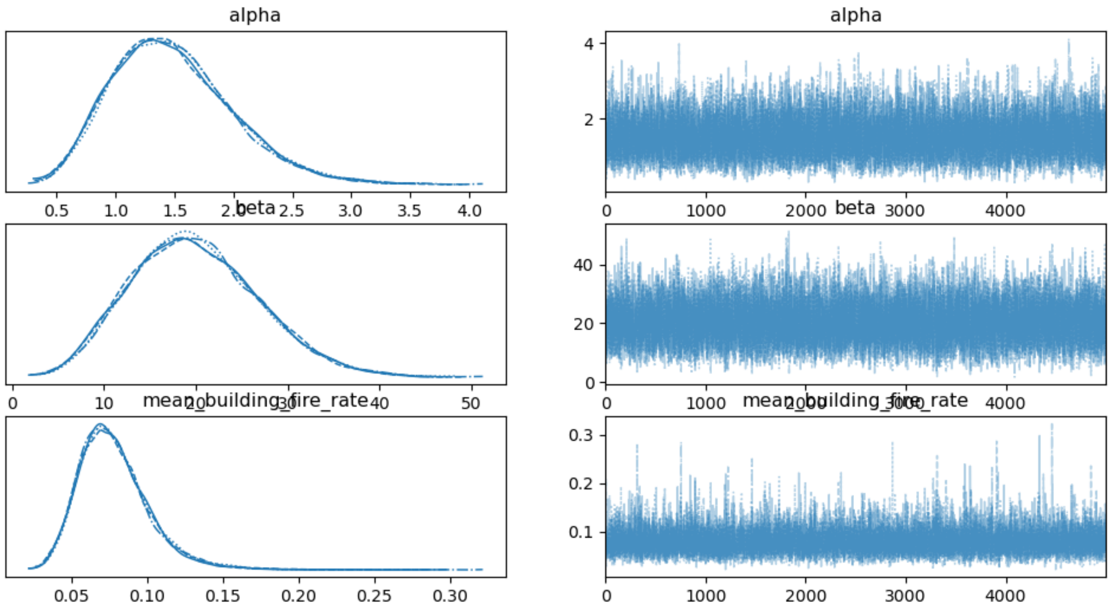
```
# look causal graph
pm.model_to_graphviz(model)
```

[ ]:



[ ]:

```
# posterior predictive check
az.plot_trace(trace, var_names=["alpha", "beta", "mean_building_fire_rate"])
plt.show()
```



[ ]:
```python
# get building rates for each zip code
az.summary(trace, var_names=["mean_building_fire_rate", "building_fire_rates"],
 hdi_prob=0.9)
```

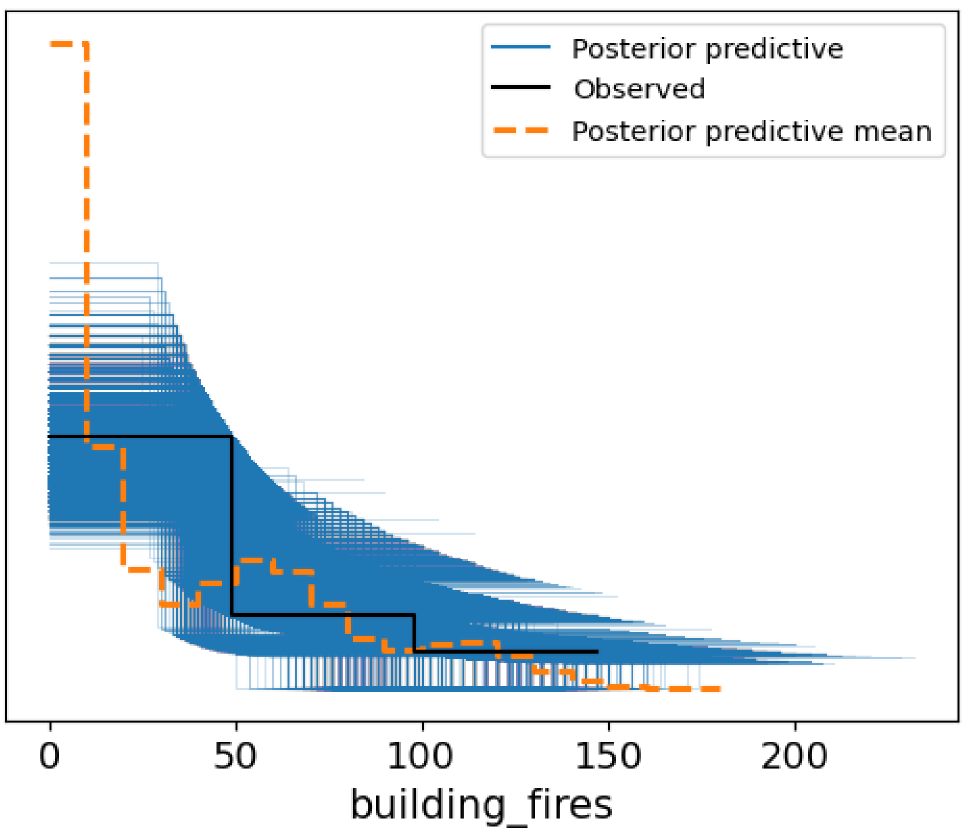
[ ]:
```
                             mean     sd  hdi_5%  hdi_95%  mcse_mean  mcse_sd  \
mean_building_fire_rate     0.078  0.024   0.042    0.114        0.0      0.0
building_fire_rates[18054]  0.047  0.013   0.026    0.067        0.0      0.0
building_fire_rates[18103]  0.074  0.068   0.000    0.160        0.0      0.0
building_fire_rates[19010]  0.041  0.005   0.032    0.049        0.0      0.0
building_fire_rates[19025]  0.047  0.013   0.025    0.068        0.0      0.0
building_fire_rates[19040]  0.044  0.006   0.034    0.054        0.0      0.0
building_fire_rates[19066]  0.061  0.012   0.041    0.079        0.0      0.0
building_fire_rates[19116]  0.069  0.063   0.000    0.148        0.0      0.0
building_fire_rates[19406]  0.054  0.005   0.046    0.062        0.0      0.0
building_fire_rates[19428]  0.036  0.004   0.030    0.043        0.0      0.0
building_fire_rates[19474]  0.109  0.076   0.007    0.208        0.0      0.0

                            ess_bulk  ess_tail  r_hat
mean_building_fire_rate      21053.0   16788.0    1.0
building_fire_rates[18054]   28756.0   14148.0    1.0
building_fire_rates[18103]   16600.0    9168.0    1.0
building_fire_rates[19010]   29696.0   14008.0    1.0
building_fire_rates[19025]   29505.0   13932.0    1.0
building_fire_rates[19040]   26603.0   14977.0    1.0
building_fire_rates[19066]   30356.0   14133.0    1.0
building_fire_rates[19116]   16213.0    8544.0    1.0
building_fire_rates[19406]   34822.0   15395.0    1.0
```

```
building_fire_rates[19428]    29828.0    15312.0     1.0
building_fire_rates[19474]    22702.0    12438.0     1.0
```

[ ]:
```
# plot posterior predictive for building fire rate
az.plot_ppc(trace)
```

```
/Users/ravj/opt/anaconda3/envs/pymc_env/lib/python3.11/site-
packages/arviz/plots/ppcplot.py:267: FutureWarning: The return type of
`Dataset.dims` will be changed to return a set of dimension names in future, in
order to be more consistent with `DataArray.dims`. To access a mapping from
dimension names to lengths, please use `Dataset.sizes`.
  flatten_pp = list(predictive_dataset.dims.keys())
/Users/ravj/opt/anaconda3/envs/pymc_env/lib/python3.11/site-
packages/arviz/plots/ppcplot.py:271: FutureWarning: The return type of
`Dataset.dims` will be changed to return a set of dimension names in future, in
order to be more consistent with `DataArray.dims`. To access a mapping from
dimension names to lengths, please use `Dataset.sizes`.
  flatten = list(observed_data.dims.keys())
```

[ ]:
```
<Axes: xlabel='building_fires'>
```



The 5K samples show that the rate of burning buildings varies between buildings. For the zip codees 18103 and 19116, the rate is zero for the observed data, and therefore the HDI includes the 0 value for the rate andalso are the ones with the longest HDI.

4.2 Q3.B: Individual Estimates

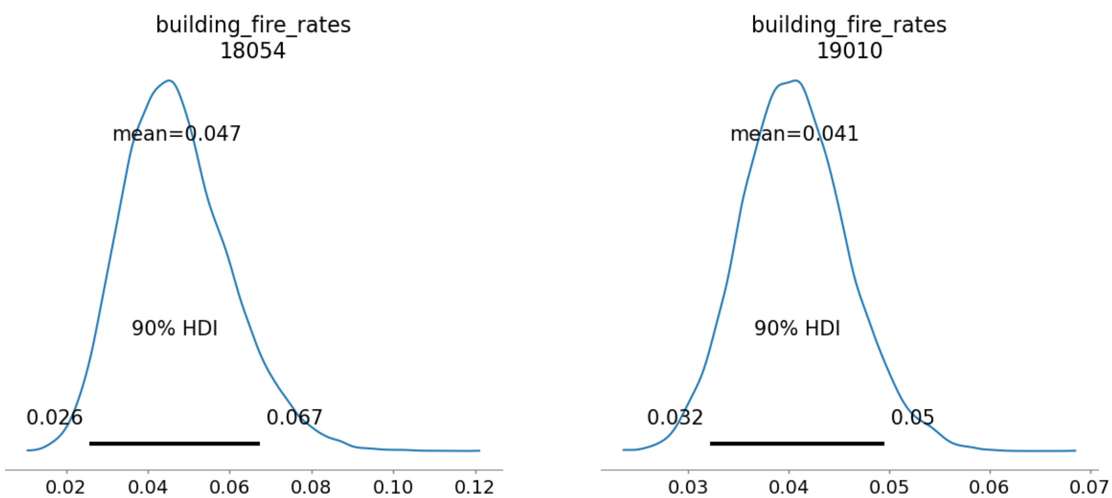
The individual estimates of the building rates for zip codes 18054 and 19010 are 12/266 and 59/1470, respectively. Contrast these estimates with the posterior means of the rates λ_1 and λ_3.

[ ]:
```
# summary of the trace
az.summary(trace,  var_names=["building_fire_rates"], coords={'zip_code':␣
 ↪[18054, 19010]}, hdi_prob=0.9)
```

[ ]:
```
                              mean     sd  hdi_5%  hdi_95%  mcse_mean  mcse_sd  \
building_fire_rates[18054]  0.047  0.013   0.026    0.067        0.0      0.0
building_fire_rates[19010]  0.041  0.005   0.032    0.050        0.0      0.0

                            ess_bulk  ess_tail  r_hat
building_fire_rates[18054]   26963.0   13453.0    1.0
building_fire_rates[19010]   29509.0   14736.0    1.0
```

[ ]:
```
# plot posterior distribution for coefficients only for the mu with obs_id =␣
 ↪Murray
az.plot_posterior(trace, var_names=["building_fire_rates"], coords={'zip_code':␣
 ↪[18054, 19010]}, hdi_prob=0.9)
```

[ ]:
```
array([<Axes: title={'center': 'building_fire_rates\n18054'}>,
       <Axes: title={'center': 'building_fire_rates\n19010'}>],
      dtype=object)
```



- The estimate rate for the zip code 18054 is 0.045 and the infered mean rate is 0.047.

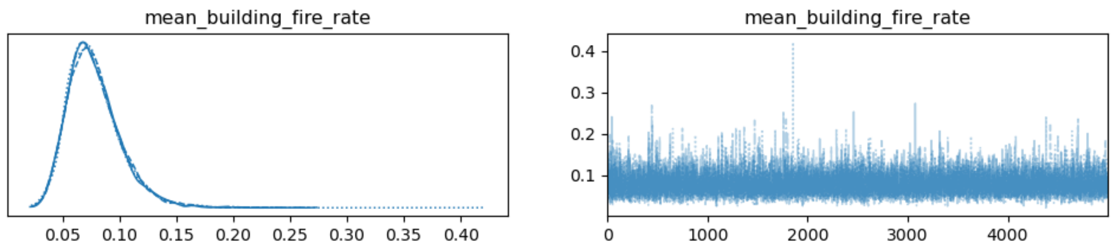
- The estimate rate for the zip code 19010 is 0.040 and the infered mean rate is 0.041.

For both cases, the estimated rate (frequency of building fires) is very close to the inferred mean rate.

4.3 Q3.C: Mean Building Fire across all zip codes

The parameter $\mu = \alpha/\beta$ represents the mean building fire rates across zip codes. Construct a density estimate of the posterior distribution of μ.

[ ]:
```
# posterior predictive check
az.plot_trace(trace, var_names=["mean_building_fire_rate"])
plt.show()
```



The average building fire rate is of 0.078 fires per call. The 90% HDI is [0.042, 0.114]. Also the simulations show that this rate has a very heavy tail, with a long tail to the right.

4.4 Q3.D: Prediction for Zip Code 19066

Suppose that the county has 50 fire calls to the zip code 19066. Use the simulated predictive distribution to construct a 90% predictive interval for the number of building fires.

[ ]:
```
# predict new data
with model:
    # new data
    pm.set_data(
        new_data={
            "fire_calls": pd.Series(50),
            "zip_code_idx": pd.Series(
                np.where(df_fire_calls["zip_code"] == 19066)[0]
                )
            },
        coords={
            "obs_id": ['19066']
            }
        )

    # sample
    posterior_pred = pm.sample_posterior_predictive(
```

```
        trace,
        var_names=["building_fires"],
        predictions=True,
        random_seed=42
    )
```

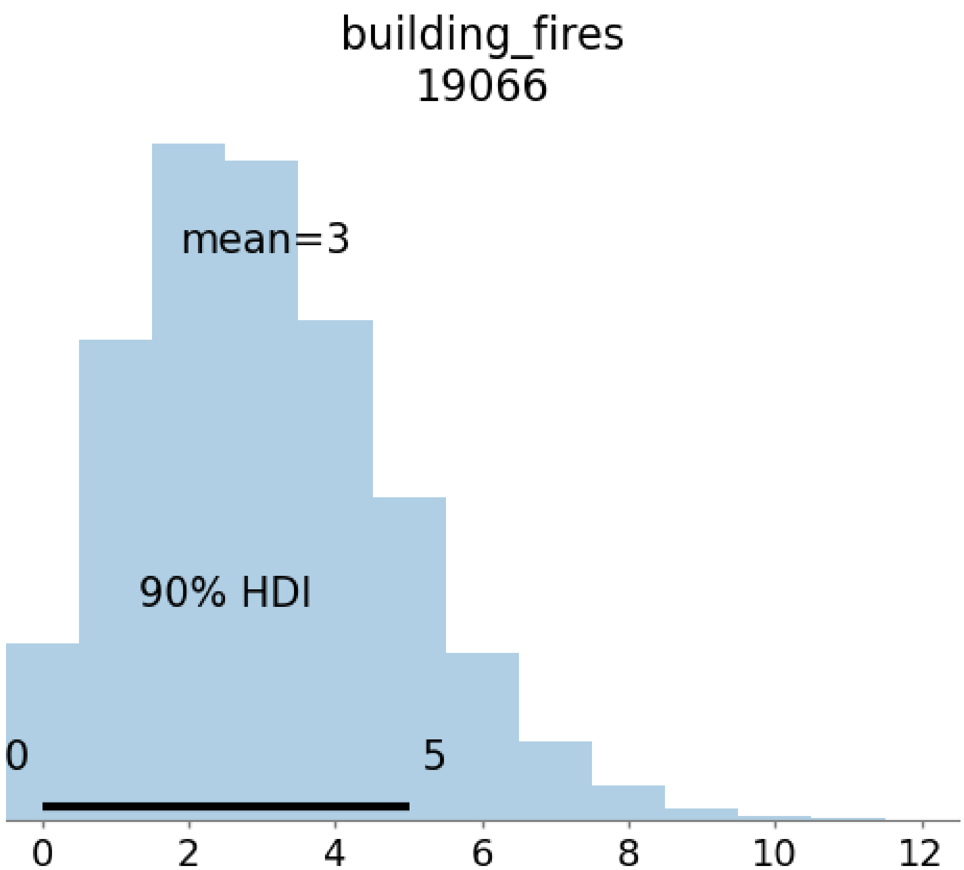
```
Sampling: [building_fires]
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

[ ]:
```
# plot the predictions
az.plot_posterior(posterior_pred, group="predictions", hdi_prob=0.90)
```

[ ]:
```
<Axes: title={'center': 'building_fires\n19066'}>
```



[ ]:
```
# show table of the predictions for the posertior pred
az.summary(
    posterior_pred, group="predictions", hdi_prob=0.90
```

```
)
```

```
[ ]:                         mean     sd  hdi_5%  hdi_95%  mcse_mean  mcse_sd  \
building_fires[19066]  3.031  1.833     0.0      5.0      0.013    0.009

                       ess_bulk  ess_tail  r_hat
building_fires[19066]   21285.0   20005.0    1.0
```

Based on the simulations for the posterior predictive of building fires for 50 fire calls for the zip code `19066` show that is highly probable to see between 0 to 5 building fires with a 90% credible interval with an average of 3 building fires.