Homework 6

Garrett Carr

Welcome to homework 6!

For this homework, we want you to use the VO2.dat dataset to predict MaxVO2ML (maximum squared milliliters of air in lungs) based on the other covariates in the dataset. The idea is simple here. Find the "best" model.

Finding the "best" model is obviously subjective. We define the "best" model here to excel in both information criterion and interpretability. In identifying this model, use one of the probabilistic programming languages (PPLs), such as JAGS or Stan, that we have been using in class to build the model.

When you have decided on a model that you feel is justified, report all useful convergence diagnostics and information criterion. After deciding on a model, REPLICATE this model in both JAGS and Stan. It is expected that you will have attempted multiple different models, but only report statistics for a single model (using both languages).

For the grading of this assignment, we expect you to explain your reasoning and provide interpretation for the model you choose.

There is added incentive to this assignment. The top 2 students with lowest reported WAIC plus a complexity penalty will each be awarded 2 bonus points to their assignment. The catch is that the WAIC will be penalized for complexity (number of terms). For the penalization, each individual covariate in the model will receive 1 point of penalty, and interactions will receive 2 penalty points. A model has to be simple and effective.

```
library(cmdstanr)
```

```
## This is cmdstanr version 0.4.0

## - Online documentation and vignettes at mc-stan.org/cmdstanr

## - CmdStan path set to: C:/Users/fyref/Documents/.cmdstanr/cmdstan-2.29.0

## - Use set_cmdstan_path() to change the path
```

```
library(rstan)
```

```
## Loading required package: StanHeaders

## Loading required package: ggplot2

## rstan (Version 2.21.3, GitRev: 2e1f913d3ca3)

## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
```

```
## Do not specify '-march=native' in 'LOCAL_CPPFLAGS' or a Makevars file
```

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v tibble  3.1.6     v dplyr   1.0.8
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
## v purrr   0.3.4

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x tidyr::extract() masks rstan::extract()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
```

```
library(posterior)
```

```
## This is posterior version 1.2.0

##
## Attaching package: 'posterior'

## The following objects are masked from 'package:rstan':
##
##     ess_bulk, ess_tail

## The following objects are masked from 'package:stats':
##
##     mad, sd, var
```

```
library(bayesplot)
```

```
## This is bayesplot version 1.8.1

## - Online documentation and vignettes at mc-stan.org/bayesplot

## - bayesplot theme set to bayesplot::theme_default()

##    * Does _not_ affect other ggplot2 plots

##    * See ?bayesplot_theme_set for details on theme setting

##
## Attaching package: 'bayesplot'

## The following object is masked from 'package:posterior':
##
##     rhat
```

```
library(R2jags)
```

```
## Loading required package: rjags
```

```
## Loading required package: coda
```

```
##
## Attaching package: 'coda'
```

```
## The following object is masked from 'package:rstan':
##
##     traceplot
```

```
## Linked to JAGS 4.3.0
```

```
## Loaded modules: basemod,bugs
```

```
##
## Attaching package: 'R2jags'
```

```
## The following object is masked from 'package:coda':
##
##     traceplot
```

```
## The following object is masked from 'package:rstan':
##
##     traceplot
```

```
# Read in Data
dat <- read.table('vo2.dat', header = TRUE) %>%
  as_tibble()

# Create data list
data_list <- list(
  N = count(dat)[[1]],
  y = dat$MaxVO2ML,
  gender = dat$Gender,
  age = dat$Age1,
  bmi = dat$BMI,
  hr = dat$HR,
  rpe = dat$RPE
)

rstan_options(auto_write = TRUE)
```

```
# Inital stan model

mod <- stan_model(file='vo2.stan')

fit <- sampling(mod, data_list, iter = 9000, warmup=1000, thin = 2, chains = 4)
```

```
##
## SAMPLING FOR MODEL 'vo2' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 9000 [  0%]  (Warmup)
## Chain 1: Iteration:  900 / 9000 [ 10%]  (Warmup)
## Chain 1: Iteration: 1001 / 9000 [ 11%]  (Sampling)
## Chain 1: Iteration: 1900 / 9000 [ 21%]  (Sampling)
## Chain 1: Iteration: 2800 / 9000 [ 31%]  (Sampling)
## Chain 1: Iteration: 3700 / 9000 [ 41%]  (Sampling)
## Chain 1: Iteration: 4600 / 9000 [ 51%]  (Sampling)
## Chain 1: Iteration: 5500 / 9000 [ 61%]  (Sampling)
## Chain 1: Iteration: 6400 / 9000 [ 71%]  (Sampling)
## Chain 1: Iteration: 7300 / 9000 [ 81%]  (Sampling)
## Chain 1: Iteration: 8200 / 9000 [ 91%]  (Sampling)
## Chain 1: Iteration: 9000 / 9000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 7.154 seconds (Warm-up)
## Chain 1:                45.224 seconds (Sampling)
## Chain 1:                52.378 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'vo2' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 9000 [  0%]  (Warmup)
## Chain 2: Iteration:  900 / 9000 [ 10%]  (Warmup)
## Chain 2: Iteration: 1001 / 9000 [ 11%]  (Sampling)
## Chain 2: Iteration: 1900 / 9000 [ 21%]  (Sampling)
## Chain 2: Iteration: 2800 / 9000 [ 31%]  (Sampling)
## Chain 2: Iteration: 3700 / 9000 [ 41%]  (Sampling)
## Chain 2: Iteration: 4600 / 9000 [ 51%]  (Sampling)
## Chain 2: Iteration: 5500 / 9000 [ 61%]  (Sampling)
## Chain 2: Iteration: 6400 / 9000 [ 71%]  (Sampling)
## Chain 2: Iteration: 7300 / 9000 [ 81%]  (Sampling)
## Chain 2: Iteration: 8200 / 9000 [ 91%]  (Sampling)
## Chain 2: Iteration: 9000 / 9000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 6.432 seconds (Warm-up)
## Chain 2:                47.132 seconds (Sampling)
## Chain 2:                53.564 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'vo2' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0 seconds
```

```
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 9000 [  0%]  (Warmup)
## Chain 3: Iteration:  900 / 9000 [ 10%]  (Warmup)
## Chain 3: Iteration: 1001 / 9000 [ 11%]  (Sampling)
## Chain 3: Iteration: 1900 / 9000 [ 21%]  (Sampling)
## Chain 3: Iteration: 2800 / 9000 [ 31%]  (Sampling)
## Chain 3: Iteration: 3700 / 9000 [ 41%]  (Sampling)
## Chain 3: Iteration: 4600 / 9000 [ 51%]  (Sampling)
## Chain 3: Iteration: 5500 / 9000 [ 61%]  (Sampling)
## Chain 3: Iteration: 6400 / 9000 [ 71%]  (Sampling)
## Chain 3: Iteration: 7300 / 9000 [ 81%]  (Sampling)
## Chain 3: Iteration: 8200 / 9000 [ 91%]  (Sampling)
## Chain 3: Iteration: 9000 / 9000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 5.068 seconds (Warm-up)
## Chain 3:                46.74 seconds (Sampling)
## Chain 3:                51.808 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'vo2' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 9000 [  0%]  (Warmup)
## Chain 4: Iteration:  900 / 9000 [ 10%]  (Warmup)
## Chain 4: Iteration: 1001 / 9000 [ 11%]  (Sampling)
## Chain 4: Iteration: 1900 / 9000 [ 21%]  (Sampling)
## Chain 4: Iteration: 2800 / 9000 [ 31%]  (Sampling)
## Chain 4: Iteration: 3700 / 9000 [ 41%]  (Sampling)
## Chain 4: Iteration: 4600 / 9000 [ 51%]  (Sampling)
## Chain 4: Iteration: 5500 / 9000 [ 61%]  (Sampling)
## Chain 4: Iteration: 6400 / 9000 [ 71%]  (Sampling)
## Chain 4: Iteration: 7300 / 9000 [ 81%]  (Sampling)
## Chain 4: Iteration: 8200 / 9000 [ 91%]  (Sampling)
## Chain 4: Iteration: 9000 / 9000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 7.516 seconds (Warm-up)
## Chain 4:                42.958 seconds (Sampling)
## Chain 4:                50.474 seconds (Total)
## Chain 4:
```

```
fit2 <- stan(file='vo2.stan', data = data_list, iter = 15000, warmup = 3000, chains = 4, thin = 1)
```

```
##
## SAMPLING FOR MODEL 'vo2' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
```

```
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:     1 / 15000 [  0%]  (Warmup)
## Chain 1: Iteration:  1500 / 15000 [ 10%]  (Warmup)
## Chain 1: Iteration:  3000 / 15000 [ 20%]  (Warmup)
## Chain 1: Iteration:  3001 / 15000 [ 20%]  (Sampling)
## Chain 1: Iteration:  4500 / 15000 [ 30%]  (Sampling)
## Chain 1: Iteration:  6000 / 15000 [ 40%]  (Sampling)
## Chain 1: Iteration:  7500 / 15000 [ 50%]  (Sampling)
## Chain 1: Iteration:  9000 / 15000 [ 60%]  (Sampling)
## Chain 1: Iteration: 10500 / 15000 [ 70%]  (Sampling)
## Chain 1: Iteration: 12000 / 15000 [ 80%]  (Sampling)
## Chain 1: Iteration: 13500 / 15000 [ 90%]  (Sampling)
## Chain 1: Iteration: 15000 / 15000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 14.094 seconds (Warm-up)
## Chain 1:                68.166 seconds (Sampling)
## Chain 1:                82.26 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'vo2' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:     1 / 15000 [  0%]  (Warmup)
## Chain 2: Iteration:  1500 / 15000 [ 10%]  (Warmup)
## Chain 2: Iteration:  3000 / 15000 [ 20%]  (Warmup)
## Chain 2: Iteration:  3001 / 15000 [ 20%]  (Sampling)
## Chain 2: Iteration:  4500 / 15000 [ 30%]  (Sampling)
## Chain 2: Iteration:  6000 / 15000 [ 40%]  (Sampling)
## Chain 2: Iteration:  7500 / 15000 [ 50%]  (Sampling)
## Chain 2: Iteration:  9000 / 15000 [ 60%]  (Sampling)
## Chain 2: Iteration: 10500 / 15000 [ 70%]  (Sampling)
## Chain 2: Iteration: 12000 / 15000 [ 80%]  (Sampling)
## Chain 2: Iteration: 13500 / 15000 [ 90%]  (Sampling)
## Chain 2: Iteration: 15000 / 15000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 14.958 seconds (Warm-up)
## Chain 2:                75.558 seconds (Sampling)
## Chain 2:                90.516 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'vo2' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:     1 / 15000 [  0%]  (Warmup)
```

```
## Chain 3: Iteration:  1500 / 15000 [ 10%]  (Warmup)
## Chain 3: Iteration:  3000 / 15000 [ 20%]  (Warmup)
## Chain 3: Iteration:  3001 / 15000 [ 20%]  (Sampling)
## Chain 3: Iteration:  4500 / 15000 [ 30%]  (Sampling)
## Chain 3: Iteration:  6000 / 15000 [ 40%]  (Sampling)
## Chain 3: Iteration:  7500 / 15000 [ 50%]  (Sampling)
## Chain 3: Iteration:  9000 / 15000 [ 60%]  (Sampling)
## Chain 3: Iteration: 10500 / 15000 [ 70%]  (Sampling)
## Chain 3: Iteration: 12000 / 15000 [ 80%]  (Sampling)
## Chain 3: Iteration: 13500 / 15000 [ 90%]  (Sampling)
## Chain 3: Iteration: 15000 / 15000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 15.562 seconds (Warm-up)
## Chain 3:                71.218 seconds (Sampling)
## Chain 3:                86.78 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'vo2' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:     1 / 15000 [  0%]  (Warmup)
## Chain 4: Iteration:  1500 / 15000 [ 10%]  (Warmup)
## Chain 4: Iteration:  3000 / 15000 [ 20%]  (Warmup)
## Chain 4: Iteration:  3001 / 15000 [ 20%]  (Sampling)
## Chain 4: Iteration:  4500 / 15000 [ 30%]  (Sampling)
## Chain 4: Iteration:  6000 / 15000 [ 40%]  (Sampling)
## Chain 4: Iteration:  7500 / 15000 [ 50%]  (Sampling)
## Chain 4: Iteration:  9000 / 15000 [ 60%]  (Sampling)
## Chain 4: Iteration: 10500 / 15000 [ 70%]  (Sampling)
## Chain 4: Iteration: 12000 / 15000 [ 80%]  (Sampling)
## Chain 4: Iteration: 13500 / 15000 [ 90%]  (Sampling)
## Chain 4: Iteration: 15000 / 15000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 15.986 seconds (Warm-up)
## Chain 4:                75.329 seconds (Sampling)
## Chain 4:                91.315 seconds (Total)
## Chain 4:
```

```
LLa <- as.array(fit2,pars='log_lik')
library(loo) # Leave One out cross validation for waic
```

```
## This is loo version 2.4.1

## - Online documentation and vignettes at mc-stan.org/loo

## - As of v2.0.0 loo defaults to 1 core but we recommend using as many as possible. Use the 'cores' ar

## - Windows 10 users: loo may be very slow if 'mc.cores' is set in your .Rprofile file (see https://gi
```

```
##
## Attaching package: 'loo'

## The following object is masked from 'package:rstan':
##
##     loo
```

```
waic(LLa)
```

```
## Warning:
## 6 (5.0%) p_waic estimates greater than 0.4. We recommend trying loo instead.

##
## Computed from 48000 by 120 log-likelihood matrix
##
##           Estimate   SE
## elpd_waic   -396.5  8.1
## p_waic         7.5  1.3
## waic         792.9 16.3
##
## 6 (5.0%) p_waic estimates greater than 0.4. We recommend trying loo instead.
```

```
loo2 <- loo(fit2,pars='log_lik')
loo2
```

```
##
## Computed from 48000 by 120 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -396.5  8.1
## p_loo         7.6  1.3
## looic       793.0 16.3
## ------
## Monte Carlo SE of elpd_loo is 0.0.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

```
pareto_k_ids(loo2)
```

```
## integer(0)
```

```
pareto_k_values(loo2)
```

```
##   [1]  0.0828027698  0.1141375492  0.0300381439  0.1362535327  0.1243324203
##   [6]  0.0437454017  0.0915306777  0.2586974654  0.0734573029  0.0779564305
##  [11]  0.0465962032  0.0901418677  0.1123171409  0.0622977487  0.0758166299
##  [16]  0.0947566148  0.0452363650  0.0661646573  0.0739200441  0.1961273655
##  [21]  0.1107353121  0.0216717203 -0.0001331568  0.0315590042  0.0608593170
##  [26]  0.1526645581  0.0604669909 -0.0031660979 -0.0538385350  0.1286874198
##  [31]  0.0056752146  0.0278656958  0.0252917560  0.0799887872 -0.0595488172
```

```
##  [36]  0.2100136100  0.1040133971  0.0776977292 -0.0316165522  0.1749772818
##  [41]  0.0822841621  0.0803784744  0.0936001295  0.0055387236  0.0817186998
##  [46]  0.0261152874  0.2344858057 -0.0032786650  0.0590189315  0.0507344494
##  [51]  0.0783095311  0.1120988918  0.1348813008  0.0216268040  0.0891743213
##  [56]  0.1578823280  0.0573209155  0.0302983235  0.0753656447  0.0586981725
##  [61]  0.0593895100 -0.0091002423  0.1112752496  0.0837999597  0.2284833317
##  [66]  0.0560590357  0.0632166761  0.2209955879  0.0654696894  0.0391208550
##  [71]  0.0394960002  0.3313530539 -0.0614496503  0.1995928171  0.0553328037
##  [76]  0.1078285438  0.0491834958  0.0794344543  0.0201232676 -0.0094052836
##  [81]  0.0447863585  0.0092736786  0.0753738998 -0.0073379564  0.1020267008
##  [86]  0.0793146746  0.0535320372  0.0720334684  0.1546555907  0.0406154321
##  [91]  0.1274556805  0.0730926097  0.2212872095  0.2685658220  0.1104777281
##  [96] -0.0618183748 -0.0310027097 -0.0653331366  0.1072441933 -0.0070510937
## [101]  0.0591459445  0.0001539937  0.0654032416  0.0531041088  0.0007626324
## [106]  0.0370247815  0.0813069773  0.0478096082  0.0450020855 -0.0058593342
## [111]  0.0215173196 -0.0143488141  0.0433377068  0.2012254370 -0.0127589446
## [116]  0.3099157167  0.0394663977  0.0881158848  0.0626875732 -0.0500818922
```

```
pareto_k_influence_values(loo2)
```

```
##   [1]  0.0828027698  0.1141375492  0.0300381439  0.1362535327  0.1243324203
##   [6]  0.0437454017  0.0915306777  0.2586974654  0.0734573029  0.0779564305
##  [11]  0.0465962032  0.0901418677  0.1123171409  0.0622977487  0.0758166299
##  [16]  0.0947566148  0.0452363650  0.0661646573  0.0739200441  0.1961273655
##  [21]  0.1107353121  0.0216717203 -0.0001331568  0.0315590042  0.0608593170
##  [26]  0.1526645581  0.0604669909 -0.0031660979 -0.0538385350  0.1286874198
##  [31]  0.0056752146  0.0278656958  0.0252917560  0.0799887872 -0.0595488172
##  [36]  0.2100136100  0.1040133971  0.0776977292 -0.0316165522  0.1749772818
##  [41]  0.0822841621  0.0803784744  0.0936001295  0.0055387236  0.0817186998
##  [46]  0.0261152874  0.2344858057 -0.0032786650  0.0590189315  0.0507344494
##  [51]  0.0783095311  0.1120988918  0.1348813008  0.0216268040  0.0891743213
##  [56]  0.1578823280  0.0573209155  0.0302983235  0.0753656447  0.0586981725
##  [61]  0.0593895100 -0.0091002423  0.1112752496  0.0837999597  0.2284833317
##  [66]  0.0560590357  0.0632166761  0.2209955879  0.0654696894  0.0391208550
##  [71]  0.0394960002  0.3313530539 -0.0614496503  0.1995928171  0.0553328037
##  [76]  0.1078285438  0.0491834958  0.0794344543  0.0201232676 -0.0094052836
##  [81]  0.0447863585  0.0092736786  0.0753738998 -0.0073379564  0.1020267008
##  [86]  0.0793146746  0.0535320372  0.0720334684  0.1546555907  0.0406154321
##  [91]  0.1274556805  0.0730926097  0.2212872095  0.2685658220  0.1104777281
##  [96] -0.0618183748 -0.0310027097 -0.0653331366  0.1072441933 -0.0070510937
## [101]  0.0591459445  0.0001539937  0.0654032416  0.0531041088  0.0007626324
## [106]  0.0370247815  0.0813069773  0.0478096082  0.0450020855 -0.0058593342
## [111]  0.0215173196 -0.0143488141  0.0433377068  0.2012254370 -0.0127589446
## [116]  0.3099157167  0.0394663977  0.0881158848  0.0626875732 -0.0500818922
```

```
pareto_k_table(loo2)
```

```
##
## All Pareto k estimates are good (k < 0.5).
```

```
mdl <- "
  model {
```

```
    for (i in 1:120) {
      y[i] ~ dnorm(mu[i], 1/vr)
      mu[i] <- b0 + bage*age[i] + bgen*gen[i] + bbmi*bmi[i] + bhr*hr[i] + brpe*rpe[i]
    }
    b0 ~ dnorm(20,10)
    bage ~ dnorm(0,10)
    bgen ~ dnorm(0,10)
    bbmi ~ dnorm(0,10)
    bhr ~ dnorm(0,10)
    brpe ~ dnorm(0,10)
    vr ~ dgamma(4,.25)
  }
"

writeLines(mdl, 'vo2reg.txt')


y <- dat$MaxVO2ML
age <- dat$Age1
gen <- dat$Gender
bmi <- dat$BMI
hr <- dat$HR
rpe <- dat$RPE

data.jags <- c('y', 'age', 'gen', 'bmi', 'hr', 'rpe')
parms <- c('b0', 'bage', 'bgen', 'bbmi', 'bhr', 'brpe', 'vr')

vo2reg.sim <- jags(data=data.jags, inits = NULL, parameters.to.save = parms,
                   model.file = 'vo2reg.txt', n.iter = 10000, n.burnin = 1000,
                   n.chains = 4, n.thin = 2)
```

```
## module glm loaded

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 120
##    Unobserved stochastic nodes: 7
##    Total graph size: 1034
##
## Initializing model
```

```
vo2reg.sim
```

```
## Inference for Bugs model at "vo2reg.txt", fit using jags,
##  4 chains, each with 10000 iterations (first 1000 discarded), n.thin = 2
##  n.sims = 18000 iterations saved
##          mu.vect sd.vect    2.5%     25%     50%     75%   97.5%  Rhat n.eff
## b0        20.057   0.317  19.444  19.842  20.061  20.272  20.676 1.001 18000
## bage       0.834   0.250   0.340   0.665   0.836   1.003   1.320 1.001 18000
## bbmi      -0.473   0.171  -0.802  -0.589  -0.475  -0.359  -0.134 1.001  6000
```

```
## bgen       0.482   0.315  -0.135   0.272   0.480   0.696   1.097 1.001 18000
## bhr        0.152   0.031   0.092   0.132   0.152   0.173   0.213 1.002  3600
## brpe       0.098   0.251  -0.395  -0.071   0.096   0.267   0.592 1.001  8600
## vr        67.988   7.531  54.459  62.703  67.577  72.763  83.934 1.001 18000
## deviance 872.169   6.005 861.418 867.999 871.802 875.920 884.960 1.001 18000
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 18.0 and DIC = 890.2
## DIC is an estimate of expected predictive error (lower deviance is better).
```

It doesn't really seem like these models are too accurate. It would help if we had more informative priors. It also seems like these models should have different intercepts, depending on some of their groups. Perhaps it's related to the gender.

I'm going to throw out the heart rate and respiration rate, as those are likely correlated with each other and seem to be highly variable anyways.

```
mdl2 <- "
  model {

    for (i in 1:120) {
      y[i] ~ dnorm(mu[i], 1/vr)
      mu[i] <- b0 + bage*age[i] + bgen*gen[i] + bbmi*bmi[i] + bmi[i]*gen[i]
    }
    b0 ~ dnorm(20,10)
    bage ~ dnorm(0,10)
    bgen ~ dnorm(0,10)
    bbmi ~ dnorm(0,10)
    vr ~ dgamma(4,.25)
  }
"

writeLines(mdl2, 'vo2reg2.txt')


y <- dat$MaxVO2ML
age <- dat$Age1
gen <- dat$Gender
bmi <- dat$BMI
hr <- dat$HR
rpe <- dat$RPE

data.jags <- c('y', 'age', 'gen', 'bmi')
parms <- c('b0', 'bage', 'bgen', 'bbmi', 'vr')

vo2reg2.sim <- jags(data=data.jags, inits = NULL, parameters.to.save = parms,
                    model.file = 'vo2reg2.txt', n.iter = 10000, n.burnin = 2000,
                    n.chains = 4, n.thin = 2)
```

```
## Compiling model graph
##    Resolving undeclared variables
```

```
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 120
##    Unobserved stochastic nodes: 5
##    Total graph size: 857
##
## Initializing model
```

```
vo2reg2.sim
```

```
## Inference for Bugs model at "vo2reg2.txt", fit using jags,
##  4 chains, each with 10000 iterations (first 2000 discarded), n.thin = 2
##  n.sims = 16000 iterations saved
##          mu.vect sd.vect    2.5%     25%     50%     75%   97.5%  Rhat n.eff
## b0        20.115   0.318  19.485  19.901  20.115  20.329  20.744 1.001 16000
## bage       1.330   0.211   0.913   1.188   1.331   1.473   1.739 1.001  7900
## bbmi      -0.096   0.145  -0.378  -0.194  -0.098   0.002   0.186 1.001 15000
## bgen      -0.316   0.311  -0.923  -0.525  -0.315  -0.106   0.292 1.001 16000
## vr        78.878   8.548  63.648  72.905  78.327  84.377  96.999 1.001 16000
## deviance 895.483   6.458 884.135 890.879 895.086 899.543 909.394 1.001 16000
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 20.9 and DIC = 916.3
## DIC is an estimate of expected predictive error (lower deviance is better).
```

```
sims <- as.mcmc(vo2reg2.sim)
gelman.diag(sims)
```

```
## Potential scale reduction factors:
##
##          Point est. Upper C.I.
## b0                1          1
## bage              1          1
## bbmi              1          1
## bgen              1          1
## deviance          1          1
## vr                1          1
##
## Multivariate psrf
##
## 1
```

```
chains <- as.matrix(sims)
sims <- as.mcmc(chains)
raftery.diag(sims)
```

```
##
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
```

```
## Probability (s) = 0.95
##
##           Burn-in  Total Lower bound  Dependence
##           (M)      (N)   (Nmin)       factor (I)
##  b0       2        3710  3746         0.990
##  bage     2        3653  3746         0.975
##  bbmi     2        3729  3746         0.995
##  bgen     2        3865  3746         1.030
##  deviance 2        3826  3746         1.020
##  vr       3        4133  3746         1.100
```

```
effectiveSize(sims)
```

```
##       b0     bage     bbmi     bgen deviance       vr
## 16719.91 15485.07 15621.92 15433.54 16000.00 12850.86
```

```
autocorr.diag(sims)
```

```
##                  b0          bage         bbmi         bgen     deviance
## Lag 0   1.000000000  1.0000000000  1.000000000  1.000000000  1.000000000
## Lag 1   0.011695625  0.0163236062  0.011925161  0.017989486 -0.008130909
## Lag 5  -0.005301557  0.0004533487  0.001593546  0.002073673 -0.002972717
## Lag 10 -0.001789713 -0.0174094961 -0.015889802 -0.004461893 -0.009129342
## Lag 50 -0.006928626 -0.0159224211 -0.015413829  0.001116603 -0.014668778
##                  vr
## Lag 0   1.000000000
## Lag 1   0.109121614
## Lag 5  -0.011835518
## Lag 10 -0.002355868
## Lag 50 -0.003053943
```

```
geweke.diag(sims)
```

```
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##       b0     bage     bbmi     bgen deviance       vr
##  -0.6962  -1.8279   1.4774   0.7637   1.1129   1.1461
```

```
samples.m2 <- jags.samples(vo2reg2.sim$model,
                           c("WAIC", "deviance"),
                           type = 'mean',
                          n.iter = 10000,
                          n.burnin = 2000,
                          n.chains = 4,
                          n.thin = 2)

samples.m2$p_waic <- samples.m2$WAIC
samples.m2$waic <- samples.m2$deviance + samples.m2$p_waic
tmp <- sapply(samples.m2, sum)
```

```r
waic.m2 <- round(c(waic = tmp[['waic']], p_waic = tmp[["p_waic"]]), 1)

waic.m2
```

```
##   waic p_waic
##  899.1    3.6
```