STA 602 HW 10

Ryan Tang

November 11th 2022

1 Exercise 7.4

(a) Here I decided to use a prior that doesn't contain much information with a sprinkle of my guesses.

$$\begin{aligned}
Y_i &\overset{iid}{\sim} \mathcal{N}(\theta, \Sigma) \\
\theta &\sim \mathcal{N}(\theta|\mu_0 = \begin{bmatrix}35\\33\end{bmatrix}, \Sigma_0 = \begin{bmatrix}5,0\\0,5\end{bmatrix}) \\
\Sigma &\sim IW(\Sigma|\nu_o = 3, S_o = \begin{bmatrix}100,60\\60,100\end{bmatrix})
\end{aligned}$$

(b) Below are 3 draws of prior predictive distribution samples resulting from prior specifications.

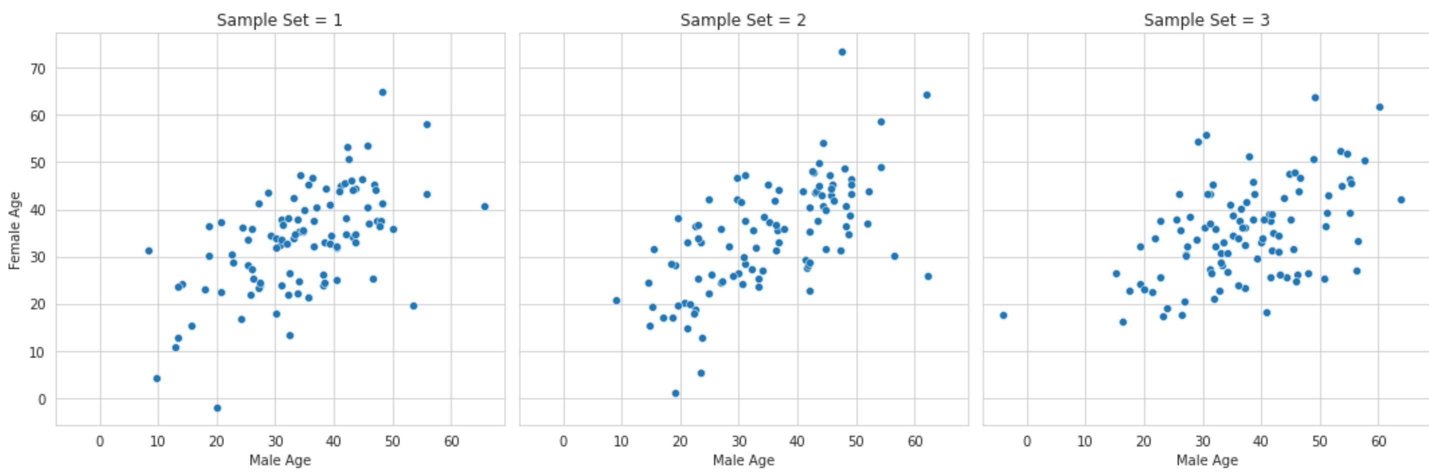

Figure 1: Prior Predictive of Y

(c) Now, we draw posterior samples from the full conditionals using a Gibbs Sampler with the provided dataset Y.

$$\begin{aligned}
\theta|Y,\Sigma &\sim \mathcal{N}(\theta|\mu_n, \Sigma_n) \\
\Sigma_n &= (\Sigma_0^{-1} + N\Sigma^{-1})^{-1} \\
\mu_n &= \Sigma_n(\Sigma^{-1}N\bar{Y} + \Sigma_0^{-1}\mu_o)
\end{aligned}$$

$$\begin{aligned}
\Sigma|Y &\sim IW(\nu_o, S_n) \\
\nu_n &= \nu_o + N \\
S_n &= S_o + S_\theta \\
S_\theta &= \sum_i^N (y_i - \theta)(y_i - \theta)^\intercal
\end{aligned}$$

Below are the plots for the joint θ posterior and the marginal posterior correlation distribution between Y_h and Y_w. We also obtained statistics around $\theta|Y$, too. $\theta_h|Y$ and $\theta_w|Y$ has around 0.903 correlation. $\theta_h|Y$'s 95% confidence interval lies between $[43.35, 46.90]$, and $\theta_w|Y$'s 95% confidence interval lies between $[39.88, 43.18]$.

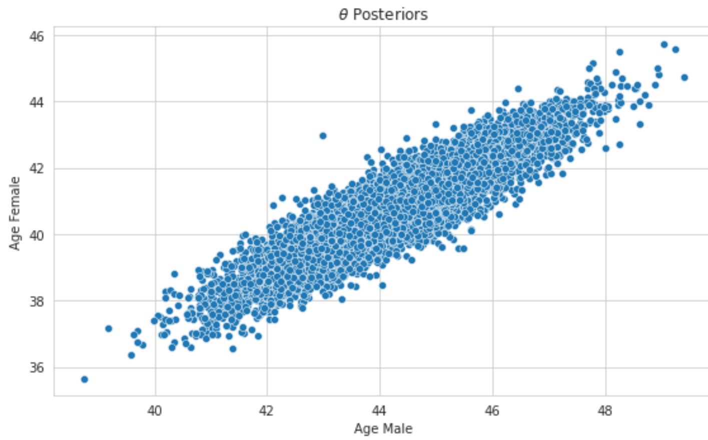

Figure 2: θ Posterior Joint

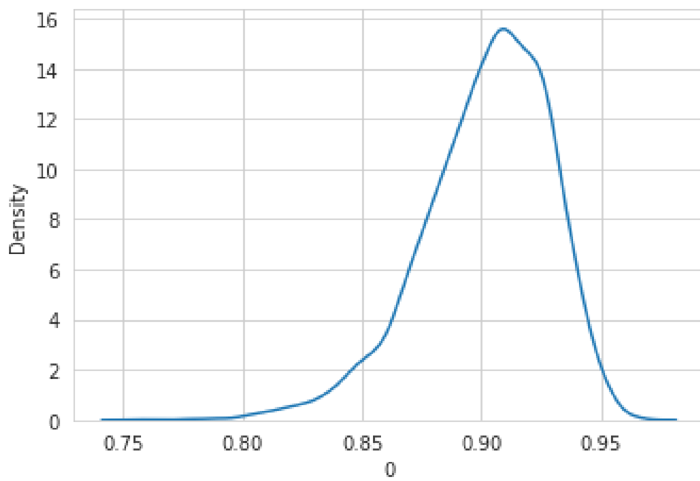

Figure 3: Y correlation distribution

(d.i) Jeffery Priors Using Jeffery Priors, we can see the chain is slow on convergence.

- $\mu_o = \mathbf{0}, \Sigma_o = \infty, \nu_o = 1, S_o = \mathbf{0}$
- Posterior correlation 0.903
- $\theta_h|Y$'s 95% confidence interval lies between $[43.50, 47.15]$
- $\theta_w|Y$'s 95% confidence interval lies between $[40.03, 43.45]$

(d.ii) Unit Information Priors Using Unit Information Priors with MLE:

- Posterior correlation 0.907
- $\theta_h|Y$'s 95% confidence interval lies between $[43.46, 47.11]$
- $\theta_w|Y$'s 95% confidence interval lies between $[39.98, 43.44]$

(d.iii) Diffuse Priors Using an extremely diffuse prior pair:

- Posterior correlation 0.853
- $\theta_h|Y$'s 95% confidence interval lies between $[43.47, 47.17]$
- $\theta_w|Y$'s 95% confidence interval lies between $[40.00, 43.48]$

(e) I certainly think my priors helps. However, all different priors arrive at a similar posterior with just a slight difference. So there are no many differences in this case.

Hypothetically speaking, we end up with 25 samples instead of 100; diffuse priors will perform worse because it wants more data. And Unit Information priors will end up over-fitting easily because of the high MLE estimation variances with small data size. Jeffery priors can work, but it is not designed for multivariate models. If we are concerned about the code start issue, we should inject some knowledge into the prior instead of relying on a pair of uninformative priors.

Appendix The code for generating the posterior samples is included here for completeness.

```python
import pandas as pd
import numpy as np
import scipy.stats as stats
from numpy.linalg import inv

class MVNMean:
    def __init__(self, μ0, Σ0):
        self.μ0 = μ0
        self.Σ0 = Σ0
        self.Λ0 = inv(self.Σ0)

    def sample_prior(self, n=1):
        return np.random.multivariate_normal(self.μ0, self.Σ0, size=n)

    def sample_posterior(self, X, Σ, n=1):
        N = X.shape[0]
        X_bar = X.mean(axis=0)
        Λ = inv(Σ)
        Λn = self.Λ0 + N*Λ
        μn = inv(Λn) @ (Λ @ (N*X_bar) + self.Λ0 @ self.μ0)

        return np.random.multivariate_normal(μn, inv(Λn), size=n)

class MVNCov:
    def __init__(self, v0, S0):
        self.v0 = v0
        self.S0 = S0

    def sample_prior(self, n=1):
        return stats.invwishart(self.v0, self.S0).rvs(size=n)

    def sample_posterior(self, X, θ, n=1):
        N = X.shape[0]
        vn = self.v0 + N
        Sθ = (X - θ).T @ (X - θ)
        Sn = self.S0 + Sθ

        return stats.invwishart(vn, Sn).rvs(size=n)

def run_gibbs_sampler(X, θ_rv, Σ_rv, n_epochs, burnin=500):
    N = X.shape[0]

    samples = []
    Σ = np.eye(X.shape[1]) # default init works for improper priors
    for epoch in range(n_epochs+burnin):
```

```
        θ = θ_rv.sample_posterior(X, Σ).ravel()
        Σ = Σ_rv.sample_posterior(X, θ)

        ys = stats.multivariate_normal(θ, Σ).rvs(size=N)
        post_corr = np.corrcoef(ys.T)[0,1]

        if epoch < burnin:
            continue
        else:
            samples.append((θ, Σ, post_corr))

    return samples

n_epochs = 10000
burin = 500

df = pd.read_csv("agehw.dat", delimiter=" ")
X = df.values

# prior parameters
θ_rv = MVNMean(
    μ0 = np.array([35,33]),
    Σ0 = np.array([
        [100, 60],
        [60, 100]
    ])
)
Σ_rv = MVNCov(
    v0 = 3,
    S0 = np.array([
        [5, 0],
        [0, 5]
    ])
)

samples = run_gibbs_sampler(X, θ_rv, Σ_rv, n_epochs, burin)
```

2 Exercise 7.5

(a) Some Statistics on Observed Data Due to missing data, here we only calculated the statistics on the observed portion of the data.

- Mean: $\hat{\theta}_{A_{obs}} = 24.20, \hat{\theta}_{B_{obs}} = 24.81$
- Variances: $\hat{\sigma}^2_{A_{obs}} = 4.09, \hat{\sigma}^2_{B_{obs}} = 4.69$
- Correlation: $\hat{\rho}_{obs} = 0.616$

(b) We imputed the data using missing at random assumption and multivariate normal sampling model. After we arrived at the imputed dataset, we ran a t-test on $\theta_A - \theta_B$ to see if there were any significant differences between the two response times. The t-test came back with some good results, saying we rejected the null hypothesis. See Figures (4) and (5) for details.

- t-statistics $= -3.455$, p-value $= 0.0018$
- 95% Confidence Interval $[-0.985, -0.238]$

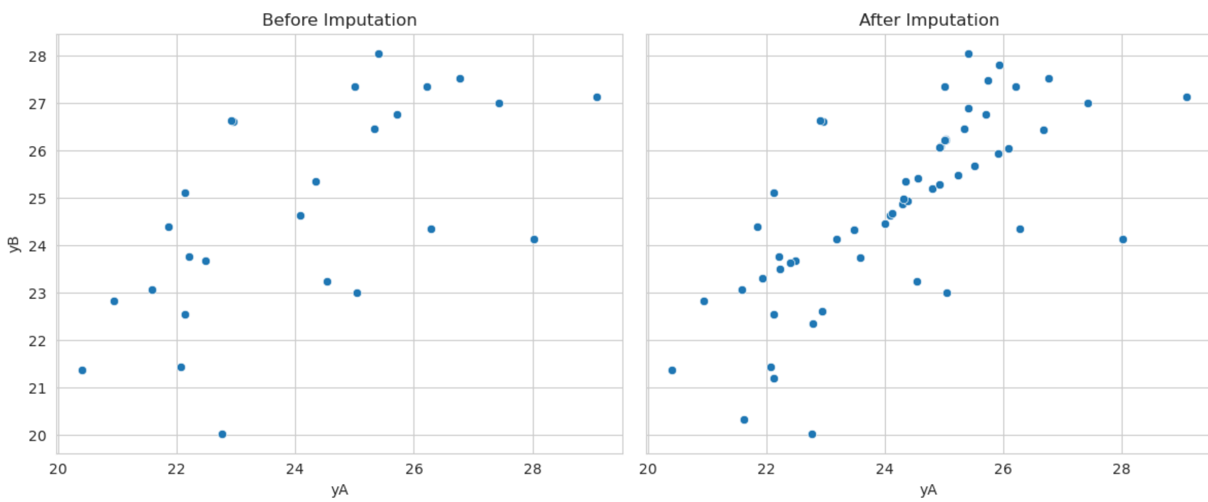

Figure 4: Imputation before and after comparison

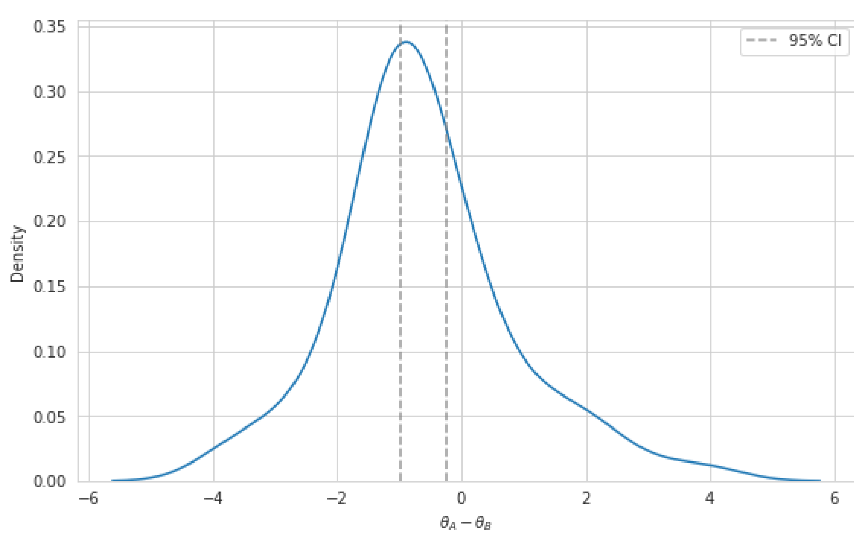

Figure 5: θ_A - θ_B Density Its Mean Interval

(c) Gibbs Sample with MVN Imputation Lastly, we use the Gibbs sampler to do the imputation and calculate the statistics about $\theta_A - \theta_B$. The posterior density is shown below in Figure 6.

- The posterior mean is -0.615
- 95% Confidence Interval $[-1.016, -0.207]$

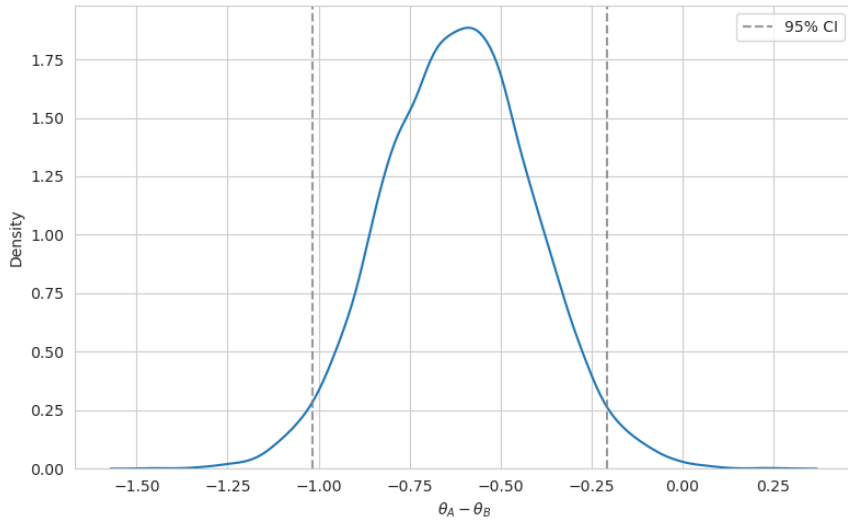

Figure 6: θ_A - θ_B Density - Gibbs Sampler

Appendix The code for generating the posterior samples is included here for completeness.

```python
import pandas as pd
import numpy as np
import scipy.stats as stats
from numpy.linalg import inv
from numpy.random import multivariate_normal
from scipy.stats import invwishart

class MVNMean:
    def __init__(self, μ0, Σ0):
        self.μ0 = μ0
        self.Σ0 = Σ0
        self.Λ0 = inv(Σ0)

    def sample_prior(self, size=1):
        return multivariate_normal(self.μ0, self.Σ0, size=size)

    def sample_posterior(self, X, Σ, size=1):
        n = X.shape[0]
        X_mean = X.mean(axis=0)

        Λ = inv(Σ)
        Λn = self.Λ0 + n * Λ
        Σn = inv(Λn)
        μn = Σn @ (Λ @ (n*X_mean) + self.Λ0 @ self.μ0)
        return multivariate_normal(μn, Σn, size=size)

class MVNSigma:
    def __init__(self, v0, S0):
        self.v0 = v0
        self.S0 = S0

    def sample_prior(self, size=1):
        return invwishart(self.v0, self.S0).rvs(size=size)
```

```python

    def sample_posterior(self, X, θ, size=1):
        n = X.shape[0]
        S = (X - θ).T @ (X - θ)

        vn = self.v0 + n
        Sn = self.S0 + S
        return invwishart(vn, Sn).rvs(size=size)

def impute_B(B_mean, yA, A_mean, B_var, A_var, rho):
    return B_mean + (yA - A_mean) * np.sqrt(B_var/A_var) * rho

def impute_A(A_mean, yB, B_mean, A_var, B_var, rho):
    return A_mean + (yB - B_mean) * np.sqrt(A_var/B_var) * rho

def cov2corr(Σ):
    var = np.diag(Σ).reshape(-1, 1)
    return Σ / np.sqrt(var @ var.T)

def impute_X(X, θ, Σ):
    θA, θB = θ
    σ2A, σ2B = Σ[0,0], Σ[1,1]
    ρ = cov2corr(Σ)[0,1]

    imputed_X = []
    for (yA, yB) in X:
        if np.isnan(yA):
            imputed_X.append((impute_A(θA, yB, θB, σ2A, σ2B, ρ), yB))
        elif np.isnan(yB):
            imputed_X.append((yA, impute_B(θB, yA, θA, σ2B, σ2A, ρ)))
        else:
            imputed_X.append((yA, yB))

    return np.array(imputed_X)

# get data
df = pd.read_fwf("interexp.dat")
X = df.values
N = df.shape[0]

# prior parameters
θ_rv = MVNMean(
    μ0 = df.mean().values,
    Σ0 = df.dropna().cov().values
)

Σ_rv = MVNSigma(
    v0 = 2,
    S0 = df.dropna().cov().values
)

samples = []
nEpochs = 10000
Σ = Σ_rv.sample_prior()
θ = θ_rv.sample_prior().ravel()
imputed_X = impute_X(X, θ, Σ)
for epoch in range(nEpochs):
    θ = θ_rv.sample_posterior(imputed_X, Σ).ravel()
    Σ = Σ_rv.sample_posterior(imputed_X, θ)
    imputed_X = impute_X(X, θ, Σ)

    diff = θ[0] - θ[1]

    samples.append((θ, Σ, diff))
```

3 Exercise 7.6

(a) Separate Posterior Parameters Comparisons Below graph shows the comparison between $\theta_{d,j}$ and $\theta_{n,j}$. At the same time, we also calculated the $P[\theta_{d,j} > \theta_{n,j}|Y]$. The obtained probability was all 100% $\forall j \in \{1 \dots 7\}$.

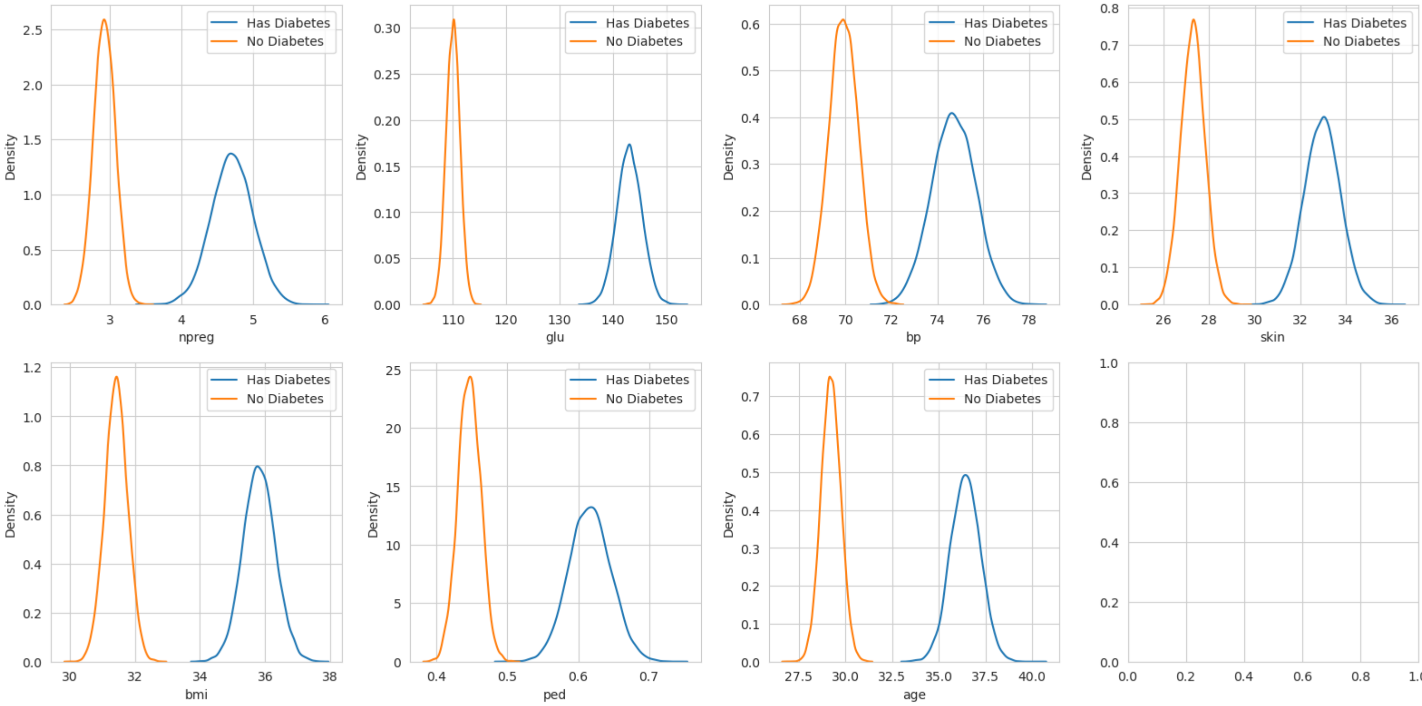

Figure 7: Posterior Comparison between the two groups

(b) $\Sigma|Y$ Comparison Lastly, we investigate the covariance samples. I averaged over all sampled posterior covariance for each group and plotted each entry color coded by the group. Below is the graph, Figure 8. We can see glucose is the feature that has the largest variation in magnitude between the two groups. Another interesting thing is to compare the correlation instead.

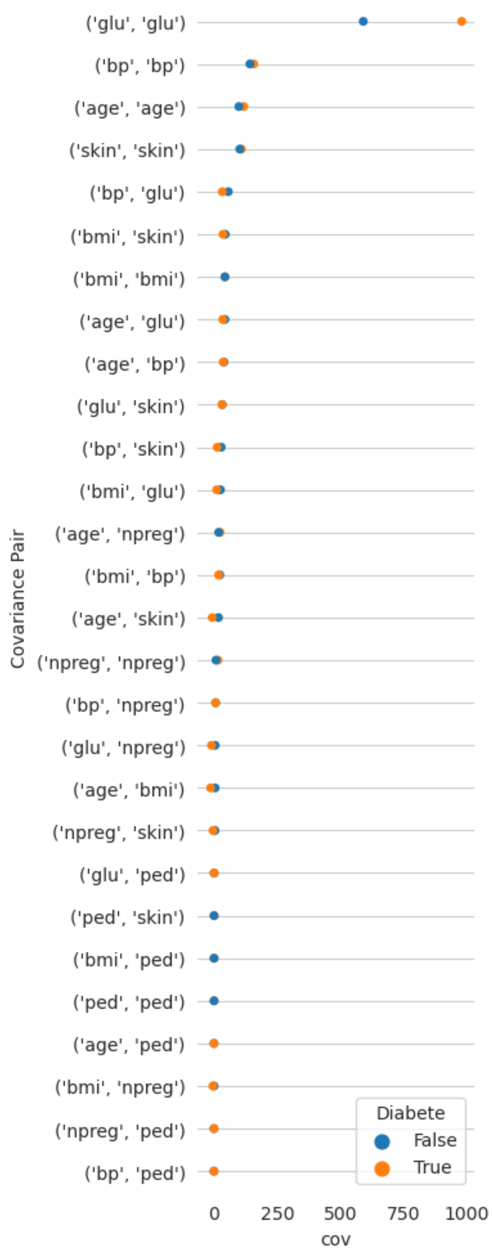

Figure 8: Posterior Σ Comparison by Entries

Appendix The code for generating the posterior samples is included here for completeness.

```
import pandas as pd
import numpy as np
import scipy.stats as stats
from numpy.linalg import inv
from numpy.random import multivariate_normal
from scipy.stats import invwishart

class MVNMean:
    def __init__(self, μ0, Σ0):
        self.μ0 = μ0
```

```python
        self.Σ0 = Σ0
        self.Λ0 = inv(Σ0)

    def sample_prior(self, size=1):
        return multivariate_normal(self.μ0, self.Σ0, size=size)

    def sample_posterior(self, X, Σ, size=1):
        n = X.shape[0]
        X_mean = X.mean(axis=0)

        Λ = inv(Σ)
        Λn = self.Λ0 + n * Λ
        Σn = inv(Λn)
        μn = Σn @ (Λ @ (n*X_mean) + self.Λ0 @ self.μ0)
        return multivariate_normal(μn, Σn, size=size)

class MVNSigma:
    def __init__(self, v0, S0):
        self.v0 = v0
        self.S0 = S0

    def sample_prior(self, size=1):
        return invwishart(self.v0, self.S0).rvs(size=size)

    def sample_posterior(self, X, θ, size=1):
        n = X.shape[0]
        S = (X - θ).T @ (X - θ)

        vn = self.v0 + n
        Sn = self.S0 + S
        return invwishart(vn, Sn).rvs(size=size)

# getting the data
df = pd.read_fwf("azdiabetes.dat", skiprows=[0], header=None)
df.columns = "npreg glu bp skin bmi ped age diabetes".split()
df.diabetes = df.diabetes == 'Yes'

df_d = df[df.diabetes].drop(['diabetes'], axis=1)
df_n = df[~df.diabetes].drop(['diabetes'], axis=1)

# sampling for diabete cases
X = df_d.values
N = df_d.shape[0]

# prior parameters
θ_rv = MVNMean(
    μ0 = df_d.mean().values,
    Σ0 = df_d.cov().values,
)

Σ_rv = MVNSigma(
    v0 = df_d.shape[1] + 2,
    S0 = df_d.cov().values,
)

nEpochs = 10000
d_samples = []

Σ = Σ_rv.sample_prior()
θ = θ_rv.sample_prior().ravel()
for epoch in range(nEpochs):
    θ = θ_rv.sample_posterior(X, Σ).ravel()
    Σ = Σ_rv.sample_posterior(X, θ)

    d_samples.append((θ, Σ))
```

```

d_theta_samples = pd.DataFrame(map(lambda x: x[0], d_samples), columns=df.columns[:-1])
d_sigma_samples = np.array(list(map(lambda x: x[1], d_samples)))

# sampling for non diabete cases
X = df_n.values
N = df_n.shape[0]

# prior parameters
θ_rv = MVNMean(
    μ0 = df_n.mean().values,
    Σ0 = df_n.cov().values,
)

Σ_rv = MVNSigma(
    v0 = df_n.shape[1] + 2,
    S0 = df_n.cov().values,
)

nEpochs = 10000
n_samples = []

Σ = Σ_rv.sample_prior()
θ = θ_rv.sample_prior().ravel()
for epoch in range(nEpochs):
    θ = θ_rv.sample_posterior(X, Σ).ravel()
    Σ = Σ_rv.sample_posterior(X, θ)

    n_samples.append((θ, Σ))

n_theta_samples = pd.DataFrame(map(lambda x: x[0], n_samples), columns=df.columns[:-1])
n_sigma_samples = np.array(list(map(lambda x: x[1], n_samples)))
```