459 Final Project, Joe M, 2025-04-25

Problem 1 Part a Loading the Heart data. It has 462 observations on 10 variables. The response variable indicates whether or not coronary heart disease was present at the time of observation.

```r
library(ncvreg) # loading the package
X = Heart$X; y = Heart$y
df = data.frame(y = y, X = X)
```

Next, let's fit probit and logit bayesian regression models. I decided to use MCMCpack for this part, because the way it generates posterior samples is relatively easy to understand. While rstan is very powerful, it uses the advanced Hamiltonian Monte Carlo method. On the other hand, MCMCpack uses the simpler Gibbs sampling method to generate samples, which I felt more comfortable working with. It also has built-in logit and probit regression functions, which makes it perfect for this task.

For both models, I decided to set the priors as follows, $\beta \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where all the means are zero ($\boldsymbol{\mu} = \mathbf{0}$) and the coefficients are independent with variance 100. $\boldsymbol{\Sigma} = (100)I$. Note that this prior also applies to the intercept. I decided have a normal prior because the distribution covers the entire real number line, and I didn't want to assume that any value for the coefficient was impossible (as motivated by Cromwell's rule). I set a relatively high variance to ensure that the model had ample coverage for most realistic values, and I set the mean to zero because I didn't want to make any assumptions about the relative effects of each features (I'm not well versed in medicine). I decided to use the default MCMC pack settings for everything else, because I thought a single chain of length 11,000 with a burn-in of 1,000 and no thinning was sufficient for this application.

```r
library(MCMCpack) # loading MCMCpack
set.seed(459)
b0 = rep(0,10) # prior mean vector, 9 features and 1 intercept, mean vector has length 10.
B0 = diag(0.01, 10) # prior precision (inverse of variance) matrix; independant variables.
mcmc_logit = MCMClogit(y ~. , data = df, b0 = b0, B0 = B0)
logit_matrix = as.matrix(mcmc_logit) # the draws of the gibbs sampler for the coefficients
coefs_logit = colMeans(logit_matrix)
mcmc_probit = MCMCprobit(y ~. , data = df, b0 = b0, B0 = B0)
probit_matrix = as.matrix(mcmc_probit)
coefs_probit = colMeans(probit_matrix)
coefs_logit = round(coefs_logit,3) # logit coeffs rounded to 3 decimals points
coefs_logit
```

```
## (Intercept)       X.sbp   X.tobacco       X.ldl X.adiposity   X.famhist
##      -6.301       0.007       0.083       0.175       0.018       0.961
##     X.typea   X.obesity   X.alcohol       X.age
##       0.041      -0.065       0.000       0.046
```

```r
print("Quantiles for each variable:")
```

```
## [1] "Quantiles for each variable:"
```

```r
cred_int_logit = apply(logit_matrix, 2, quantile, probs = c(0.025, 0.975))
round(cred_int_logit, 3)
```

```
##       (Intercept)  X.sbp X.tobacco X.ldl X.adiposity X.famhist X.typea
## 2.5%       -8.909 -0.004     0.029 0.057      -0.040     0.531   0.017
## 97.5%      -3.737  0.018     0.140 0.295       0.073     1.476   0.066
##       X.obesity X.alcohol X.age
## 2.5%     -0.154    -0.008  0.02
## 97.5%     0.014     0.009  0.07
```

Here are the coefficients for the logit model. How can we interpret these? Remember that the formula for a logit regression model is: $\log(\frac{p}{p-1}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 ... \beta_n x_n$. From the R documentation, a response of 1 indicates coronary heart disease and a response of 0 indicates no coronary heart disease, so p reflects the probability of coronary heart disease. In a logit model, a positive coefficient for a variable corresponds to a positive increase in the log odds when the variable increases. Here, increases in the features with positive coefficients corresponds to an increase in log odds of having heart disease. So based on the positive coefficients here, we can infer that sbp (systolic blood pressure), tobacco (tobacco consumption), ldl (low-density lipoprotein cholesterol), adiposity (Adipose tissue concentration), famhist (Family history of heart disease (1=Present, 0=Absent)), typea (Score designed to measure type-A behavior), and age are all positively correlated with heart disease (because all these features are positively correlated with the log odds, and the log odds is positively correlated with p, the probability of heart disease). Interestingly, increased obesity seems to be linked with a decrease in the log-odds of heart disease. This is probably a result of multicolinearity because adposity measures a similar thing. Meanwhile the alcohol feature, which corresponds to the subject's current consumption of alcohol, seems to have a neutral impact on the log odds of heart disease, as its mean coefficient is roughly 0, meaning any change in alcohol consumption won't affect the response.

```
coefs_probit = round(coefs_probit,3) # probit coefs rounded to 3 decimals points
coefs_probit
```

```
## (Intercept)       X.sbp   X.tobacco       X.ldl X.adiposity   X.famhist
##      -3.606       0.004       0.050       0.103       0.013       0.549
##     X.typea   X.obesity   X.alcohol       X.age
##       0.024      -0.042       0.000       0.027
```

```
print("Quantiles for each variable:")
```

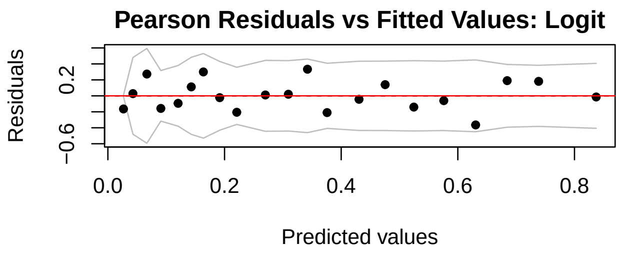
```
## [1] "Quantiles for each variable:"
```

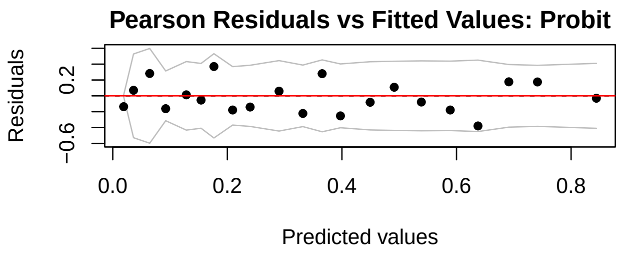
```
cred_int_probit = apply(probit_matrix, 2, quantile, probs = c(0.025, 0.975))
round(cred_int_probit, 3)
```

```
##       (Intercept)  X.sbp X.tobacco X.ldl X.adiposity X.famhist X.typea
## 2.5%       -5.094 -0.003     0.018 0.036      -0.022     0.278   0.010
## 97.5%      -2.190  0.010     0.081 0.173       0.047     0.817   0.038
##       X.obesity X.alcohol X.age
## 2.5%     -0.092    -0.005 0.013
## 97.5%     0.009     0.005 0.041
```

The link function of the probit model is the inverse standard normal CDF, so our model is assuming: $P(Y = 1|X) = \Phi(X^T\beta)$. The normal CDF is monotonically increasing, meaning that a positive coefficient again indicates that an increase in the feature correlates to an increased risk of heart disease. While the magnitudes of the coefficients of the probit model are proportionally lower than those of the logit model (due to different link functions), their signage is the same. I.e, the sbp, tobacco, ldl, adiposity, famhist, typea, and age features still correlate positively with heart disease. Alcohol again has an essentially negligible effect on heart disease risk, while increased obesity correlates with a decrease in heart disease.

```
par(mfrow = c(1, 2), mar = c(4, 4, 2, 1))   # margins: bottom, left, top, right
X_with_intercept = cbind(intercept = 1, X) # creating design matrix
preds = plogis(coefs_logit %*% t(X_with_intercept)) # getting predictions
pearson_resid = (y - preds) / sqrt(preds * (1 - preds)) # calculating pearson residuals
arm::binnedplot(preds, pearson_resid, xlab = "Predicted values", ylab = "Residuals", main = "Pearson Re
abline(h = 0, col = "red")
preds = pnorm(coefs_probit %*% t(X_with_intercept))
pearson_resid = (y - preds) / sqrt(preds * (1 - preds))
arm::binnedplot(preds, pearson_resid, xlab = "Predicted values", ylab = "Residuals", main = "Pearson Re
abline(h = 0, col = "red")
```





This is a binned pearson residuals plot for both models. The pearson part means that the variance of these residuals has been standardized. The binned part means that residuals of similar predicted values have been binned into the points shown above. The trend of the residuals is relatively flat around 0, meaning we've achieved linearity on the link scale and haven't missed any non-linear effects. It also implies that we've chosen appropriate link functions more generally. The variance also looks fairly constant, which is what we would expect from a pearson residual, in which we've divided out the natural variance of the binomial probability parameter. There are no obvious outliers and the binned residuals all appear to be independent, meaning we can be reasonably confident that most of the GLM assumptions have been met.

Part b This part asks us to compute the marginal likelihood of each model so that we can compute the Bayes factor to compare the two models. In order to calculate this, I will be using the bridge_sampler function from the 'bridgesampling' package. Bridge sampling is an MCMC method that draws a "bidge" between the unormalized posterior of some model and a proposal distribution to estimate the marginal likelihood. I will explain more regarding bridge sampling in part 2 of this assignment, where I'll use it again. First, we need to derive the unnormalized log posterior function for the probit and logit models. This is relatively easy because this is the sum of the log likelihood and log prior. We already know the prior because we set it earlier (multivariate normal). The likelihood is just a binomial distribution for both models. Since we only care about the finding a function proportional to the binomial, we can ignore the choose function and take the a sum of the logs of the terms, which we can find by calculating the probabilities associated with our simulated samples.

```r
logit_post = function(par, data){ # logit proportional posterior
  X = model.matrix(y ~., data = df)
  y = df$y
  eta = X %*% par
  log_likelihood = sum(y*plogis(eta, log.p = TRUE) + (1-y)* plogis(eta, lower.tail= FALSE, log.p = TRUE)
  log_prior = mvtnorm::dmvnorm(par, mean = rep(0, 10), sigma = diag(100, 10), log = TRUE)
  return (log_likelihood + log_prior)} # log posterior is sum of log prior and log likelihood
posit_post = function(par, data){ # Same thing but with different probit function
  X = model.matrix(y ~., data = df)
  y = df$y
  eta = X %*% par
  log_likelihood = sum(y*pnorm(eta, log.p = TRUE) + (1-y)* pnorm(eta, lower.tail= FALSE, log.p = TRUE))
  log_prior = mvtnorm::dmvnorm(par, mean = rep(0, 10), sigma = diag(100, 10), log = TRUE)
  return (log_likelihood + log_prior)}
library(bridgesampling)
lb = rep(-Inf, 10) # have to explicitly state bounds for parameters or else function gets mad at me
ub = rep(Inf,10)
names(lb) = names(ub) = colnames(logit_matrix)
bridge_logit = bridge_sampler(samples = logit_matrix, log_posterior = logit_post, data  = df, lb = lb,
bridge_probit = bridge_sampler(samples = probit_matrix, log_posterior = posit_post, data  = df, lb = lb
logml_logit  = bridge_logit$logml
logml_probit  = bridge_probit$logml
bf = exp(logml_logit)/exp(logml_probit) # get the bayes factor by removing the logs from the equation
cat("Bayes factor: ", bf)
```

```
## Bayes factor:  143.926
```

The Bayes factor is a useful metric for comparing two models. It represents the likelihood of Model K over the likelihood of Model L. In this case, Model K is the logit model and Model L is the probit model. The ratio of the marginal likelihood of the logit model over the marginal likelihood of the probit model was found to be 143.926. According to Jeffreys scale, a log likelihood of over 100 indicates decisive evidence in favor of the model in the numerator, which in this case means we have decisive evidence for Model K being better than Model L. In other words, the computed Bayes factor strongly suggests that the logit model is more appropriate than the probit model for this data.

Part c Let's randomly drop a datapoint from the sample, retrain the models, and get a HPD interval for the left-out sample.

```r
set.seed(459)
rando_index = sample(dim(X)[1], 1) # find datapoint to drop
X_reduced = X[-rando_index,]
y_reduced = y[-rando_index]
X_left_out = X[rando_index,]
y_left_out = y[rando_index]
df_reduced = data.frame(y = y_reduced, X = X_reduced) # train model on reduced dataset
df_one = data.frame(y = y_left_out, X = X_left_out)
logit_model = MCMClogit(y ~. , data = df_reduced) # Making the reduced models
probit_model = MCMCprobit(y ~., data = df_reduced)
logit_matrix_left_out = as.matrix(logit_model)
probit_matix_left_out = as.matrix(probit_model)
X_left_out = unname(X_left_out)
X_left_out_new = c(1, X_left_out)
logit_response = logit_matrix_left_out %*% X_left_out_new # predicting the left-out variable
probit_response = probit_matix_left_out %*% X_left_out_new
logit_probabilities = plogis(logit_response) # transform these into probabilities
probit_probabilities = pnorm(probit_response)
library(coda) # Using coda to get the HPD interval because it works well with MCMC objects
hpd_logit = HPDinterval(as.mcmc(logit_probabilities), prob = 0.95)
mean_prob_logit = mean(logit_probabilities)
hpd_probit = HPDinterval(as.mcmc(probit_probabilities), prob = 0.95)
mean_prob_probit = mean(probit_probabilities)
cat("The 95% hpd interval for logistic: [", hpd_logit, "]. \n")
```

```
## The 95% hpd interval for logistic: [ 0.3225864 0.5820579 ].
```

```r
cat("Posterior predictive mean of datapoint = ", mean_prob_logit, ", True label =", y_left_out, "\n")
```

```
## Posterior predictive mean of datapoint =  0.4543663 , True label = 0
```

```r
cat("The 95% hpd interval for probit: [", hpd_probit, "]. \n")
```

```
## The 95% hpd interval for probit: [ 0.335554 0.5731406 ].
```

```r
cat("Posterior predictive mean of datapoint ", mean_prob_probit, ", True label =", y_left_out, "")
```

```
## Posterior predictive mean of datapoint  0.4577052 , True label = 0
```

Simulating from the posterior predictive distributions of both the logit and probit models produced fairly similar HPD intervals, [0.32, 0.58] and [0.34, 0.57] respectively. Likewise, the posterior predictive means were similar at 0.454 and 0.458 respectively. The true label of the response value was 0. If our cutoff towards definitively classifying a point was $\hat{y} = 1$ if $p > 0.5$, then we would classify this point as 0 and get a correct prediction. This, along with the fact that the model conclusions were consistent with each other, gives me more faith in the accuracy of the models.

Problem 2

Part a

First, I'll pre-process the data by combining the men's and women's results and only including the gender, Vaporfly, marathon (course), and full_name features, while treating time_minutes as the response. I'll also remove any rows where the Vaporfly variable was listed as NA. Here, gender = 0 signifies woman, gender = 1 signifies man, and vaporfly = 0 means the runner did not wear the Vaporfly while Vaporfly = 1 means the runner did wear the Vaporfly.

```r
df_women = read.csv("~/Downloads/women_sampled_shoe.csv")
df_men = read.csv("~/Downloads/men_sampled_shoe.csv")
df_women$gender = 0
df_men$gender = 1
df_combo = rbind(df_women, df_men) # combining men and women datasets
df <- df_combo[, !(names(df_combo) %in% c("name_age", "match_name", "year", "date", "time"))]
df$vaporfly <- as.numeric(df$vaporfly) # encoding the vaporfly feature
df_clean <- df[!is.na(df$vaporfly), ] # removing all the rows with no value of for vaporfly
head(df_clean)
```

```
##         full_name        marathon time_minutes vaporfly gender
## 1 CAROLINE ROTICH Boston Marathon     144.9167        0      0
## 2     MARE DIBABA Boston Marathon     144.9833        0      0
## 3   BUZUNESH DEBA Boston Marathon     145.1500        0      0
## 4  DESIREE LINDEN Boston Marathon     145.6500        0      0
## 5   SHARON CHEROP Boston Marathon     146.0833        0      0
## 6  CAROLINE KILEL Boston Marathon     146.6667        0      0
```

Since the response here is continuous (marathon time in minutes), I'm going to use a linear regression model for this problem. There are 657 unique runners in the dataset, this is way too many for us to model it as a feature without running into serious dimensionality issues. While modeling the runner as a random effect is a reasonable solution to this issue, for whatever reason, bridge sampling performs poorly when the runner is included, even when modeling it as a random effect (I'll demostrate this later). For this reason, I have decided to omit runner from most of my models in the hopes that the gender and marathon (course) features can capture most of this variation themselves. There are 23 different courses, which is small enough to where the bridge sampler can handle it, but big enough to justify modeling it as a random effect. A random effects model accounts for variation across the courses by assuming that group-level effects are randomly drawn from a common distribution.

For the analysis, I will be creating five models. The first will treat the Vaporfly effect as being separate from both the course and gender. The second will treat the Vaporfly effect as varying by course. The third will treat it as if it varies by gender. The fourth will treat it as if it varies by both course and gender. The fifth is the same as the forth but includes runner as a random effect. One important assumption I will make is that the effect of the Vaporfly on finishing time is additive and not multiplicative. In the study linked in the assignment, the authors modeled the effect as both a multiplicative one and an additive one, but in the interest of limiting the experiment to 3-5 models I will just stick to the additive interpretation, in which the effect is modeled as a constant shift rather than a proportional shift. I am also working with many of the same assumptions as the authors in the original paper, assuming that the Vaporfly effect doesn't differ by runner, that the marathon time is normally distributed (when in actuality it is probably skewed). Also I am also not differentiating between the different models of Vaporfly.

For the fixed effects, I decided to set a normal prior with mean 0 and standard deviation 5. I didn't want to make any strong assumptions about the relative effects of each feature on the final marathon time, so I just set a mean of 0. I thought a standard deviation of 5 cast a decently wide net, and once again, the normal distribution is nice because it covers the whole real line and doesn't rule out any values for the priors. For the standard deviation of the group-level random effects, I also opted for a normal distribution because of its coverage, this time with mean = 0 and sd = 2. This time I included a lower bound of 0 because standard deviations can't be negative (I think brm already accounts for this but I wanted to make it explicit for the sake of clarity). I put most of the mass around 0 because I wanted to adopt a non-informative posture regarding between-group differences. For similar reasons to the other two priors, I set a standard normal prior for the residual standard deviation with a lower bound of 0.

```r
library(brms) # Setting Priors
prior = c(prior(normal(0, 5), class = "b"),       # For fixed effects
  prior(normal(0, 2), class = "sd", lb = 0),       # For group-level SDs
  prior(normal(0, 1), class = "sigma", lb = 0))    # For residual SD
```

Part b I decided to use brms because it allows me to access Stan, an extremely powerful Monte Carlo simulator that normally runs on C++, in R. Stan simulates MCMC chains through something called the Hamiltonian Monte Carlo Method, an instance of the Metropolis-Hastings algorithm. It is also extremely compatible with Bridge Sampling, something I will use later to estimate the marginal likelihood and Bayes Factor.

According to the Stan website, "Hamiltonian Monte Carlo (HMC) is a Markov chain Monte Carlo (MCMC) method that uses the derivatives of the density function being sampled to generate efficient transitions spanning the posterior (see, e.g., Betancourt and Girolami (2013), Neal (2011) for more details). It uses an approximate Hamiltonian dynamics simulation based on numerical integration which is then corrected by performing a Metropolis acceptance step." Further details can be found on the Stan website, https://mc-stan.org/docs/reference-manual/mcmc.html. The posterior draws of the parameters are done via this process.

I decided to use the default brm parameters to determine the sampling architecture, with iter = 2000, chains = 4, and with the warm-up period as half of the iters (1000). In my testing the chains seemed to converge well with these default settings.

Model 1 assumes that the Vaporfly effect is separate from gender and course, while the course is treated as a random effect. (1|marathon) indicates that I am treating the course as a random effect, each course having its own intercept.

```r
# MODEL 1: Vaporfly effect is not dependent on gender or course. Course treated as a random effect
fit1 = brm(time_minutes ~ vaporfly + gender + (1|marathon), data = df_clean, prior = prior,
           save_pars = save_pars(all = TRUE))
```

Model 2 assumes that the Vaporfly effect varies by course, while gender is still independent of it. I have removed Vaporfly as its own feature and instead have (1 + vaporfly | marathon), which defines a varying-intercept and varying-slope model. This structure allows both the baseline finishing time and the Vaporfly effect to vary across marathons.

```r
# MODEL 2: Assumes Vaporfly effect varies by course
fit2 = brm(time_minutes ~  gender + (1 + vaporfly | marathon), data = df_clean,
           prior = prior, save_pars = save_pars(all = TRUE), refresh = 0)
```

Model 3 assumes that the Vaporfly effect varies by gender, while the course is independent of it. I've captured this interaction effect with the vaporfly*gender, which assumes a different vaporfly effect for each gender and also includes their individual effects.

```r
# MODEL 3: Assumes Vaporfly effect varies by gender
fit3 = brm(time_minutes ~ vaporfly*gender  + (1 | marathon),  data = df_clean, prior = prior,
           save_pars = save_pars(all = TRUE), refresh = 0)
```

Model 4 assumes that the Vaporfly effect varies by both gender and course, incorporating both of the interaction terms I used in models 2 and 3.

```r
# MODEL 4: Assumes Vaporfly effect varies by both gender and course
fit4 = brm(time_minutes ~ vaporfly * gender + (1 + vaporfly | marathon),
           data = df_clean, prior = prior, save_pars = save_pars(all = TRUE), refresh = 0)
```

Lastly, I'll model the runner (full_name) as a random effect to see if it improves the model.

```r
# MODEL 5: same as model 4 but with runner as a random effect
fit5 = brm(time_minutes ~ vaporfly * gender + (1 + vaporfly | marathon) + (1|full_name),
           data = df_clean, prior = prior, save_pars = save_pars(all = TRUE), refresh = 0)
```

Part c Since I've fit all the models, I will use the bridgesampling pacakge to find marginal likelihoods much in the same way I did in problem 1 of this assignment.

```r
set.seed(459) # going to perform bridge sampling on all the models
bridge_fit1 = bridgesampling::bridge_sampler(fit1)
bridge_fit2 = bridgesampling::bridge_sampler(fit2)
bridge_fit3 = bridgesampling::bridge_sampler(fit3)
bridge_fit4 = bridgesampling::bridge_sampler(fit4)
bridge_fit5 = bridgesampling::bridge_sampler(fit5)
```

Next, I'll use the bf function in bridgesampling to get the Bayes factors.

```r
bridgesampling::bf(bridge_fit1, bridge_fit2)
```

```
## Estimated Bayes factor in favor of bridge_fit1 over bridge_fit2: 642.44766
```

```r
bridgesampling::bf(bridge_fit1, bridge_fit3)
```

```
## Estimated Bayes factor in favor of bridge_fit1 over bridge_fit3: 4.91327
```

```r
bridgesampling::bf(bridge_fit1, bridge_fit4)
```

```
## Estimated Bayes factor in favor of bridge_fit1 over bridge_fit4: 7.99539
```

```r
bf_5 = bridgesampling::bf(bridge_fit5, bridge_fit1)
format(bf_5$bf, scientific = TRUE)
```

```
## [1] "9.48126e+43"
```

According to Jeffreys scale, a Bayes factor over 1 indicates evidence for h0. Here, the Bayes Factor for fit1 over fits 2, 3 and 4 is over 1, indicating that we have evidence that fit1 is better than those three other models. Due to fit1 being favored over the other three models in head-to-head comparisons, we can conclude that fit1, which assumes that the Vaporfly effect is independent of gender and course, is the best-fitting model.

The Bayes factor of fit5 over fit 1 is an extremely large value, but this is **NOT** decisive evidence that fit5 is better than fit 1. Rather it is evidence that bridge sampling is extremely unstable while modeling the runner as a random effect, probably due to the aforementioned large number of different runners present in the data (over 650), leading to an untrustworthy marginal likelihood. I know this because when I ran the bridge sampler on fit5 it maxed-out at 1000 iterations and threw the warning: >logml could not be estimated within maxiter, rerunning with adjusted starting value. Estimate might be more variable than usual.

I did not receive this warning for any of the other bridge samplers, which converged in under 10 iterations. Even after increasing the max iterations, the sampler failed to converge onto a stable marginal likelihood. Because the bridge sampler is extremely unstable when the runner modeled as a random effect and returns an extremely high-variance Bayes factor, we can disregard the bayes factor for fit 5, citing the instability of the estimator, and assume that **Fit 1** is, in fact, the best model.

These marginal likelihoods were all estimated using bridge sampling from the bridgesampling package. I previously used bridge sampling to compute the marginal likelihoods in problem 1. As mentioned before, bridge sampling uses both the unnormalized posterior and a proposal desnity (usually multivariate normal) so estimate the marginal likelihood. In problem 1, I provided the function with the unnormalized posterior, but here, I didn't have to because Stan-based MCMC models automatically pass the posterior information into bridge sampling. Bridge sampling writes marginal likelihood as a ratio of expectations under the posterior and proposal distributions. It then solves for this using an iterative scheme that minimizes the variance of the estimator. This process gives us a very stable and accurate result. For more information of bridge sampling, one can look at the 2017 paper “A Tutorial on bridge sampling:” https://arxiv.org/pdf/1703.05984.

Part **d**

```r
summary(fit1)
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: time_minutes ~ vaporfly + gender + (1 | marathon)
##    Data: df_clean (Number of observations: 1618)
##   Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup draws = 4000
##
## Multilevel Hyperparameters:
## ~marathon (Number of levels: 23)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     3.67      0.60     2.63     5.01 1.00     1111     1733
##
## Regression Coefficients:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept   161.61      0.90   159.92   163.44 1.01      557     1195
## vaporfly     -2.47      0.53    -3.49    -1.42 1.00     4414     2479
## gender      -20.28      0.38   -21.04   -19.53 1.00     3667     2881
##
## Further Distributional Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     7.13      0.12     6.90     7.37 1.00     4066     2600
##
## Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

Let’s look at model 1 more closely. As you can see, the model assigns a coefficient value of ~-2.5 for the Vaporfly variable, which implies that that wearing Vaporfly improves the marathon time of experienced runners by ~2.5 minutes on average. We can conclude that a Vaporfly effect probably exists because the 95% credible interval bounds for the Vaporfly feature are well below zero, with the upper bound being ~-1.4. We can also conclude that the Vaporfly effect does not vary significantly by course and/or gender, because we found that the marginal likelihood of the model in which the Vaporfly effect was independent of those other two features was greater than the marginal likelihood of the models in which the Vaporfly effect varied based on one or both of the two other features. It is also reasonably safe to assume that the Vaporfly effect does not vary by runner, because the overall variance between the two gender groups is larger than the variance between runners of the same gender. It seems unlikely that the Vaporfly effect does not vary significantly between the gender groups but does between individual runners, so we can assume that the effect does not vary significantly with the runner. Based on these results, it is reasonably safe to conclude that a) the Vaporfly effect exists and b) it probably does not vary significantly by runner, course, or gender.