BDA - Assignment 7

Anonymous

Contents:

Exercise 1

- Exercise 1 A
- Exercise 1 B
- Exercise 1 C
- Exercise 1 D

Exercise 2

- General Explanations
- Separate Model
- Exercise 2 A
- Exercise 2 B
- Exercise 2 C
- Exercise 2 D
- Pooled Model
- Exercise 2 A
- Exercise 2 B
- Exercise 2 C
- Exercise 2 D
- Hierarchical Model
- Exercise 2 A
- Exercise 2 B
- Exercise 2 C
- Exercise 2 D

Exercise 1

Exercise 1 A

In [1]:

```
#read the bioassay data
import numpy as np
import csv
import pystan

#reading the data
file = open(r".\drowning.txt", 'r')
temp = file.read().splitlines()

years = []
drowned = []
for line in temp:
    values = line.split(" ")
    years.append(float(values[0]))
    drowned.append(float(values[1]))
years = np.array(years)
drowned = np.array(drowned)
```

The mistakes were that: 1) `real<upper=0> sigma` should have been `real<lower=0> sigma`

2) There is no computation of `mu_pred` required for finding out ypred.

3) `ypred = normal_rng(mu, sigma)` should have been `ypred = normal_rng(mu_pred, sigma)`

In [8]:

```
fixed_model = '''
data {
    int<lower=0> N; // number of data points
    vector[N] x; // observation year
    vector[N] y; // observation number of drowned
    real xpred; // prediction year
}
parameters {
    real alpha;
    real beta;
    real<lower=0> sigma;
}
transformed parameters {
    vector[N] mu;
    real pred_mu;
    mu = alpha + beta * x;
    pred_mu = alpha + beta * xpred;
}
model {
    y ~ normal(mu, sigma);
}
generated quantities {
    real ypred;
    ypred = normal_rng(pred_mu, sigma);
}
'''

data = {
    "N": len(years),
    "x": years,
    "y": drowned,
    "xpred": 2020

}
```

In [17]:

```
posterior = pystan.StanModel(model_code=fixed_model)
fit = posterior.sampling(data=data)
print(fit)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_5d2a30df9c6b111b7b660b34acbd17
fc NOW.
WARNING:pystan:941 of 4000 iterations saturated the maximum tree depth of 10 (23.5 %)
WARNING:pystan:Run again with max_treedepth larger than 10 to avoid saturation
Inference for Stan model: anon_model_5d2a30df9c6b111b7b660b34acbd17fc.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

          mean se_mean     sd   2.5%    25%    50%    75%  97.5%  n_eff   Rhat
alpha   2464.2   21.98 728.57 1079.6 1984.2 2472.3 2925.8 3936.1   1099    1.0
beta     -1.17    0.01   0.36   -1.9   -1.4  -1.17  -0.93  -0.47   1099    1.0
sigma    26.35    0.08   3.08  21.15   24.2  26.11  28.14  33.24   1393    1.0
mu[1]    157.0    0.22   8.25  140.5 151.62 156.97 162.45 173.44   1371    1.0
mu[2]   155.84    0.21   7.94 139.97  150.7 155.83 161.05 171.66   1398    1.0
mu[3]   154.67     0.2   7.63 139.43 149.76 154.62 159.69 169.91   1429    1.0
mu[4]   153.51    0.19   7.33 138.93 148.79 153.47 158.34 168.12   1465    1.0
mu[5]   152.34    0.18   7.03 138.52  147.8 152.31 156.96 166.32   1508    1.0
mu[6]   151.18    0.17   6.74 138.08  146.8 151.08 155.64 164.56   1561    1.0
mu[7]   150.01    0.16   6.46 137.38 145.79  149.9  154.3 162.88   1624    1.0
mu[8]   148.85    0.15   6.19 136.79 144.78 148.72 152.96 161.15   1699    1.0
mu[9]   147.68    0.14   5.92 136.14 143.78 147.55 151.61 159.55   1791    1.0
mu[10]  146.52    0.13   5.67 135.46 142.78 146.41 150.23 157.87   1892    1.0
mu[11]  145.35    0.12   5.43 134.65 141.81 145.24 148.95 156.14   2010    1.0
mu[12]  144.19    0.11    5.2 133.96 140.82 144.09 147.63 154.45   2154    1.0
mu[13]  143.02     0.1   4.99 133.25 139.73 142.96 146.32 152.88   2329    1.0
mu[14]  141.86     0.1    4.8 132.42 138.72 141.82 145.03 151.48   2542    1.0
mu[15]  140.69    0.09   4.63 131.57 137.63 140.61 143.76 149.88   2797    1.0
mu[16]  139.52    0.08   4.49 130.76  136.6 139.44 142.49 148.46   3095    1.0
mu[17]  138.36    0.07   4.37 129.84  135.5 138.31 141.22 147.14   3415    1.0
mu[18]  137.19    0.07   4.29 128.73 134.42 137.14  140.0 145.87   3696    1.0
mu[19]  136.03    0.07   4.27 127.71 133.24  136.0 138.79 144.53   3932    1.0
mu[20]  134.86    0.07   4.18 126.67  132.1 134.81 137.66 143.18   4025    1.0
mu[21]   133.7    0.07   4.17  125.6 131.01 133.64 136.48 141.95   4031    1.0
mu[22]  132.53    0.07    4.2 124.39 129.77 132.48 135.32 140.91   3970    1.0
mu[23]  131.37    0.07   4.27 123.02  128.6  131.3 134.18  139.9   3756    1.0
mu[24]   130.2    0.07   4.36 121.61 127.34 130.19 133.07 138.99   3484    1.0
mu[25]  129.04    0.08   4.48 120.19  126.1  129.0 132.04 137.96   3159    1.0
mu[26]  127.87    0.09   4.62 118.74 124.83 127.81 130.97 137.03   2856    1.0
mu[27]  126.71    0.09   4.79 117.19 123.57 126.68 129.92 136.22   2594    1.0
mu[28]  125.54     0.1   4.98 115.55 122.27 125.56 128.88 135.31   2374    1.0
mu[29]  124.38    0.11   5.19 113.98  121.0 124.39 127.82 134.48   2193    1.0
mu[30]  123.21    0.12   5.41 112.39 119.67 123.26 126.79 133.86   2044    1.0
mu[31]  122.05    0.13   5.65 110.75 118.37 122.09 125.76 133.17   1907    1.0
mu[32]  120.88    0.14    5.9 109.09 117.04 120.95 124.77 132.45   1787    1.0
mu[33]  119.72    0.15   6.17 107.33 115.71 119.73 123.81 131.77   1690    1.0
mu[34]  118.55    0.16   6.44  105.6 114.36 118.57 122.86 131.19   1610    1.0
mu[35]  117.38    0.17   6.72 103.83 113.04  117.4  121.8 130.46   1543    1.0
mu[36]  116.22    0.18   7.01 102.12 111.77 116.22 120.84 129.86   1488    1.0
mu[37]  115.05    0.19   7.31 100.42 110.44 115.04 119.88 129.32   1443    1.0
mu[38]  113.89     0.2   7.61  98.56  109.1 113.89 118.96  128.7   1406    1.0
mu[39]  112.72    0.21   7.92  96.64  107.7 112.74 117.93 128.17   1375    1.0
mu[40]  111.56    0.22   8.23   95.0 106.36 111.56 116.95 127.67   1347    1.0
pred_mu 110.39    0.23   8.54   93.2 105.02 110.38 116.01 127.12   1323    1.0
ypred   110.77     0.5  28.15   55.4  91.86 110.51 129.34 166.24   3187    1.0
lp__    -146.6    0.04   1.25 -149.8 -147.2 -146.3 -145.7 -145.2   1160    1.0

Samples were drawn using NUTS at Thu Oct 20 02:15:55 2022.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

Exercise 1 B

We know that beta follows a normal distribution N(0, σ).

We also know that the 0.005 percentile is -69, and that the 0.995 percentile is 69.

Ideally, what we would try to do in this case in order to find out σ would be to solve for the integral between values -69 and 69 and equal that to 69. But given that the Normal Distribution's form does not have a compact integral, we need to find other ways to solve this.

That's why I will be using the inverse error function (computed I assume with the Taylor Expansion under the hood) to approximate the correct sigma value.

We know that the formula of the CDF is $\frac{1}{2} * [1 + erf(\frac{x-\mu}{\sigma*\sqrt{2}})$. Starting from this, we can prove that the formula for a quantile is $\mu + \sigma\sqrt{2} * erf^{-1}(2p-1)$.

(We also know that μ is 0)

Therefore,

$69 = 0 + \sigma\sqrt{2} * erf^{-1}(2 * 0.995 - 1)$
$69 = 0 + \sigma\sqrt{2} * erf^{-1}(0.99)$
$69 = 0 + \sigma\sqrt{2} * 1.8213...$
$\sigma = 26.79$, using Scipy's erfinv function

In [9]:

```
from scipy.special import erfinv

beta_sigma = (69 / (erfinv(0.99) * np.sqrt(2))).round(2)
print(f"Value of beta's sigma is {beta_sigma}")
```

```
Value of beta's sigma is 26.79
```

Exercise 1 C

In [10]:

```
fixed_model_beta_prior = '''
data {
    int<lower=0> N; // number of data points
    vector[N] x; // observation year
    vector[N] y; // observation number of drowned
    real xpred; // prediction year
}
parameters {
    real alpha;
    real beta;
    real<lower=0> sigma;
}
transformed parameters {
    vector[N] mu;
    real pred_mu;
    mu = alpha + beta * x;
    pred_mu = alpha + beta * xpred;
}
model {
    beta ~ normal(0, 26.79); //added this line to account for beta's prior
    y ~ normal(mu, sigma);
}
generated quantities {
    real ypred;
    ypred = normal_rng(pred_mu, sigma);
}
'''

data = {
    "N": len(years),
    "x": years,
    "y": drowned,
    "xpred": 2020
}
```

Solved this by adding line `beta ~ normal(0, 26.79);` in block `model`

Exercise 1 D

My thought process for this exercise is that the intercept should be somewhat compensating for the values of $\beta * x$. Taking this into consideration, given that beta's mean is 0, and the assignment tells us the mean of drownings is 138, this means we'll set the mean of our intercept's prior to 138.

The question is now how to find the variance. We'll apply the same logic we did when computing the variance for the β hyperparameter.

Given that the mean of our x's value is 2000, we'll ask that $Pr(-2000 * 69 < \alpha < 2000 * 69) = 0.99$.
Using the same function as before, we get that:

In [13]:

```
beta_sigma = ((2000*69 - 138) / (erfinv(0.99) * np.sqrt(2))).round(2)
print(f"Value of beta's sigma is {beta_sigma}")
```

```
Value of beta's sigma is 53521.4
```

Therefore the prior of alpha will be N(138, 53521.4)

In [19]:

```
fixed_model_beta_alpha_priors = '''
data {
    int<lower=0> N; // number of data points
    vector[N] x; // observation year
    vector[N] y; // observation number of drowned
    real xpred; // prediction year
}
parameters {
    real alpha;
    real beta;
    real<lower=0> sigma;
}
transformed parameters {
    vector[N] mu;
    real pred_mu;
    mu = alpha + beta * x;
    pred_mu = alpha + beta * xpred;
}
model {
    alpha ~ normal(138, 53521.4); // added this line to account for alpha's prior
    beta ~ normal(0, 26.79); // added this line to account for beta's prior
    y ~ normal(mu, sigma);
}
generated quantities {
    real ypred;
    ypred = normal_rng(pred_mu, sigma);
}
'''

data = {
    "N": len(years),
    "x": years,
    "y": drowned,
    "xpred": 2020,
}
```

This is the line added for alpha's prior: `alpha ~ normal(138, 53521.4)`.

In [20]:

```
posterior = pystan.StanModel(model_code=fixed_model_beta_alpha_priors)
fit = posterior.sampling(data=data)
print(fit)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_7405d454bc49856d0653f9737e8f0e NOW.
WARNING:pystan:847 of 4000 iterations saturated the maximum tree depth of 10 (21.2 %)
WARNING:pystan:Run again with max_treedepth larger than 10 to avoid saturation
Inference for Stan model: anon_model_7405d454bc49856d0653f9737e8f0e.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

[illegible]

Samples were drawn using NUTS at Thu Oct 20 22:17:12 2022.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

Exercise 2

In [45]:

```
import matplotlib.pyplot as plt

# read data
file = open(r".\factory.txt", 'r')
temp = file.read().splitlines()

quality = [[] for i in range(6)]
for line in temp:
    values = line.split(" ")
    while True:
        try:
            values.remove("")
        except:
            break
    for i in range(len(values)):
        quality[i].append(float(values[i]))
```

First of all, I will explain why I think the model shows such strange results:

The reason for that is the low variance on both μ and σ which impose bias. If the number of samples is low and the prior distribution is narrow, then its influence on the posterior will be high and the data will be neglected. This is why in all of the examples below I made sure to choose a weakly informative prior with high variance for the normal distribution for μ and a wider distribution for σ.

More on this topic could be found here: https://discourse.mc-stan.org/t/standardizing-predictors-and-outputs-in-a-hierarchical-model/2974/9

Additionally, there is also a mismatch between the unit scale of the data and the unit scale of the prior. This explanation was taken from https://mc-stan.org/users/documentation/case-studies/weakly_informative_shapes.html.

The differences between the 3 types of models: 1) The separate model builds itself on the assumption that all the machines are independent of each other. That's why each of the values for the priors (we have 6 priors) are independent draws from a pre-defined distribution. And this is why we cannot use this kind of model to infer the behaviour of a new 7th machine.

2) The pooled model builds itself on the assumption that all the machines are the same, thus all of their samples can be treated as belonging to a single machine, a pooled one. That is why there is a single pair of parameters μ and σ which result in a single prior.

3) The hierarchical model is the middleground solution between the separate model and the pooled model. It builds on the assumption that all the machines have some common ground, but they can all behave differently. The way it achieves this is by allowing each of the machines to have its own prior, but all of the priors are "sampled" from a distribution defined with tunable hyperparameters. This way, we achieve both the independence of the separate model and the similarity of the pooled model.

Separate Model

Exercise 2 A

$y_j \sim N(\mu_j, \sigma_j)$
$\mu_j \sim N(0, 50)$
$\sigma_j \sim Inv-\chi^2(1)$

The reason for choosing a variance of 50 for the μ_j term is due to the low number of samples, which would have been completely misbehaved if we were to choose a low variance, and therefore the mean value to chose would've been biased (See explanations in the beginning of the exercise). I tried with 1, and the mean was around 0.5, and with 10 it went up to about 7-10, whereas with 50, it allows the "data to speak".

The same thing applies for the σ_j value, where a distribution that is too narrow would add bias.

Exercise 2 B

In [46]:

```
separate_model = '''
data {
    int<lower=0> N;
    int<lower=0> J;
    matrix[N, J] y;
}

parameters {
    vector[J] mu;
    vector<lower=0>[J] sigma;
}

model {
    // priors
    for (j in 1:J){
        mu[j] ~ normal(0, 50);
        sigma[j] ~ inv_chi_square(1);
    }

    // likelihood
    for (j in 1:J)
        y[,j] ~ normal(mu[j], sigma[j]);
}

generated quantities {
    real ypred[J];

    for (j in 1:J) {
        ypred[j] = normal_rng(mu[j], sigma[j]);
    }
}
'''
```

In [56]:

```
data = {
    "N": len(quality),
    "J": len(quality[0]),
    "y": quality
}

posterior = pystan.StanModel(model_code=separate_model)
fit = posterior.sampling(data=data)
print(fit)
```

```
[illegible]
```

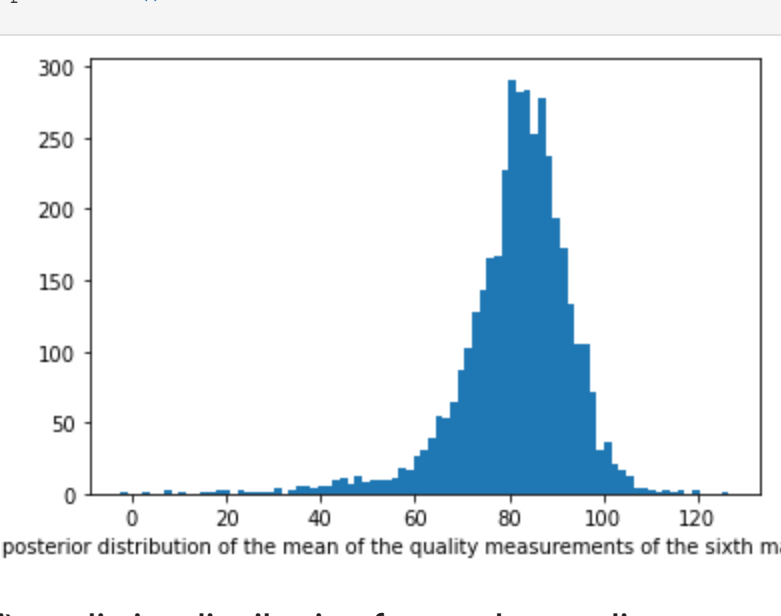
Exercise 2 C

i) posterior distribution of the mean of the quality measurements of the sixth machine

In [97]:

```
last_machine_mu = fit.extract("mu")

plt.hist(last_machine_mu['mu'][:,5], bins="auto")
plt.xlabel('posterior distribution of the mean of the quality measurements of the sixth machine')
plt.show()
```

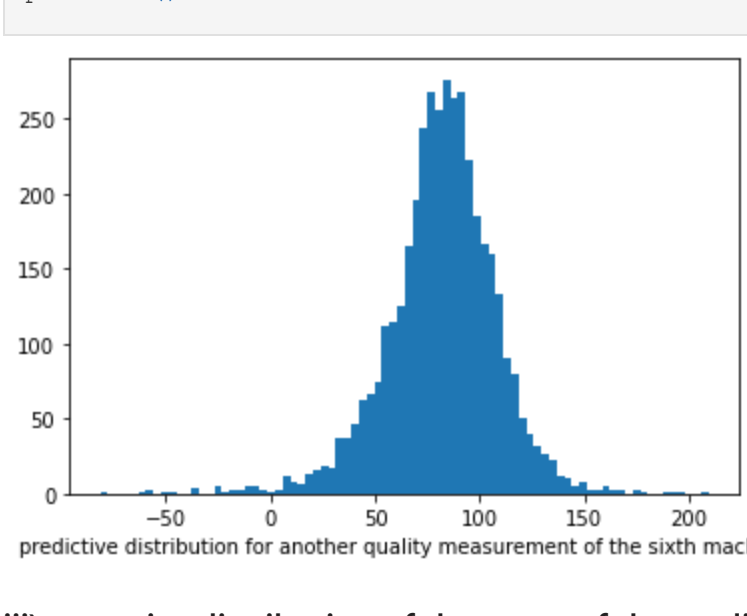


ii) predictive distribution for another quality measurement of the sixth machine

In [98]:

```
last_machine_pred = fit.extract("ypred")

plt.hist(last_machine_pred['ypred'][:,5], bins='auto')
plt.xlabel('predictive distribution for another quality measurement of the sixth mach
plt.show()
```



iii) posterior distribution of the mean of the quality measurements of the seventh machine

Given that each machine has its own model, and we know no data about machine nr 7, we cannot find out the posterior distribution of its mean.

Exercise 2 D

In [68]:

```
separate_model = '''
data {
    int<lower=0> N;
    int<lower=0> J;
    matrix[N, J] y;
}

parameters {
    vector[J] mu;
    vector<lower=0>[J] sigma;
}


model {
    // priors
    for (j in 1:J){
        mu[j] ~ normal(0, 10);
        sigma[j] ~ gamma(1,1);
    }

    // likelihood
    for (j in 1:J)
        y[,j] ~ normal(mu[j], sigma[j]);
}

generated quantities {
    real ypred[J];

    for (j in 1:J) {
        ypred[j] = normal_rng(mu[j], sigma[j]);
    }
}
'''
```

In [69]:

```
data = {
    "N": len(quality),
    "J": len(quality[0]),
    "y": quality
}

posterior = pystan.StanModel(model_code=separate_model)
fit = posterior.sampling(data=data)
print(fit)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_09f23fe9ba9bf5fd740161bbea1180
69 NOW.
Inference for Stan model: anon_model_09f23fe9ba9bf5fd740161bbea118069.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

           mean se_mean     sd   2.5%    25%    50%    75%  97.5%  n_eff   Rhat
mu[1]      50.4    0.16   9.11  30.28  44.53  51.01  57.08  65.94   3061    1.0
mu[2]     48.88    0.22   12.3  25.26  40.48  48.61  56.91  73.99   3153    1.0
mu[3]     58.22    0.21  12.07  33.13  49.83  59.14   67.3  78.25   3159    1.0
mu[4]     47.81    0.21  11.93  24.87  39.82  47.75  55.99  72.02   3340    1.0
mu[5]     60.77    0.23   12.9  34.15  51.65   61.6   70.7  82.22   3205    1.0
mu[6]     51.57    0.18  10.78  29.75  44.27  52.22  59.42  70.39   3663    1.0
sigma[1]  15.99    0.06   3.45  10.22  13.44  15.64  18.16  23.68   2932    1.0
sigma[2]  24.66    0.08   4.53  15.92  21.69  24.51  27.58  33.97   3347    1.0
sigma[3]  16.14    0.08   4.52   8.35  12.79  15.84  19.15  25.66   3065    1.0
sigma[4]  26.33    0.08   4.49   17.8  23.33  26.17  29.18   35.6   3388    1.0
sigma[5]  15.71    0.09   4.85   7.34  11.95  15.52  18.97  25.64   3223    1.0
sigma[6]   18.5    0.07   3.94  11.68  15.57  18.26  21.14  26.87   3626    1.0
ypred[1]  50.01     0.3   18.5   9.07  38.59   51.6  62.72   82.6   3692    1.0
ypred[2]  49.09    0.45  28.04 -10.95  31.38   50.9  68.62  99.02   3946    1.0
ypred[3]   58.4    0.35  20.81  10.49  46.11  60.68   73.4  91.98   3588    1.0
ypred[4]  47.63    0.47  29.23  -12.2  28.58  49.07  67.93 100.84   3849    1.0
ypred[5]  60.39    0.37  21.18  11.93  48.04  63.03   75.6  94.57   3309    1.0
ypred[6]  51.17    0.37  22.13   3.56  37.32  52.87  66.24  90.29   3662    1.0
lp__     -346.9    0.06   2.31 -352.2 -348.3 -346.7 -345.2 -343.3   1708    1.0

Samples were drawn using NUTS at Sun Oct 23 23:41:04 2022.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

In [71]:

```
mu_1_data = fit.extract("mu")["mu"][:, 0]

print(f"The 90% interval for the posterior mu1 is [{np.quantile(mu_1_data, 0.05)} , {r
```

```
The 90% interval for the posterior mu1 is [34.55500750246398 , 64.21656206452728]
```

Pooled Model

Exercise 2 A

$y \sim N(\mu, \sigma)$
$\mu \sim N(0, 10)$
$\sigma \sim Inv - \chi^2(1)$

The reasoning is similar to the separate model, but given that we pool together all the data, we have a higher number of inputs which would remove the bias imposed by a narrower prior. This is why I chose to use $N(0, 10)$ as the prior for μ, and $Inv - \chi^2(10)$ for σ

Exercise 2 B

In [126…

```
pooled_model = '''
data {
    int<lower=0> N;
    int<lower=0> J;
    matrix[N, J] y;
}

parameters {
    real mu;
    real<lower=0> sigma;
}


model {
    // priors
    mu ~ normal(0, 10);
    sigma ~ inv_chi_square(10);

    // likelihood
    for (j in 1:J)
        y[,j] ~ normal(mu, sigma);
}

generated quantities {
    real ypred;

    ypred = normal_rng(mu, sigma);
}
'''
```

In [127…

```
data = {
    "N": len(quality),
    "J": len(quality[0]),
    "y": quality
}

posterior = pystan.StanModel(model_code=pooled_model)
fit = posterior.sampling(data=data)
print(fit)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_5a9a0618da3906392c9303c87c811e
45 NOW.
Inference for Stan model: anon_model_5a9a0618da3906392c9303c87c811e45.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

        mean se_mean     sd   2.5%    25%    50%    75%  97.5%  n_eff   Rhat
mu     81.55    0.17   5.33  69.65  78.78   82.2  85.03   89.9   1024    1.0
sigma  20.29    0.13   4.02  14.61  17.63   19.6  22.12  29.82   1006    1.0
ypred  81.78    0.36  21.15  37.77  68.83  82.32  95.66 121.63   3409    1.0
lp__  -155.7    0.03   1.12 -158.6 -156.1 -155.4 -154.9 -154.6   1033    1.0

Samples were drawn using NUTS at Mon Oct 24 00:34:46 2022.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

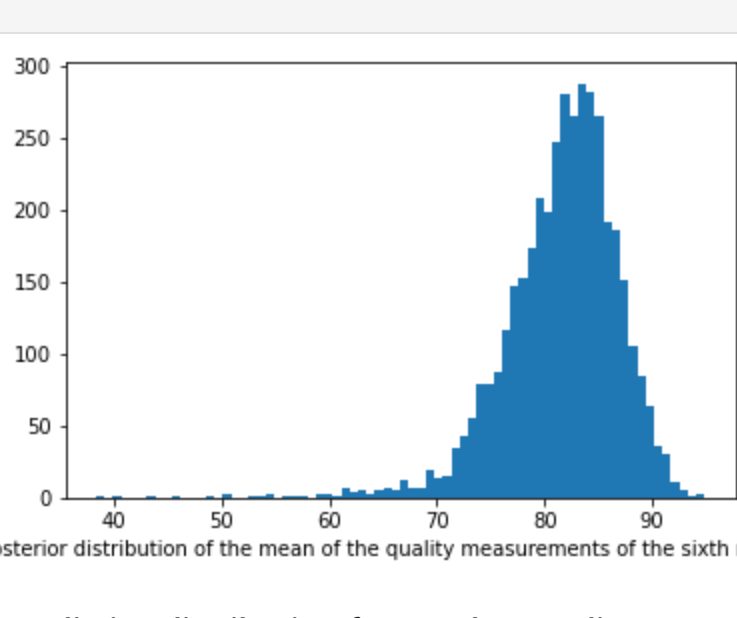
Exercise 2 C

i) posterior distribution of the mean of the quality measurements of the sixth machine

In [128…

```
last_machine_mu = fit.extract("mu")

plt.hist(last_machine_mu['mu'], bins='auto')
plt.xlabel('posterior distribution of the mean of the quality measurements of the sixt
plt.show()
```

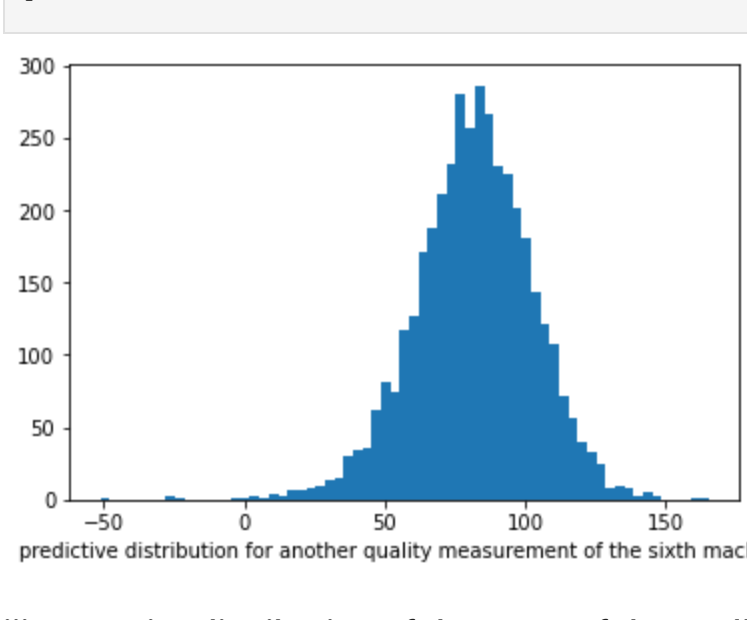


ii) predictive distribution for another quality measurement of the sixth machine

In [129…

```
last_machine_pred = fit.extract("ypred")

plt.hist(last_machine_pred['ypred'], bins='auto')
plt.xlabel('predictive distribution for another quality measurement of the sixth mach
plt.show()
```

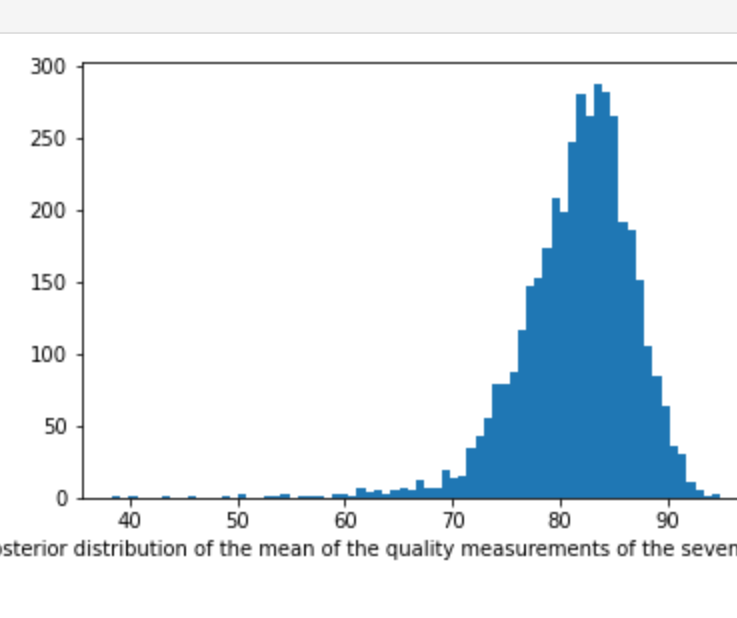


iii) posterior distribution of the mean of the quality measurements of the seventh machine

In [130…

```
last_machine_mu = fit.extract("mu")

plt.hist(last_machine_mu['mu'], bins='auto')
plt.xlabel('posterior distribution of the mean of the quality measurements of the seve
plt.show()
```



Exercise 2 D

In [80]:

```
pooled_model = '''
data {
    int<lower=0> N;
    int<lower=0> J;
    matrix[N, J] y;
}

parameters {
    real mu;
    real<lower=0> sigma;
}


model {
    // priors
    mu ~ normal(0, 10);
    sigma ~ gamma(1,1);

    // likelihood
    for (j in 1:J)
        y[,j] ~ normal(mu, sigma);
}

generated quantities {
    real ypred;

    ypred = normal_rng(mu, sigma);
}
'''
```

In [81]:

```
data = {
    "N": len(quality),
    "J": len(quality[0]),
    "y": quality
}

posterior = pystan.StanModel(model_code=pooled_model)
fit = posterior.sampling(data=data)
print(fit)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_d8c618b9387f1eca5cfb93e0c0f24a
72 NOW.
Inference for Stan model: anon_model_d8c618b9387f1eca5cfb93e0c0f24a72.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

        mean se_mean     sd   2.5%    25%    50%    75%  97.5%  n_eff   Rhat
mu      85.5    0.08    3.2   78.7  83.49  85.73  87.65  91.23   1744    1.0
sigma  16.12    0.05   1.85  12.99  14.85  15.93  17.19  20.22   1453    1.0
ypred  85.94    0.28  16.62  52.65  75.16  86.18  97.18 117.71   3509    1.0
lp__  -155.5    0.03   1.07 -158.2 -155.9 -155.1 -154.8 -154.5   1441    1.0

Samples were drawn using NUTS at Sun Oct 23 23:44:13 2022.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

In [83]:

```
data = fit.extract("mu")["mu"]
print(f"The 90% confidence interval for mu is [ {np.quantile(data, 0.05)} - {np.quant
```

```
The 90% confidence interval for mu is [ 79.90168614324715 - 90.43073398301917 ]
```

Hierarchical model

Exercise 2 A

$\mu_P \sim N(0, 1)$

$\sigma_P \sim Inv - \chi^2(10)$

$\theta_j \sim N(\mu_P, \sigma_P^2)$

$\sigma_j \sim Inv - \chi^2(10)$

$y_{ij} \sim N(\theta_j, \sigma_j^2)$

Same reasoning as above still stands.

Exercise 2 B

In [3]:

```
hierarchical_model = '''
data {
    int<lower=0> N;
    int<lower=0> J;
    matrix[N, J] y;
}

parameters {
    real mu;
    real<lower=0> sigma;
    real theta[J];
    real<lower=0> sigmaJ;

    real new_machine;
}


model {
    // priors
    mu ~ normal(0, 10);
    sigma ~ inv_chi_square(1);
    for (j in 1:J)
        theta[j] ~ normal(mu, sigma);
    new_machine ~ normal(mu, sigma);
    sigmaJ ~ inv_chi_square(1);

    // likelihood
    for (j in 1:J){
        y[,j] ~ normal(theta[j], sigmaJ);
        y[,j] ~ normal(new_machine, sigmaJ);
    }
}

generated quantities {
    real ypred[J];
    real yprednew;


    for (j in 1:J) {
        ypred[j] = normal_rng(theta[j] , sigmaJ);
    }

    yprednew = normal_rng(new_machine , sigmaJ);

}
'''
```

In [4]:

```
data = {
    "N": len(quality),
    "J": len(quality[0]),
    "y": quality
}

posterior = pystan.StanModel(model_code=hierarchical_model)
fit = posterior.sampling(data=data)
print(fit)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_2a0e5c85d01a6edcd4652bc1c755fa
a5 NOW.
Inference for Stan model: anon_model_2a0e5c85d01a6edcd4652bc1c755faa5.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

              mean se_mean     sd   2.5%    25%    50%    75%  97.5%  n_eff   Rhat
mu            8.94    0.13  10.61  -11.7   1.96   8.81   16.1  29.78   6277    1.0
sigma        91.62    0.53  32.45  50.04  70.37  85.03 105.08 172.11   3716    1.0
theta[1]     75.44    0.09   7.35  60.99  70.65  75.46  80.22  89.65   6952    1.0
theta[2]    105.35    0.09   7.62  90.46  100.3 105.34 110.27 120.73   7092    1.0
theta[3]     87.15    0.08   7.33  72.24  82.53  87.18  91.97 101.81   8577    1.0
theta[4]    110.79     0.1   7.75   95.2 105.63 110.87 116.16 126.03   6581    1.0
theta[5]     89.31    0.08    7.4   75.0   84.2  89.38  94.25 103.85   7705    1.0
theta[6]     85.28    0.09   7.48  70.34  80.33  85.31  90.18  99.69   6763    1.0
sigmaJ       16.64    0.02   1.68  13.74  15.47  16.51  17.66  20.36   4890    1.0
new_machine  92.78    0.03    3.0  87.05  90.79  92.76  94.78  98.65   7707    1.0
ypred[1]     75.35    0.29  18.37  38.88  62.99  75.11   87.9 110.97   4069    1.0
ypred[2]     105.4    0.28  18.21  69.42  93.33 105.83 117.53 141.35   4238    1.0
ypred[3]     87.24    0.29  18.18  51.59  75.36  86.86  99.63 123.51   4002    1.0
ypred[4]     110.5    0.29  18.22  73.38  98.63  110.8 122.67 146.43   3988    1.0
ypred[5]     89.19    0.32  18.07  54.37  76.63  88.71 101.37 125.25   3287    1.0
ypred[6]     85.69    0.28  18.42  49.32  73.65  85.72  97.91 122.22   4256    1.0
yprednew     92.31    0.27   16.8  59.46  80.74  92.84 103.57 124.58   3895    1.0
lp__        -238.2    0.06   2.38 -243.8 -239.5 -237.8 -236.5 -234.5   1628    1.0

Samples were drawn using NUTS at Sun Oct 30 13:42:00 2022.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

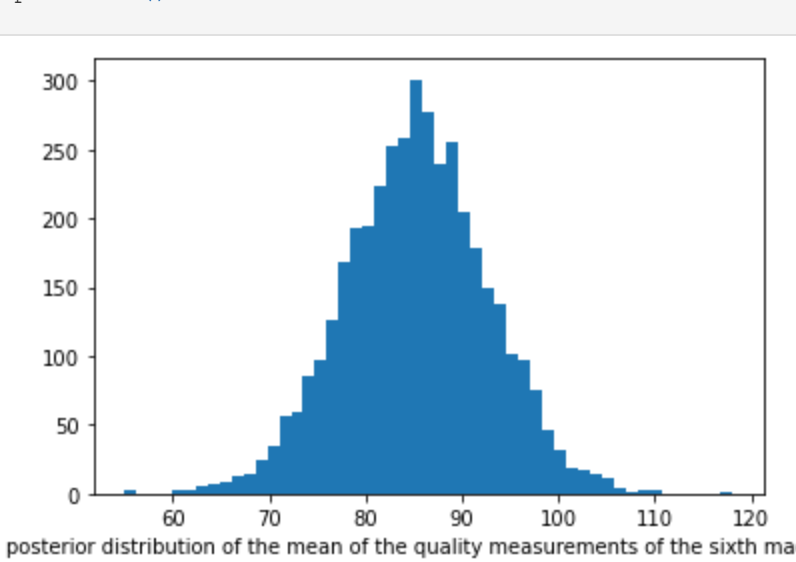
Exercise 2 C

i) posterior distribution of the mean of the quality measurements of the sixth machine

In [5]:

```
last_machine_mu = fit.extract("theta")

plt.hist(last_machine_mu['theta'][:,5], bins='auto')
plt.xlabel('posterior distribution of the mean of the quality measurements of the sixt
plt.show()
```

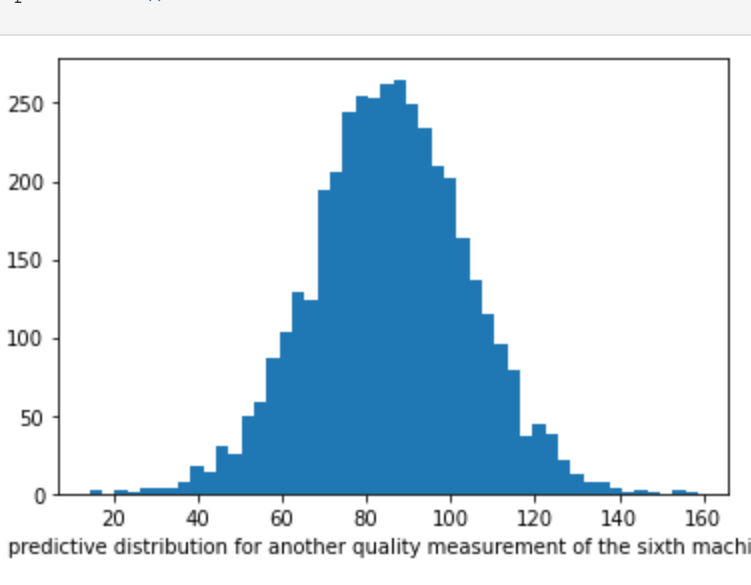


ii) predictive distribution for another quality measurement of the sixth machine

In [6]:

```
last_machine_pred = fit.extract("ypred")

plt.hist(last_machine_pred['ypred'][:,5], bins='auto')
plt.xlabel('predictive distribution for another quality measurement of the sixth mach:
plt.show()
```

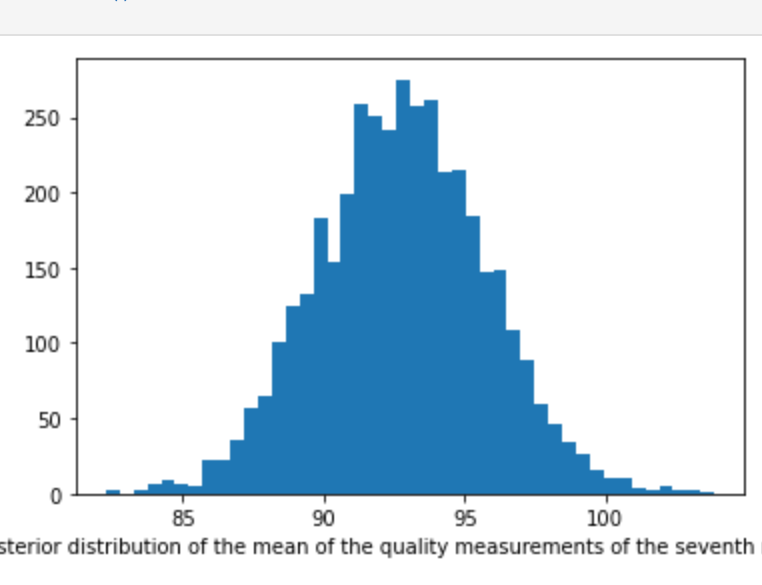


iii) posterior distribution of the mean of the quality measurements of the seventh machine

In [7]:

```
last_machine_mu = fit.extract("new_machine")

plt.hist(last_machine_mu['new_machine'], bins='auto')
plt.xlabel('posterior distribution of the mean of the quality measurements of the seve
plt.show()
```



Exercise 2 D

In [8]:

```
hierarchical_model = '''
data {
    int<lower=0> N;
    int<lower=0> J;
    matrix[N, J] y;
}

parameters {
    real mu;
    real<lower=0> sigma;
    real theta[J];
    real<lower=0> sigmaJ;

    real new_machine;
}


model {
    // priors
    mu ~ normal(0, 10);
    sigma ~ gamma(1,1);
    for (j in 1:J)
        theta[j] ~ normal(mu, sigma);
    new_machine ~ normal(mu, sigma);
    sigmaJ ~ gamma(1,1);

    // likelihood
    for (j in 1:J){
        y[,j] ~ normal(theta[j], sigmaJ);
        y[,j] ~ normal(new_machine, sigmaJ);
    }
}

generated quantities {
    real ypred[J];
    real yprednew;


    for (j in 1:J) {
        ypred[j] = normal_rng(theta[j] , sigmaJ);
    }

    yprednew = normal_rng(new_machine , sigmaJ);

}
'''
```

In [9]:

```
data = {
    "N": len(quality),
    "J": len(quality[0]),
    "y": quality
}

posterior = pystan.StanModel(model_code=hierarchical_model)
fit = posterior.sampling(data=data)
print(fit)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_4502017bf0c46e856174a137a920e5
9f NOW.
WARNING:pystan:305 of 4000 iterations ended with a divergence (7.62 %).
WARNING:pystan:Try running with adapt_delta larger than 0.8 to remove the divergences.
Inference for Stan model: anon_model_4502017bf0c46e856174a137a920e59f.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

              mean se_mean     sd   2.5%    25%    50%    75%  97.5%  n_eff   Rhat
mu           80.15    0.45   9.52  54.96  76.12  83.07  86.93  91.33    453   1.01
sigma         7.69     0.3   5.23   1.16   3.59   6.54  10.67  20.66    300   1.01
theta[1]     80.26    0.36   6.86  66.01   75.7  81.06   85.6  91.13    361    1.0
theta[2]     94.47    0.27   6.33  83.98   89.7  93.74  98.74 107.87    561    1.0
theta[3]     85.84    0.12   5.13  74.86  82.74  86.18  89.24  95.17   1686    1.0
theta[4]      96.8    0.37   7.42  85.08  90.87  96.02 102.08 112.38    397    1.0
theta[5]     87.15    0.11   4.95  77.04  84.19  87.32  90.19  96.91   2144    1.0
theta[6]      84.9    0.14   5.24  73.33  81.76  85.33  88.39  94.11   1344    1.0
sigmaJ       15.52    0.05   1.42  12.99  14.53  15.43  16.43  18.41    697    1.0
new_machine  91.05    0.13   2.89  85.34  89.09  91.09  93.01  96.76    532    1.0
ypred[1]     80.13    0.37  16.82   47.5   68.8  80.04  91.33 113.76   2054    1.0
ypred[2]     94.24    0.32  16.87   61.2  83.16  94.18 105.32 127.05   2764    1.0
ypred[3]     86.09    0.28  16.57   53.6  75.28  86.52   96.9  118.4   3556    1.0
ypred[4]     96.98    0.47  17.43  63.23  85.14  97.28 108.87 131.67   1373    1.0
ypred[5]      87.1    0.26   16.2   54.7  76.08  87.73  97.66 117.84   4031    1.0
ypred[6]     84.76    0.27  16.66  52.54  73.72  84.66  95.97 117.56   3679    1.0
yprednew     91.04    0.26  16.22  58.49  80.39  91.13 101.78 122.83   3998    1.0
lp__        -272.1     0.3   3.86 -279.1 -274.7 -272.5 -270.0 -263.1    168   1.01

Samples were drawn using NUTS at Sun Oct 30 13:43:16 2022.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

In [10]:

```
data = fit.extract("theta[1]")["theta[1]"]
print(f"The 90% confidence interval for mu is [ {np.quantile(data, 0.05)} - {np.quant
```

```
The 90% confidence interval for mu is [ 68.05223932058905 - 90.03666160987042 ]
```