认知建模基础作业4

In [1]:
```
import numpy as np
import pymc as pm
import pytensor.tensor as pt
from scipy.stats import gaussian_kde
import matplotlib.pyplot as plt
import arviz as az
import pandas as pd
```

1 Bayesian Factor

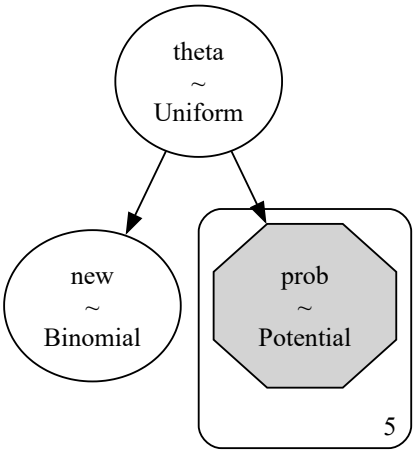
在全部题目均为判断题的考试中，共有100道题目，答对0-59道题目为 `不及格(F)`，答对60-84道题目为 `及格(B)`，答对85-100道题目为 `优秀(A)`，同学 X 前三次考试均为 `及格(B)`，第四、五次考试成绩为 `不及格(F)`，假设同学 X 的能力 θ（即做对每一道题的概率，不同题目是否做对是独立的，不同考试之间不发生改变）的先验是 $U(0,1)$。

1. 请你将成绩视作删失数据（censored data 可参考课上Cha Sa soon的例子），用MCMC采样（本题为保证结果稳定请采样不少于10000次）的方法，报告同学 X 能力的后验分布，并基于后验分布，给出如果该同学额外参加一次考试，成绩为 `优秀(A)` `及格(B)` `不及格(F)` 概率的后验预测分布（10分）。

In [39]:
```
result = np.array([1, 1, 1, 0, 0])  # 1表示B，0表示F
with pm.Model() as ExamX:
    theta = pm.Uniform('theta', lower = 0, upper = 1)
    pB = pt.log(pt.exp(pm.Binomial.logcdf(84, 100, theta)) - pt.exp(pm.Binomial.logcdf(59, 100, theta)))
    pC = pm.Binomial.logcdf(59, 100, theta)
    prob = pm.Potential('prob', pt.switch(result, pB, pC))
    new = pm.Binomial('new', n = 100, p = theta)  # 额外参加一次考试
    trace_exam = pm.sample(10000, progressbar=False, random_seed=42)
pm.model_to_graphviz(ExamX)
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>NUTS: [theta]
>Metropolis: [new]
Sampling 4 chains for 1_000 tune and 10_000 draw iterations (4_000 + 40_000 draws total) took 28 seconds.
```

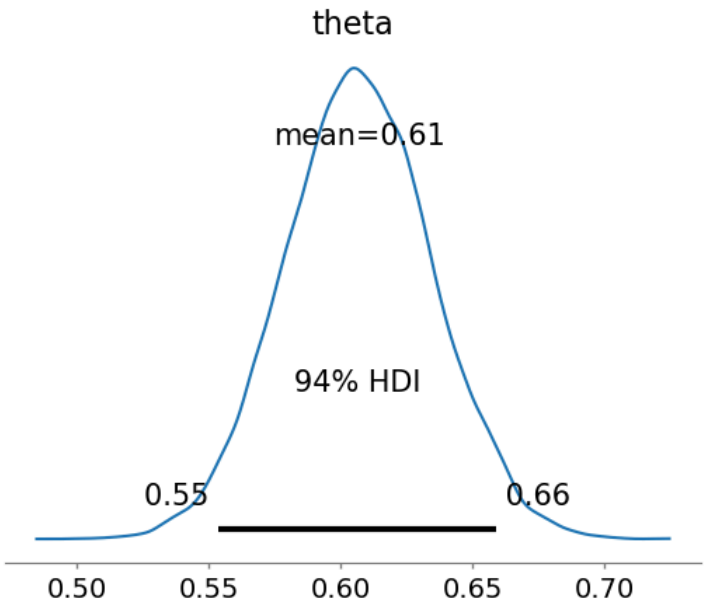
Out[39]:



In [40]:
```
# 正确率theta的后验分布
az.plot_posterior(trace_exam, var_names=['theta'])
az.summary(trace_exam)
```

Out[40]:

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
new	60.700	5.574	50.000	70.000	0.073	0.051	5907.0	8584.0	1.0
theta	0.607	0.028	0.554	0.659	0.000	0.000	9104.0	16937.0	1.0



In [41]:
```
# 再参加一次考试的后验预测分布
## 使用不同seed的结果不太一样
new = trace_exam.posterior['new'].values.flatten()  # 常见的读取trace中的数据为一列array
print(f'F: {np.mean(new <60)*100:.2f}%, B: {(np.mean(new < 85) - np.mean(new < 60))*100:.2f}%, A: {np.mean(new > 84)*100:.
```

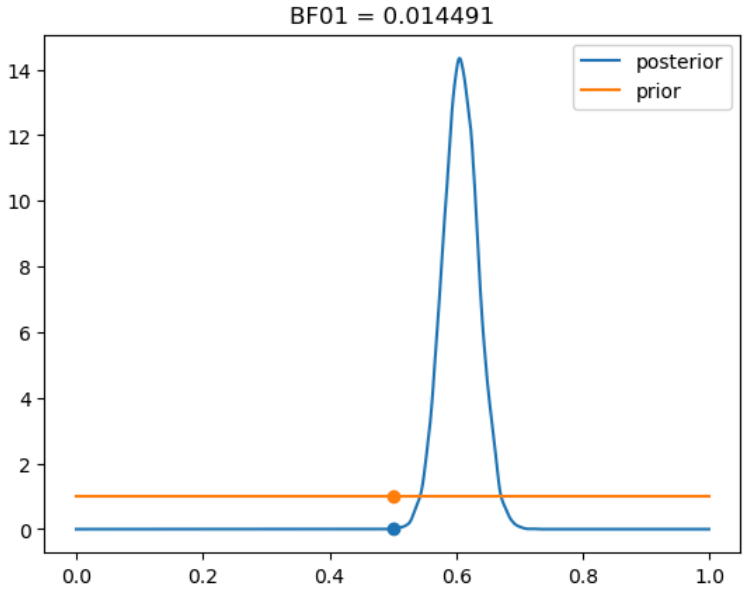
```
F: 40.79%, B: 59.20%, A: 0.01%
```

2. 有两个模型可以解释同学 X 的行为，$\mathcal{H}_0: \theta = 0.5$ 和 $\mathcal{H}_1: \theta \sim U(0,1)$，对于这样的嵌套模型，我们可以用Savage-Dickey方法估计贝叶斯因子，具体做法是，在 $\mathcal{H}_1$的假设下进行MCMC采样后，使用参数 θ 的先验分布和后验分布计算：

$$BF_{01} = \frac{p(\mathcal{D}|\mathcal{H}_0)}{p(\mathcal{D}|\mathcal{H}_1)} = \frac{p(\theta = 0.5|\mathcal{D}, \mathcal{H}_1)}{p(\theta = 0.5|\mathcal{H}_1)}$$

其中先验分布（分母）可以直接获得，而后验分布（分子）可以对采样样本进行平滑（如直方图或者高斯核估计等）得到，请你自行查阅资料，解释算得贝叶斯因子对两个假设的支持程度（10分）。

In [42]:
```
# 取出theta的后验分布
theta = trace_exam.posterior['theta'].values.flatten()

# 使用Gaussian KDE估计theta的后验分布
kde_theta = gaussian_kde(theta)
x = np.arange(0, 1, 0.001)
plt.plot(x, kde_theta(x), label = 'posterior')

# 先验分布是已知的均匀分布
plt.plot(x, [1]*len(x), label = 'prior')

# 计算贝叶斯因子
plt.scatter(0.5, kde_theta(0.5))  # 分子
plt.scatter(0.5, 1)  # 分母
plt.title(f'BF01 = {kde_theta(0.5).item():.6f}')  #BF
plt.legend()
plt.show()

# BF01的证据非常强的支持H1
# BF01 < 1/100, 极强的证据支持H1
# 1/100 < BF01 < 1/30, 非常强的证据支持H1
# 1/30 < BF01 < 1/10, 强的证据支持H1
# 1/10 < BF01 < 1/3, 中等的证据支持H1
# 1/3 < BF01 < 1, 较弱的证据支持H1
# 1 < BF01 < 3, 较弱的证据支持H0
# 3 < BF01 < 10, 中等的证据支持H0
# 10 < BF01 < 30, 强的证据支持H0
# 30 < BF01 < 100, 非常强的证据支持H0
# BF01 > 100, 极强的证据支持H0
```

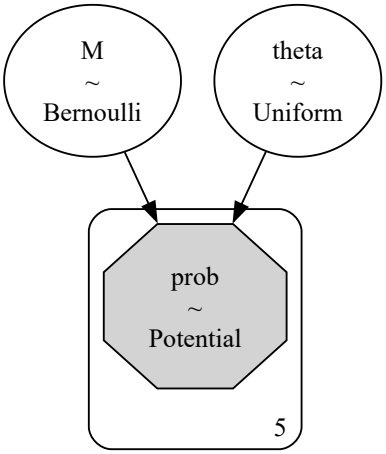


补充：事实上还有一个纯粹贝叶斯的方法实现模型比较，假设我们有一个最高层的因变量M表示选择模型，先验分布是 Bernoulli(0.5)，如果取1则按照H1生成模型 ($\theta \sim U(0,1)$), 如果取0则按照H0生成模型 ($\theta = 0.5$)，最后我们对M的后验均值 $p(M1)$ 取odds的倒数 $\frac{1-p(M1)}{p(M1)}$ 就是BF01的结果，取样误差还是有点大的

In [112…
```
result = np.array([1, 1, 1, 0, 0])  # 1表示B，0表示F
with pm.Model() as ExamComp:
    M = pm.Bernoulli('M', p = 0.5)
    theta1 = pm.Uniform('theta', lower = 0, upper = 1)
    theta0 = 0.5
    theta = pm.math.switch(M, theta1, theta0)

    pB = pt.log(pt.exp(pm.Binomial.logcdf(84, 100, theta)) - pt.exp(pm.Binomial.logcdf(59, 100, theta)))
    pC = pm.Binomial.logcdf(59, 100, theta)
    prob = pm.Potential('prob', pt.switch(result, pB, pC))

    trace_exam = pm.sample(10000, progressbar=False, random_seed=42)
pm.model_to_graphviz(ExamComp)
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>BinaryGibbsMetropolis: [M]
>NUTS: [theta]
Sampling 4 chains for 1_000 tune and 10_000 draw iterations (4_000 + 40_000 draws total) took 29 seconds.
```

Out[112…



In [113…
```
M_sum = az.summary(trace_exam, var_names=['M'])
pm1 = M_sum["mean"].values[0]
print(f'p(M1) = {pm1:.4f}, BF01 = {(1-pm1)/pm1:.4f}')
```

```
p(M1) = 0.9920, BF01 = 0.0081
```

3. 对于先验的选择，$\mathcal{H}_2 : \theta \sim U(0.5, 1)$ 可能更加合理，请你预测使用在这一先验下贝叶斯因子 BF_{02} 相比 BF_{01} 如何改变（你可以选择修改模型后重新采样的结果，或者使用 $\mathcal{H}_1$的结果通过数值计算估计新的贝叶斯因子的值，10分）

首先，先验分布 $\theta \sim U(0.5, 1)$ 在 $\theta = 0.5$ 处的pdf是 2，问题在于如何从后验样本获得后验pdf估计

- 思路1：重新构建模型，修改theta的先验后，sample得到新样本后重新进行KDE估计

- 思路2：不改变模型，但是只保留theta trace中大于等0.5的部分，重新进行KDE估计
- 思路3：不改变模型并使用原本的KDE估计，对大于等于0.5的部分重新进行归一化

三种思路的结果还是有比窘大的差异的，个人认为因为0.5在分布的边缘，这样新trace的KDE估计是不稳定的，而思路3在数值稳定性上表现得更好

In [ ]:
```
result = np.array([1, 1, 1, 0, 0])  # 1表示B，0表示F
with pm.Model() as ExamX2:
    theta = pm.Uniform('theta', lower = 0.5, upper = 1)
    pB = pt.log(pt.exp(pm.Binomial.logcdf(84, 100, theta)) - pt.exp(pm.Binomial.logcdf(59, 100, theta)))
    pC = pm.Binomial.logcdf(59, 100, theta)
    prob = pm.Potential('prob', pt.switch(result, pB, pC))
    trace_exam2 = pm.sample(10000, progressbar=False, random_seed=42)
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>NUTS: [theta]
>Metropolis: [new]
CompoundStep
>NUTS: [theta]
>Metropolis: [new]
Sampling 4 chains for 1_000 tune and 10_000 draw iterations (4_000 + 40_000 draws total) took 31 seconds.
```

In [46]:
```
# 思路1
theta2 = trace_exam2.posterior['theta'].values.flatten()
kde_theta1 = gaussian_kde(theta2)
kde_theta1(0.5)/2  # 可以尝试不同的random_seed，结果会有差异
```

Out[46]:
```
array([7.72581283e-05])
```

In [49]:
```
# 思路2
theta_half = theta[theta >= 0.5]
kde_theta2 = gaussian_kde(theta_half)
kde_theta2(0.5)/2
```

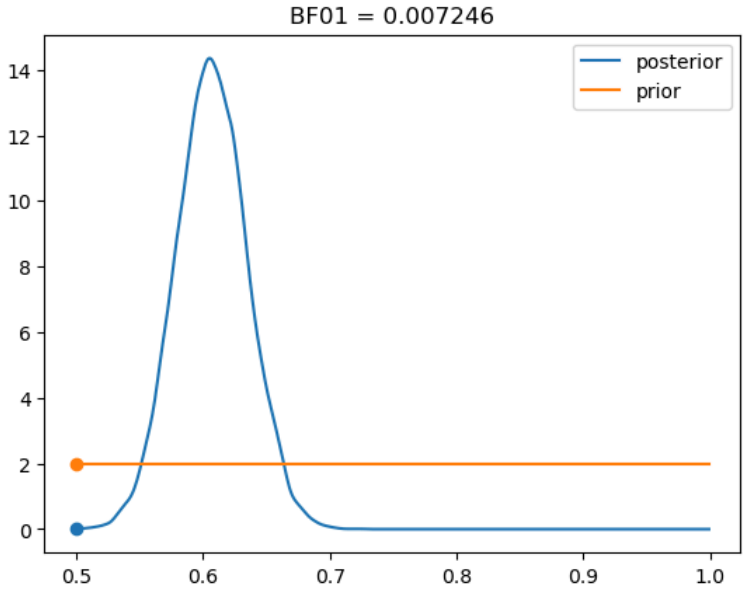
Out[49]:
```
array([0.00503372])
```

In [50]:
```
# 思路3
kde_theta(0.5) / np.mean(theta >= 0.5) /2
```

Out[50]:
```
array([0.00724632])
```

In [51]:
```
theta = trace_exam.posterior['theta'].values.flatten()
kde_theta = gaussian_kde(theta)
x = np.arange(0.5, 1, 0.001)
plt.plot(x, kde_theta(x)/np.mean(theta >=0.5), label = 'posterior')
plt.scatter(0.5, kde_theta(0.5)/np.mean(theta >=0.5))
plt.plot(x, [2]*len(x), label = 'prior')
plt.scatter(0.5, 2)
plt.title(f'BF01 = {kde_theta(0.5).item()/np.mean(theta >=0.5)/2:.6f}')
plt.legend() #BF < .01, 非常强的证据支持H1
```

Out[51]:
```
<matplotlib.legend.Legend at 0x20b55b62030>
```



2 Bayesian Model Selection

在本题目中，我们希望通过一个现实情形的类比，帮助大家理解贝叶斯模型选择的概念。某中心的老师希望探究 A 同学的课堂参与情况，有两个模型进行解释：

- $\mathcal{M}_1$：该同学是**出席者**；
- $\mathcal{M}_0$：该同学是**缺席者**。

检验模型的数据为 A 同学在16周课程中的签到情况 $\mathcal{D} = \{\mathcal{D}_i \in \{0,1\}\}_{i=1}^{16}$：`1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0`，其中 `1` 表示签到成功，`0` 表示签到失败。我们对模型证据作出如下假定：出席者签到成功的概率为99%，而缺席者签到成功的概率为40%，不同观测之间独立。即模型证据函数为：

$$p(\mathcal{D}_i|\mathcal{M}_1) = \begin{cases} 0.99, \text{ if } \mathcal{D}_i = 1 \\ 0.01, \text{ if } \mathcal{D}_i = 0 \end{cases}$$
$$p(\mathcal{D}_i|\mathcal{M}_0) = \begin{cases} 0.4, \text{ if } \mathcal{D}_i = 1 \\ 0.6, \text{ if } \mathcal{D}_i = 0 \end{cases}$$

1. 固定效应假说：该模型假设 A 同学要么是**出席者**（$\mathcal{M}_1$）要么是**缺席者**（$\mathcal{M}_0$），在整个学期不会发生改变。请分别计算似然函数 $p(\mathcal{D}|\mathcal{M}_1)$ 和 $p(\mathcal{D}|\mathcal{M}_0)$，并报告按照最大似然法选择的可以解释 A 同学行为的模型（5分）；

In [61]:
```
data = np.array([1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0])
# data = np.array([0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0])  # 两者不一致
counts = np.bincount(data)
p_D_M1 = 0.01 ** counts[0] * 0.99 ** counts[1]
p_D_M0 = 0.6 ** counts[0] * 0.4 ** counts[1]
p_D_M1, p_D_M0 # p_D_M1 < p_D_M0, MLE结果认为A是缺席者(M0)
print(f'log likilihood of M0: {np.log(p_D_M0):.4f}, log likilihood of M1: {np.log(p_D_M1):.4f}, 支持模型M0')
```

```
log likilihood of M0: -13.4443, log likilihood of M1: -13.9462, 支持模型M0
```

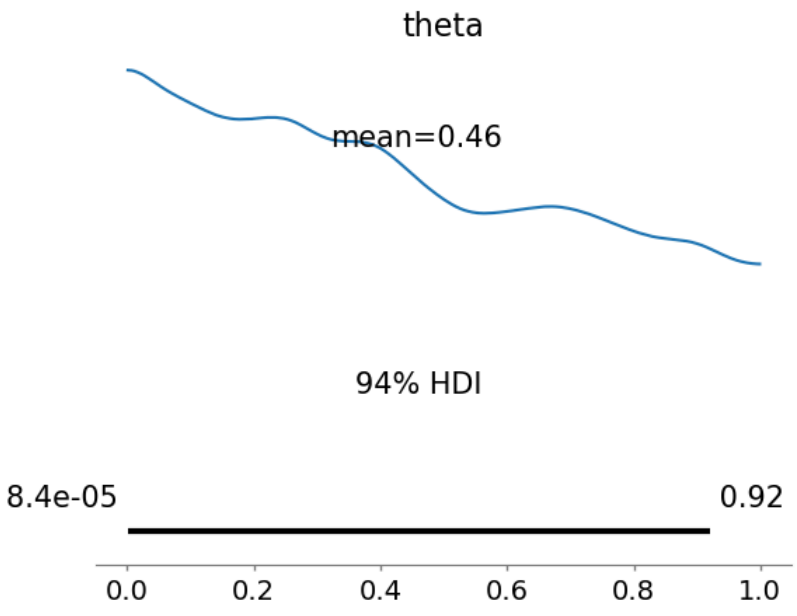
2. 在固定效应假说基础上，我们假定 A 同学是出席者的概率为 $p(\mathcal{M}_1) = \theta$，而 θ 的先验概率为 $[0,1]$ 均匀分布，请画出图模型，并通过**MCMC**采样报告 θ 的后验分布和后验期望（5分）；

In [ ]:
```
with pm.Model() as AbsentFixEffect:
    theta = pm.Uniform('theta', lower = 0, upper = 1)
    identity = pm.Bernoulli('identity', p = theta)
    p1 = pt.switch(identity, 0.99, 0.4)
    sign = pm.Bernoulli('sign', p = p1, observed = data)
    trace_absentfix = pm.sample(2000, progressbar=False, random_seed=42)
pm.model_to_graphviz(AbsentFixEffect)
az.plot_posterior(trace_absentfix, var_names=['theta'])
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>NUTS: [theta]
>BinaryGibbsMetropolis: [identity]
Sampling 4 chains for 1_000 tune and 2_000 draw iterations (4_000 + 8_000 draws total) took 32 seconds.
```

Out[ ]:

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
theta	0.456	0.287	0.0	0.92	0.006	0.004	2262.0	3366.0	1.0



In [72]:
```
theta_sum = az.summary(trace_absentfix, var_names=['theta'], hdi_prob=0.95)
print(f'theta的后验均值为{theta_sum["mean"].values[0]:.3f}, 95%可信区间为[{theta_sum["hdi_2.5%"].values[0]:.3f},{theta_sum
```

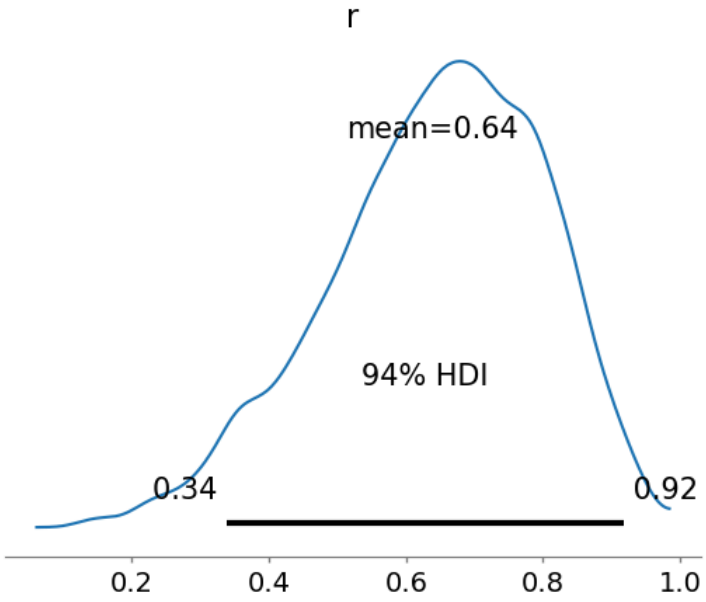
theta的后验均值为0.456, 95%可信区间为[0.000,0.933]

3. 随机效应假说：该模型假设 A 同学每节课前都会随机"选择"成为**出席者**（$\mathcal{M}_1$）或者**缺席者**（$\mathcal{M}_0$）。具体来讲， A 同学成为出席者的先验概率是 $r \sim U(0,1)$，隐变量 $z_i \sim Bernoulli(r)$ 表示每节课的状态（1对应出席者，0对应缺席者）。画出图模型，并通过MCMC采样报告 r 的后验分布和后验期望，同时报告哪那些周 A 同学更可能缺席了课程（5分）；

In [ ]:
```
with pm.Model() as AbsentRandomEffect:
    r = pm.Uniform('r', lower = 0, upper = 1)
    z = pm.Bernoulli('z', p = r, shape = len(data))
    p1 = pt.switch(z, 0.99, 0.4)
    # p0 = pt.switch(identity, 0.1, 0.8)
    sign = pm.Bernoulli('sign', p = p1, observed = data)
    trace_absentrand = pm.sample(2000, progressbar=False, random_seed=42)
pm.model_to_graphviz(AbsentRandomEffect)
az.plot_posterior(trace_absentrand, var_names=['r'], round_to=3)
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>NUTS: [r]
>BinaryGibbsMetropolis: [z]
Sampling 4 chains for 1_000 tune and 2_000 draw iterations (4_000 + 8_000 draws total) took 37 seconds.
```

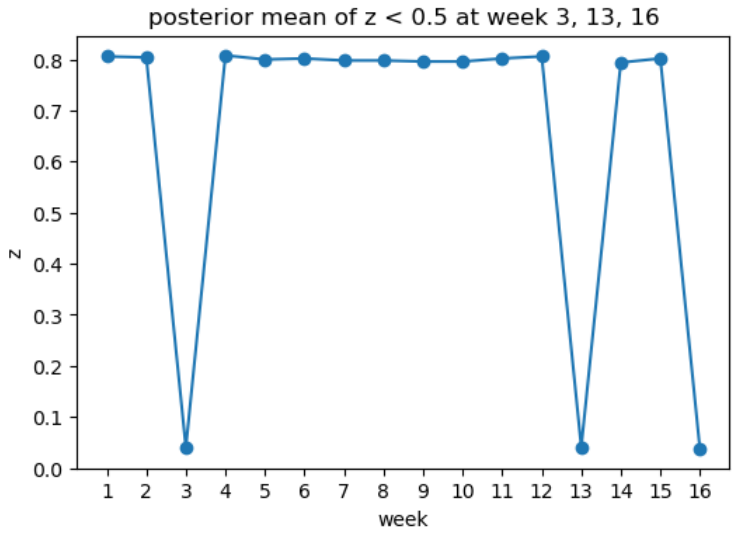
Out[ ]: <Axes: title={'center': 'r'}>



In [75]:
```
r_sum = az.summary(trace_absentrand, var_names=['r'], hdi_prob=0.95) #多数课程A还是出席的
print(f'r的后验均值为{r_sum["mean"].values[0]:.3f}, 95%可信区间为[{r_sum["hdi_2.5%"].values[0]:.3f},{r_sum["hdi_97.5%"].va
```

r的后验均值为0.639, 95%可信区间为[0.329,0.930]

In [98]:
```
z_sum = az.summary(trace_absentrand, var_names=['z']) #这是很明显的，3 13 16三周
fig, ax = plt.subplots(figsize=(6, 4))
ax.plot(np.arange(16), z_sum['mean'], '-o')
ax.set(xlabel='week', ylabel='z', title='posterior mean of z < 0.5 at week 3, 13, 16',
       xticks=np.arange(16), xticklabels=np.arange(1, 17))
plt.show()  # 这是非常符合直觉的
```



4. 在随机效应模型下，除了用 r 的后验期望表示 A 同学每节课前成为**出席者**的平均概率，我们还可以近似计算出席者模型($\mathcal{M}_1$)的胜出概率（probability of exceedance)，其定义是：

$$\phi_1 = p(r > 1 - r|\mathcal{D}) = \int_{0.5}^{1} p(r|\mathcal{D})dr$$

我们可以用**MCMC**采样结果中 $r > 0.5$ 的比例进行近似，请估计 $\mathcal{M}_1$ 的胜出概率（5分）

In [102…
```
r_trace = trace_absentrand.posterior['r'].values.flatten()
print(f'probability of exceedance of M1:{(r_trace > 0.5).mean():.3f}')
```

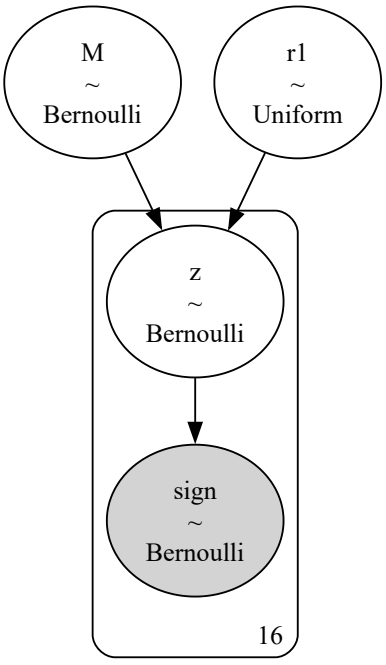
```
probability of exceedance of M1:0.807
```

补充，事实上0.807的胜出概率是一个很不稳定的结果，不能得到确切的证据，我们一般会使用受保护的胜出概率（protected probability of exceedance），直观上讲一下图模型的实现是，我们比较两个模型，H1是 $r \sim U(0,1)$, H0是 $r = 0.5$，之后用第一大题的方法，我们还是可以计算BF01，如果H1没有优于H0，那么我们算出来高于0.5的胜出概率就是没有统计学意义的。具体受保护的胜出概率的计算就是$PXP_1 = \phi_1 p(H1) + 0.5p(H0)$，如果 $p(H1) >> p(H0)$ 那么受保护的胜出概率还是 ϕ_1, 反之如果 $p(H1) << p(H0)$ 那么受保护的胜出概率会被校正到0.5。

In [126…
```
with pm.Model() as AbsentRandomEffect:
    M = pm.Bernoulli('M', p = 0.5)
    r1 = pm.Uniform('r1', lower = 0, upper = 1)
    r0 = 0.5
    r = pm.math.switch(M, r1, r0)

    z = pm.Bernoulli('z', p = r, shape = len(data))
    p1 = pt.switch(z, 0.99, 0.4)
    # p0 = pt.switch(identity, 0.1, 0.8)
    sign = pm.Bernoulli('sign', p = p1, observed = data)
    trace_absentrand = pm.sample(2000, progressbar=False, random_seed=42)
pm.model_to_graphviz(AbsentRandomEffect)
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>BinaryGibbsMetropolis: [M, z]
>NUTS: [r1]
Sampling 4 chains for 1_000 tune and 2_000 draw iterations (4_000 + 8_000 draws total) took 17 seconds.
```

Out[126…



In [135…
```
az.summary(trace_absentrand, var_names=['M', 'r1'])
m_trace = trace_absentrand.posterior['M'].values.flatten()
r1_trace = trace_absentrand.posterior['r1'].values.flatten()
phi1 = (r1_trace[m_trace == 1] > 0.5).mean()
PXP1 = (m_trace == 1).mean() * phi1 + (m_trace == 0).mean() * 0.5
print(f'phi = {phi1:.3f}, PXP = {PXP1:.3f}')
```

```
phi = 0.800, PXP = 0.624
```

5. 下面我们考虑现实的模型比较问题，假设现在有 $\mathcal{M}_1$ 和 $\mathcal{M}_0$ 两个模型，分别对16名被试的数据进行逐被试的拟合得到两组AIC指标 `AIC1` 和 `AIC0`，数据存储在 `aic.csv` 中，我们可以用AIC指标对模型证据进行近似：

$$\log p(\mathcal{D}_i|M_0) = -\frac{1}{2}AIC_0[i])$$
$$\log p(\mathcal{D}_i|M_1) = -\frac{1}{2}AIC_1[i])$$

除此之外均与2-4问中图模型结构一致，请你仿照2-4问的做法，（1）在固定效应假说下采样模型得到 θ 的后验分布和后验期望；（2）在随机效应假说下采样模型得到 r 的后验分布和后验期望，并报告模型 M_1 的胜出概率。（你可能需要用到pm.Potential函数，它接收的值应该是对数概率，10分）

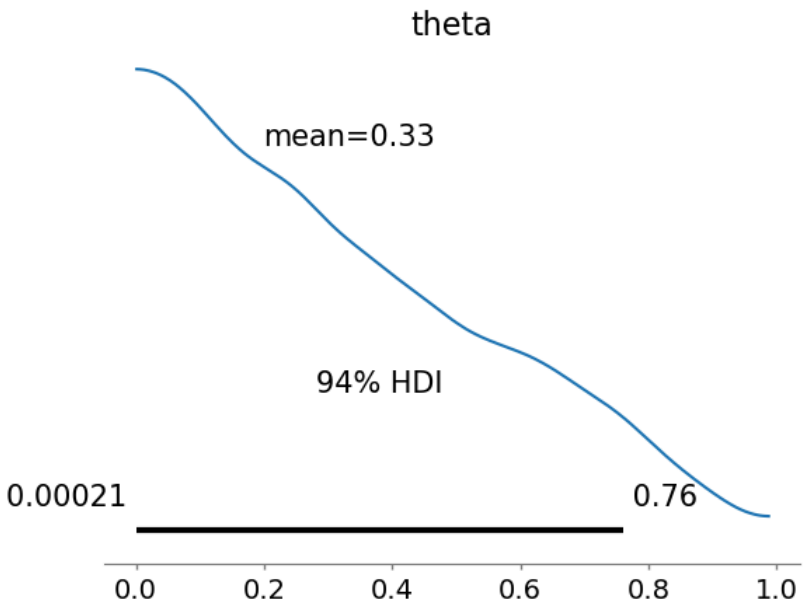
In [142…
```
data = pd.read_csv('aic.csv')
AIC0 = np.array(data.AIC0)
AIC1 = np.array(data.AIC1)
```

In [146…
```
with pm.Model() as AICFixEffect:
    theta = pm.Uniform('theta', lower = 0, upper = 1)
    z = pm.Bernoulli('z', p = theta)
    AIC = AIC1 * z + (1 - z) * AIC0
    evidence = pm.Potential('evidence', -0.5 * AIC)
    trace_AICfix = pm.sample(2000, progressbar=False, random_seed=42)
pm.model_to_graphviz(AICFixEffect)
az.plot_posterior(trace_AICfix, var_names=['theta'])
az.summary(trace_AICfix, var_names=['theta'])
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>NUTS: [theta]
>BinaryGibbsMetropolis: [z]
Sampling 4 chains for 1_000 tune and 2_000 draw iterations (4_000 + 8_000 draws total) took 30 seconds.
c:\Users\asus\miniconda3\envs\pymc\Lib\site-packages\arviz\stats\diagnostics.py:596: RuntimeWarning: invalid value encounte
red in scalar divide
  (between_chain_variance / within_chain_variance + num_samples - 1) / (num_samples)
```

Out[146…

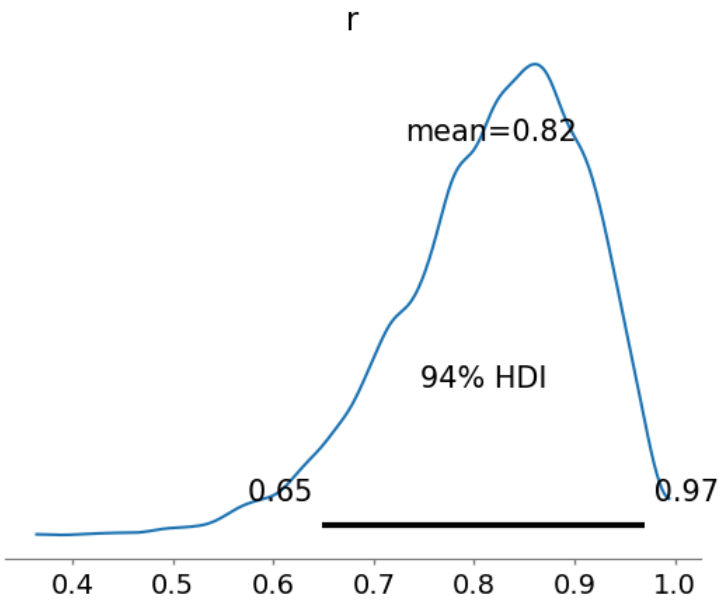
	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
theta	0.333	0.24	0.0	0.76	0.004	0.003	3041.0	3453.0	1.0



In [ ]:
```
with pm.Model() as AICRandomEffect:
    r = pm.Uniform('r', lower = 0, upper = 1)
    identity = pm.Bernoulli('identity', p = r, shape = len(AIC1))
    AIC = AIC1 * identity + (1 - identity) * AIC0
    evidence = pm.Potential('evidence', -0.5 * AIC)
    trace_AICrand = pm.sample(2000, progressbar=False, random_seed=42)
pm.model_to_graphviz(AICRandomEffect)
az.plot_posterior(trace_AICrand, var_names=['r'])
az.summary(trace_AICrand, var_names=['r'])
```

Out[ ]:

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
r	0.817	0.092	0.648	0.971	0.002	0.001	2612.0	3213.0	1.0



In [151…
```
pexc = (trace_AICrand.posterior['r'].values > 0.5).mean()
print(f'probability of exceedance of M1:{pexc:.3f}')
```

```
probability of exceedance of M1:0.997
```

6. 结合上面两个具体的例子，解释在固定效应和随机效应两种假设下，进行模型比较时会得到不同结果的可能原因，你认为在模型比较中更应该采用哪种思路？（5分）

来自一些同学们的答案，供参考（等批改完作业后再补充一份更完整的）：

- 可能出现不同结果，是因为**固定效应模型**假设被试出席的概率/群体中出席者的比例固定，而**随机效应模型**假设被试行为存在随机性/群体中各个个体出席的平均概率不固定，二者的假设在根本上不同；我认为**随机效应模型**更加合理，因为人类行为具有随机性，完全固定的"出席/缺席"分类不太符合常识。

Case Study

3.Hierarchical Latent Mixture Model

在`homework3`中，我们引入了`Coffee or Beer`这样一道题目，题目内容如下：

> 在一项食品研究中，研究人员想要区分两类专业人士的口味偏好：
>
> - **咖啡师**：专长于咖啡的风味、烘焙程度、酸度等；
> - **酿酒师**：专长于啤酒的苦度、麦芽风味、酒精发酵工艺等。
>
> 咖啡师给咖啡评分(α_0)比啤酒分数(β_0)更高，而酿酒师给啤酒评分(α_1)比咖啡分数更高(β_1)。同时咖啡师给分极差更大($\beta_0 < \beta_1 < \alpha_1 < \alpha_0$)。在实际拟合数据时，**我们发现可能还存在第三类人：跨界专家。**他们是研究食品风味的科学家，对不同风味有自己研究，对啤酒和咖啡并无给分高低之分，但由于见多识广,评分(θ)不高不低($\beta_0 < \beta_1 < \theta < \alpha_1 < \alpha_0$)
>
> **另外，值得注意的是，第一名rater是酿酒师，第一杯饮料是咖啡。**在本题目中，**分数范围是0-100的连续值。**

上面的建模却忽略了个体差异，**比如不同咖啡师给咖啡打分均值和标准差可能也是不同的。**在这道题目中，我们利用**分层贝叶斯建模**，即考虑群体差异同时考虑个体差异，在上述假设基础上，本题增加额外假设如下：

> - **不同群体对不同饮料给分均值** μ：服从以下关系($\beta_0 < \beta_1 < \theta < \alpha_1 < \alpha_0$)，处于同一群体的不同个体对同一类别饮料给分均值相同。
> - **不同群体对饮料给分幅度均值** $\bar{\sigma}_i$：由于酒精的作用，酿酒师给分相对来说比较随意，其对于饮品给分幅度均值($\bar{\sigma}_1$)相对较大，大于等于咖啡师给分幅度均值($\bar{\sigma}_0$)和跨界专家给分均值($\bar{\sigma}_2$)，也即($\bar{\sigma}_0 \leq \bar{\sigma}_1$，$\bar{\sigma}_2 \leq \bar{\sigma}_1$)。同时 $0 < \bar{\sigma}_1 < 5$。
> - **每名个体给分幅度** σ_j：服从高斯分布，均值为个体所在群体给分幅度均值$\bar{\sigma}_i$，标准差为定值 0.1，例如对某位酿酒师：$\sigma_j \sim \mathcal{N}(\bar{\sigma}_1, 0.1^2)$，并且同一个体对不同饮料给分幅度相同（即同一个个体给咖啡或者啤酒的标准差没有差异）；
> - 每名个体 j 给不同饮料的评分均服从高斯分布 $\mathcal{N}(\mu, \sigma_j^2)$。

本题数据储存在`cb.csv`中，可参考课件中`6.2 Exam scores with individual differences`代码进行作答：

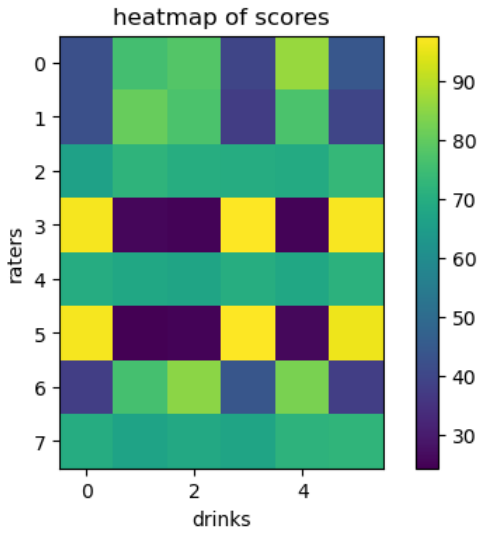
1. 请你根据本题目的假设写出模型并绘制出概率图模型(5分)

In [179…

```
# load data
data = pd.read_csv('cb.csv')
scores = np.array(data)[:, :-1]
n, drinks = scores.shape
fig, ax = plt.subplots(figsize=(6, 4))
heatmap = ax.imshow(scores)
ax.set(xlabel='drinks', ylabel='raters', title='heatmap of scores')
plt.colorbar(heatmap)
```

Out[179…

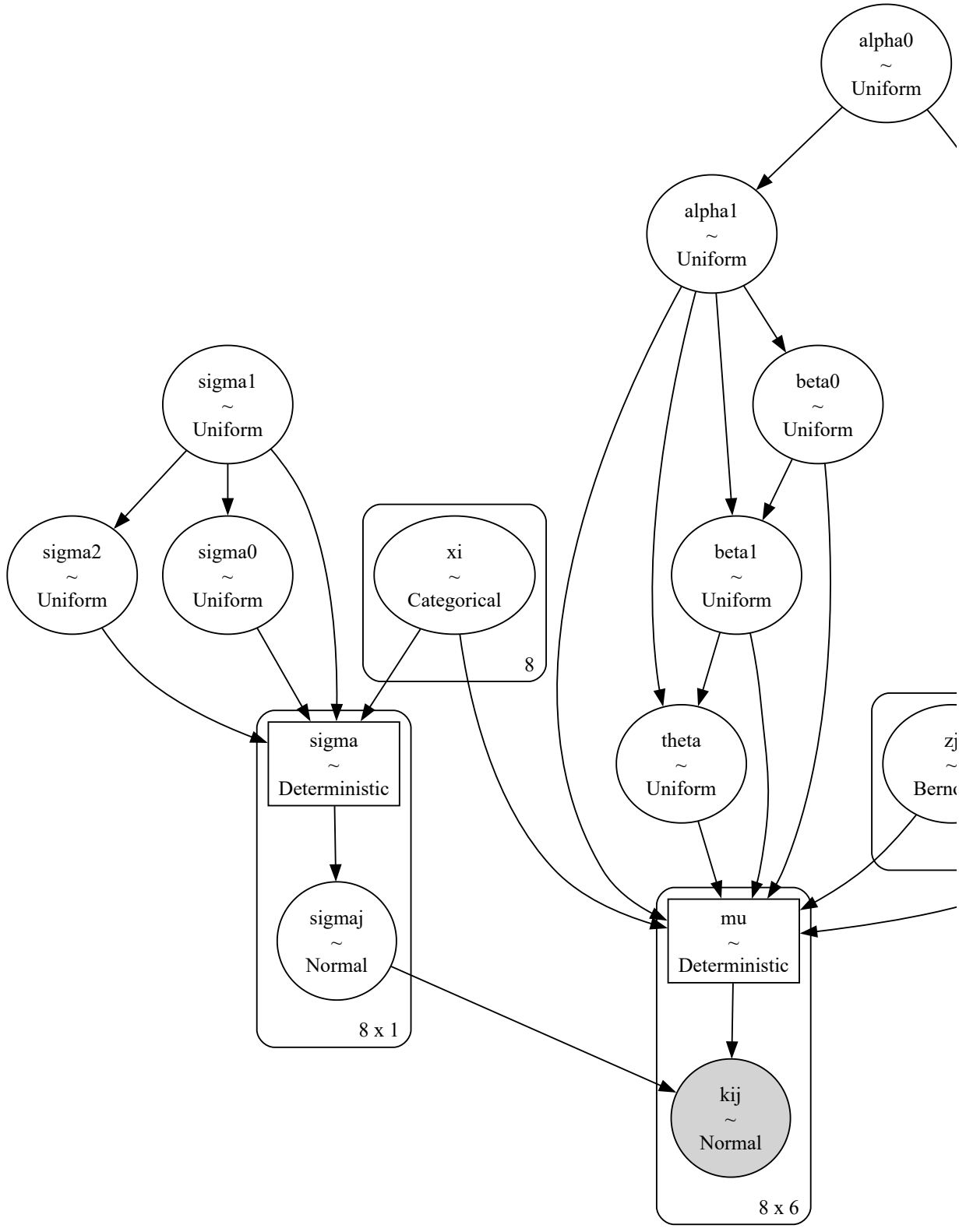
```
<matplotlib.colorbar.Colorbar at 0x20b5b2a9d00>
```



In [180…

```
# set model
with pm.Model() as model1:
    # population mu
    alpha0 = pm.Uniform('alpha0', lower=0., upper=100)
    alpha1 = pm.Uniform('alpha1', lower=0., upper=alpha0)
    beta0 = pm.Uniform('beta0', lower=0., upper=alpha1)
    beta1 = pm.Uniform('beta1', lower=beta0, upper=alpha1)
    theta = pm.Uniform('theta', lower=beta1, upper=alpha1)

    # population sigma
    sigma1 = pm.Uniform('sigma1', lower=0., upper=5)
    sigma2 = pm.Uniform('sigma2', lower=0., upper=sigma1)
    sigma0 = pm.Uniform('sigma0', lower=0., upper=sigma1)

    # 设置 xi 和 zj
    xi_probs = np.vstack([[0, 1, 0]] + [[1/3, 1/3, 1/3]] * (n - 1))
    xi_probs = pt.as_tensor_variable(xi_probs)
    xi = pm.Categorical('xi', p=xi_probs, shape=(n,))
    xi = xi[:, None]

    # 设置第一个饮品是0
    zj = pm.Bernoulli("zj", p=[0]+[0.5]*(drinks-1), shape=(1, drinks))


    # 设置参数 theta，使用嵌套的 pt.switch 进行二重判定
    mu = pm.Deterministic('mu',
        pt.switch(
        pt.eq(xi, 2),
        theta,
        pt.switch(
            pt.eq(xi, zj),
            pt.switch(pt.eq(xi, 0), alpha0, alpha1),
            pt.switch(pt.eq(xi, 0), beta0, beta1)
        )
    ))

    sigma = pm.Deterministic('sigma',
        pt.switch(
        pt.eq(xi, 2),
        sigma2,
        pt.switch(
            pt.eq(xi, 0),
            sigma0, sigma1
        )
    ))
    sigmaj = pm.Normal('sigmaj', mu=sigma, sigma=0.1)

    kij = pm.Normal('kij', mu=mu, sigma=sigmaj, observed=scores)

pm.model_to_graphviz(model1)
```

Out[180…



2. `homework3` 题目 `coffee or beer` 中 `h2` 的概率图模型见附件 `cb_h2.pdf`，请你比较和(1)中生成的概率图模型。请说明本题目中**群体差异**和**个体差异**是如何在概率图模型中体现(5分)

- 群体差异，根据xi与zj是否相等，给分均值有 $\alpha_0,\alpha_1,\beta_0,\beta_1$ 之分；根据xi的类别，不同类别人士给分幅度均值有$\sigma_0,\sigma_1,\sigma_2$区分
- 个体差异：由于被试间差异，不同个体给分受自己类别影响同时，还受给分噪音的影响，体现在σ_j彼此并不相同。

3. 请采用MCMC进行采样报告参数α_0，α_1，β_0，β_1，θ，σ_0，σ_1, σ_2的期望并绘制出其对应的后验分布（10分)[采样时请设置 `tune=2000`，`random_seed=422`，`target_accept=0.9`]

In [181…

```
# 使用NUTS采样似乎确实需要用很久很久的时间
with model1:
```

```
    idata1 = pm.sample(tune=2000,random_seed=422,target_accept=0.9)
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>NUTS: [alpha0, alpha1, beta0, beta1, theta, sigma1, sigma2, sigma0, sigmaj]
>CategoricalGibbsMetropolis: [xi]
>BinaryGibbsMetropolis: [zj]
```

Output()

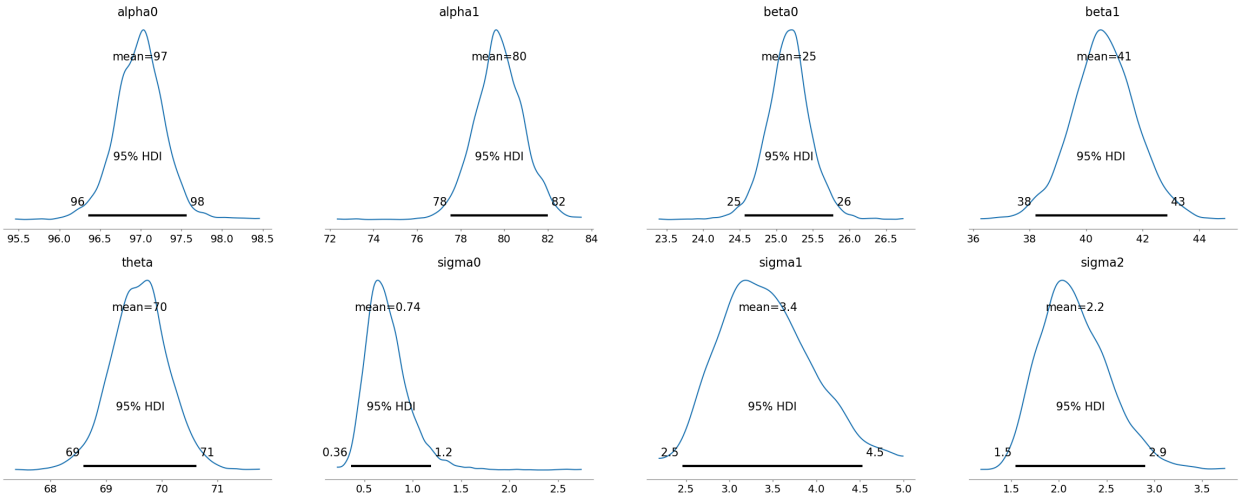
```
Sampling 4 chains for 2_000 tune and 1_000 draw iterations (8_000 + 4_000 draws total) took 61 seconds.
c:\Users\asus\miniconda3\envs\pymc\Lib\site-packages\arviz\stats\diagnostics.py:596: RuntimeWarning: invalid value encounte
red in scalar divide
  (between_chain_variance / within_chain_variance + num_samples - 1) / (num_samples)
The rhat statistic is larger than 1.01 for some parameters. This indicates problems during sampling. See https://arxiv.org/
abs/1903.08008 for details
```

In [190…

```
varlist = ['alpha0','alpha1','beta0','beta1','theta','sigma0','sigma1','sigma2']
az.plot_posterior(idata1,var_names=varlist, hdi_prob=0.95)
az.summary(idata1,var_names=varlist, hdi_prob=0.95)
```

Out[190…

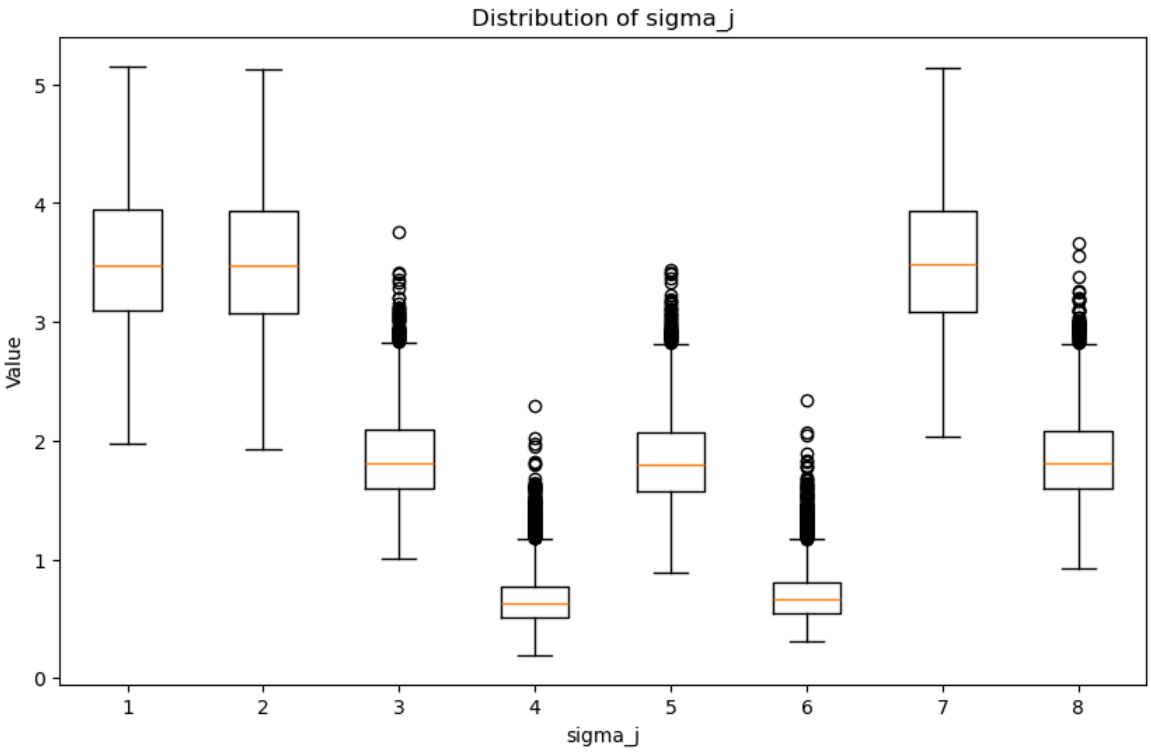
	mean	sd	hdi_2.5%	hdi_97.5%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
alpha0	96.990	0.309	96.354	97.562	0.007	0.005	2159.0	1245.0	1.00
alpha1	79.759	1.147	77.542	81.988	0.026	0.018	2230.0	2090.0	1.00
beta0	25.167	0.306	24.569	25.778	0.005	0.003	4087.0	2422.0	1.00
beta1	40.629	1.157	38.196	42.862	0.018	0.013	4137.0	2331.0	1.00
theta	69.600	0.512	68.594	70.624	0.008	0.005	4573.0	2936.0	1.00
sigma0	0.742	0.236	0.361	1.191	0.010	0.009	993.0	589.0	1.01
sigma1	3.441	0.541	2.459	4.528	0.017	0.012	961.0	1467.0	1.00
sigma2	2.167	0.361	1.545	2.902	0.010	0.007	1543.0	1297.0	1.00



4. 利用(3)中的后验分布，请绘制出综合所有链与所有采样的σ_j的箱线图(5分)

In [61]:

```
# 假设数据来自 trace2.posterior['sigmaj']
sigmaj = idata1.posterior['sigmaj']

# 重组数据维度并选择第一个 dim_1（索引为0）
sigmaj_reshaped = sigmaj.stack(sample=('chain', 'draw'))
sigmaj_reshaped = sigmaj_reshaped.sel(sigmaj_dim_1=0)
sigmaj_reshaped = sigmaj_reshaped.transpose('sample', 'sigmaj_dim_0')

plt.figure(figsize=(10, 6))
plt.boxplot(sigmaj_reshaped)
plt.title('Distribution of sigma_j')
plt.ylabel('Value')
plt.xlabel('sigma_j')
plt.show()
```



5. 斯丢皮德认为，本题目的模型在理想采样情况下likelihood会不小于homework3中h2对应的理想采样结果对应的likelihood。**请问斯丢皮德说的是否正确？第三次作业中h2是否是本次作业模型的嵌套** (5分)

他的理由如下：

> 从模型关系上来看，作业3中h2的假设对应的模型是本次作业假设对应模型的嵌套，homework3的h2对应的模型并未考虑到不同类别人群的群体差异以及群体内部的个体差异，而本次作业对应模型则考虑到这一点，因此表征能力更强。即使尚未存在群体差异与个体差异，本文对应模型在理想状态下对应参数也会收敛到作业3中h2到模型，因此本作业对应模型likelihood应该不小于上次作业模型的likelihood。

本次作业中的假设，如果没有$\sigma_s = 0.1$的限制，确实如斯丢皮德所说，homework3中h2的假设是本作业的嵌套，但是因为$\sigma_s > 0$，个人间的差异始终存在，本次作业模型不会收敛到homework3中的不同被试无差异的情况。

6. 如果我们没有**第一杯饮料是咖啡**这样的假设，请你重新进行采样，在`tune=2000`，`random_seed=422，target_accept=0.9`的情况下重新进行实验，请报告`xi`，`sigmaj`的r hat值，请问为什么会出现这样的情况(5分)

In [62]:
```
# 原来的模型
az.summary(idata1,var_names=['sigmaj'])
```

Out[62]:

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
sigmaj[0, 0]	3.537	0.595	2.530	4.708	0.018	0.010	1121.0	1324.0	1.0
sigmaj[1, 0]	3.526	0.602	2.495	4.698	0.018	0.011	1152.0	1131.0	1.0
sigmaj[2, 0]	1.863	0.367	1.226	2.553	0.010	0.009	1659.0	1234.0	1.0
sigmaj[3, 0]	0.669	0.238	0.307	1.121	0.008	0.009	1287.0	916.0	1.0
sigmaj[4, 0]	1.842	0.378	1.190	2.571	0.010	0.008	1660.0	1314.0	1.0
sigmaj[5, 0]	0.701	0.225	0.355	1.108	0.007	0.009	1315.0	892.0	1.0
sigmaj[6, 0]	3.531	0.595	2.566	4.753	0.018	0.010	1144.0	1255.0	1.0
sigmaj[7, 0]	1.861	0.372	1.258	2.607	0.010	0.009	1614.0	1288.0	1.0

In [191…
```
# set model
with pm.Model() as model2:
    # population mu
    alpha0 = pm.Uniform('alpha0', lower=0., upper=100)
    alpha1 = pm.Uniform('alpha1', lower=0., upper=alpha0)
    beta0 = pm.Uniform('beta0', lower=0., upper=alpha1)
    beta1 = pm.Uniform('beta1', lower=beta0, upper=alpha1)
    theta = pm.Uniform('theta', lower=beta1, upper=alpha1)

    # population sigma
    sigma1 = pm.Uniform('sigma1', lower=0., upper=5)
    sigma2 = pm.Uniform('sigma2', lower=0., upper=sigma1)
```

```python
    sigma0 = pm.Uniform('sigma0', lower=0., upper=sigma1)

    # 设置 xi 和 zj
    xi_probs = np.vstack([[0, 1, 0]] + [[1/3, 1/3, 1/3]] * (n - 1))
    xi_probs = pt.as_tensor_variable(xi_probs)
    xi = pm.Categorical('xi', p=xi_probs, shape=(n,))
    xi = xi[:, None]

    # 设置第一个饮品是0
    zj = pm.Bernoulli("zj", p=[0.5]+[0.5]*(drinks-1), shape=(1, drinks))


    # 设置参数 theta，使用嵌套的 pt.switch 进行二重判定
    mu = pm.Deterministic('mu',
        pt.switch(
        pt.eq(xi, 2),
        theta,
        pt.switch(
            pt.eq(xi, zj),
            pt.switch(pt.eq(xi, 0), alpha0, alpha1),
            pt.switch(pt.eq(xi, 0), beta0, beta1)
        )
    ))

    sigma = pm.Deterministic('sigma',
        pt.switch(
        pt.eq(xi, 2),
        sigma2,
        pt.switch(
            pt.eq(xi, 0),
            sigma0, sigma1
        )
    ))
    sigmaj = pm.Normal('sigmaj', mu=sigma, sigma=0.1)

    kij = pm.Normal('kij', mu=mu, sigma=sigmaj, observed=scores)

with model2:
    idata2 = pm.sample(tune=2000, random_seed=422, target_accept=0.9)
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>NUTS: [alpha0, alpha1, beta0, beta1, theta, sigma1, sigma2, sigma0, sigmaj]
>CategoricalGibbsMetropolis: [xi]
>BinaryGibbsMetropolis: [zj]
Output()
```

```
Sampling 4 chains for 2_000 tune and 1_000 draw iterations (8_000 + 4_000 draws total) took 64 seconds.
c:\Users\asus\miniconda3\envs\pymc\Lib\site-packages\arviz\stats\diagnostics.py:596: RuntimeWarning: invalid value encountered in scalar divide
  (between_chain_variance / within_chain_variance + num_samples - 1) / (num_samples)
c:\Users\asus\miniconda3\envs\pymc\Lib\site-packages\arviz\stats\diagnostics.py:596: RuntimeWarning: divide by zero encountered in scalar divide
  (between_chain_variance / within_chain_variance + num_samples - 1) / (num_samples)
The rhat statistic is larger than 1.01 for some parameters. This indicates problems during sampling. See https://arxiv.org/abs/1903.08008 for details
The effective sample size per chain is smaller than 100 for some parameters.  A higher number is needed for reliable rhat and ess computation. See https://arxiv.org/abs/1903.08008 for details
```

In [192…

```python
az.summary(idata2,var_names=['sigmaj','xi'])
```

```
c:\Users\asus\miniconda3\envs\pymc\Lib\site-packages\arviz\stats\diagnostics.py:596: RuntimeWarning: invalid value encountered in scalar divide
  (between_chain_variance / within_chain_variance + num_samples - 1) / (num_samples)
c:\Users\asus\miniconda3\envs\pymc\Lib\site-packages\arviz\stats\diagnostics.py:596: RuntimeWarning: divide by zero encountered in scalar divide
  (between_chain_variance / within_chain_variance + num_samples - 1) / (num_samples)
c:\Users\asus\miniconda3\envs\pymc\Lib\site-packages\arviz\stats\diagnostics.py:596: RuntimeWarning: divide by zero encountered in scalar divide
  (between_chain_variance / within_chain_variance + num_samples - 1) / (num_samples)
c:\Users\asus\miniconda3\envs\pymc\Lib\site-packages\arviz\stats\diagnostics.py:596: RuntimeWarning: divide by zero encountered in scalar divide
  (between_chain_variance / within_chain_variance + num_samples - 1) / (num_samples)
```

Out[192…

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
sigmaj[0, 0]	4.375	1.023	2.796	5.658	0.474	0.367	5.0	32.0	2.10
sigmaj[1, 0]	3.857	0.911	2.610	5.317	0.376	0.294	7.0	34.0	1.53
sigmaj[2, 0]	2.189	0.394	1.509	2.937	0.012	0.009	1254.0	1321.0	1.00
sigmaj[3, 0]	2.991	2.309	0.397	5.559	1.147	0.879	5.0	34.0	2.10
sigmaj[4, 0]	2.174	0.404	1.447	2.930	0.012	0.009	1264.0	1378.0	1.00
sigmaj[5, 0]	3.021	2.273	0.522	5.569	1.130	0.866	5.0	33.0	2.10
sigmaj[6, 0]	3.874	0.911	2.637	5.347	0.378	0.296	7.0	30.0	1.54
sigmaj[7, 0]	2.188	0.397	1.505	2.950	0.012	0.009	1221.0	1286.0	1.00
xi[0]	1.000	0.000	1.000	1.000	0.000	0.000	4000.0	4000.0	NaN
xi[1]	0.751	0.434	0.000	1.000	0.216	0.165	4.0	4.0	16.93
xi[2]	1.500	0.866	0.000	2.000	0.431	0.330	4.0	4.0	inf
xi[3]	0.501	0.502	0.000	1.000	0.250	0.191	4.0	4.0	8.86
xi[4]	1.500	0.866	0.000	2.000	0.431	0.330	4.0	4.0	inf
xi[5]	0.501	0.502	0.000	1.000	0.249	0.191	4.0	4.0	9.92
xi[6]	0.751	0.435	0.000	1.000	0.216	0.165	4.0	4.0	14.67
xi[7]	1.500	0.866	0.000	2.000	0.431	0.330	4.0	4.0	inf

会发现 `xi` 和 `sigmaj` 的r hat出现了大于1.01的情况，原因是缺少了这样的假设模型引入了更多的不确定性。影响了 `xi` 和 `mu` 的计算逻辑和一致性。通过固定第一个饮品为0，可以在模型设置中提供更清晰的决策依据，帮助模型更有效地探索参数空间并收敛(2分)