Bayesian Missing Data - Multiple Imputation Methods

Presented to

CSUF

COLLEGE OF

Natural Sciences and Mathematics

California State University, Fullerton

Math 538 Fall 2023

Dr. Poynor

Prepared by

Paul LOPEZ

Emilio VASQUEZ

December 07, 2023

Contents

1 Introduction

Missing data is an unavoidable issue that arises in many real-world datasets across various fields, including finance, healthcare, and social sciences. When data are missing, it can introduce biases and lead to invalid statistical inferences if the missing values are simply ignored or eliminated. Multiple imputation is a principled approach for handling missing data that involves creating multiple complete versions of the incomplete dataset, with the missing values imputed through simulated draws from an appropriate model. The key insight is that missing data uncertainty can be represented by generating multiple imputed datasets.

In the Bayesian approach to multiple imputation, prior distributions are specified for the model parameters and missing data, representing our beliefs before seeing the data. The posterior distributions of the parameters and missing values are estimated by fitting the model to the observed data using Markov chain Monte Carlo (MCMC) methods. This allows simulating multiple imputed datasets by drawing missing values from their posterior predictive distributions. The completed datasets can then be analyzed using standard complete-data methods, with final estimates pooled across the imputed datasets to incorporate missing data uncertainty. Bayesian multiple imputation provides a flexible framework for handling missing data while properly representing imputation uncertainty.

A key advantage of the Bayesian approach to multiple imputation is that it allows incorporating appropriate prior information and modeling complex relationships between variables. For example, hierarchical priors can be used to share information between related parameters or models, improving estimates for variables with limited data. Bayesian models like mixtures and nonparametric models provide flexibility to adapt to complex patterns in the data. MCMC provides a convenient computational approach for fitting Bayesian models with missing data, avoiding analytical intractability. The posterior predictive distribution for the missing values conditions on both the observed data and model parameters, ensuring proper imputation uncertainty. Practical implementations utilize packages like mice in R and mi in Python, which automate iterative Bayesian modeling, imputation, and analysis. Overall, Bayesian multiple imputation provides a robust approach for handling missing data that accounts for uncertainty and allows incorporating flexible modeling and prior information about the data structure.

2 Motivation

2.1 Missing Data

Unlike working with data in a classroom setting, real data is messy and never clean. Often times you can have records that have non-uniform inputs captured by somebody. Maybe someone left in a free-form field for entering in a birthday so all the birthdays entered in by the users may all be in different formats. The biggest dilemma that many professional will face in their data science careers is the big question of missing data. Missing data come in the following flavors:

2.1.1 Missing Completely At Random (MCAR)

Missing data can come in the form of missing completely at random. This is when the missing data is completely random. For example, imagine students are taking a survey regarding study habits. At random, some of these students spilled liquid on their survey accidentally making these unable or some portion unusable. In this scenario, missing of the data is completely at random as there is not a systematic pattern related to the missing data. This scenario can be accounted for.

2.1.2 Missing at Random (MAR)

Missing at random is when the rate of missing data can be explained if one knew of some other factor. Missing data is common in financial datasets especially due to non-response on surveys or forms. For example, customers may not fill out all the fields on a loan application or investors may skip questions on an investment risk tolerance questionnaire. However, this missing data might be able to be explained by some other question that they answered on the survey, perhaps maybe on occupation. So in theory if one observed that customers who happened to be teachers skipped questions at a higher rate than those who are not teachers, this missing data could be solved if by knowint the occupation.

2.1.3 Missing Not At random (MNAR)

Missing not at random is when the missing data is related to the variable of interest. For example, suppose that one is collecting data on income, age, and home ownership status in which some individuals refuse to disclose their income. You notice that individuals who refuse to disclose their income happen to make more than those who disclose. Data of this regard cannot be necessarily accounted for.

For the types of missing data above, it becomes imperative for a data science or statistician to have tools in their toolbox to overcome the obstacle of missing data. Depending on the scenario

such as having a small dataset, throwing out incomplete records could result in losing valuable information and potential biases. Multiple imputation is one way to predict missing values based on patterns in the observed data. This provides a principled way to fill in missing values while accounting for uncertainty, rather than ad hoc methods like mean imputation. Some models that can be used are multivariate normal imputation or chained equations/sequential regression approaches. The multiple imputed datasets can then be used to train the financial models, with results appropriately combined across the imputed datasets which allows building more robust models on messy real-world data.

3 Structure of Model/Process

The model structure is a standard Bayesian regression model relating a continuous response Y to predictor variables X:

$$Y \sim N(\mu, \sigma)$$

$$\mu = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ...$$

where:

- Y is the response variable with some values missing
- X_1, X_2, etc are the predictor variables
- β_0 is the intercept
- β_1, β_2, etc are the regression coefficients
- σ is the error standard deviation

In addition, we have missingness indicator variables R for each observation, where R = 1 if Y is observed and R = 0 if Y is missing for that observation.

The priors on the parameters $\beta_0, \beta_1, \beta_2,$ and σ are weakly informative normals or half-normals:

$$\beta_0 \sim N(0, 10)$$

$$\beta_1 \sim N(0, 2)$$

$$\beta_2 \sim N(0, 2)$$

$$\sigma \sim HalfNormal(0, 10)$$

For the missing Y values, we treat them as parameters to be estimated, with priors based on the observed data likelihood:

$$Y_{miss} \sim N(\mu, \sigma)$$

where μ and σ come from fitting the model on the observed data. This makes the missing values exchangeable with the observed data.

Multiple imputation is done by drawing missing Y values from their posterior predictive distribution based on the fitted model. Multiple datasets are created by repeating this process and used to account for missing data uncertainty.

In summary, the model leverages Bayesian regression, weakly informative priors, and treats missing values as parameters to plausibly fill in gaps while quantifying uncertainty. The multiple imputed datasets integrate over the posterior distribution of the missing data.

4 Example

The model being fit in this analysis is a Bayesian linear regression model with time-series data. The formula for the model is:

$$return_t = \beta_0 + \beta_1 laggedreturn_{t-1} + \epsilon_t$$

- $return_t$ is the dependent variable, representing the log return of the AAPL stock on day t. It is calculated as the difference in the logarithm of the closing prices between two consecutive days, i.e., log($close_t$) - log($close_{t-1}$).
- $laggedreturn_{t-1}$ is the independent variable, which is the log return of the AAPL stock on the previous day t-1.
- β_0 is the intercept of the regression line, which represents the expected value of $return_t$, when $laggedreturn_{t-1}$ is zero.

- β_1 is the slope coefficient, indicating how much $return_t$ is expected to increase when $laggedreturn_{t-1}$ increases by one unit.
- ϵ_t is the error term, which accounts for the variability in $return_t$ that is not explained by $laggedreturn_{t-1}$.

The Bayesian aspect of this model comes from the use of priors and the Bayesian inference process. Instead of just finding point estimates for β_0 and β_1 as in classical regression, the Bayesian approach estimates the entire posterior distributions for these parameters based on the prior distributions and the observed data.

Below are the steps taken for this analysis:

4.1 Data Acquisition

The `tq_get` function from the `tidyquant` package retrieves historical stock price data for AAPL from an online source.

```
set.seed(555)

# Get Apple stock data
stocks <- tq_get("AAPL", get = "stock.prices", from = "2013-01-01")

# Help make sure we don't accidentally add new data
stocks <- stocks[stocks$date <= as.Date('2023-11-17'),]
```

4.2 Exploratory Data Analysis

4.2.1 Examining APPL

```
## Warning: `aes_string()` was deprecated in ggplot2 3.0.0.
## i Please use tidy evaluation idioms with `aes()`.
## i See also `vignette("ggplot2-in-packages")` for more information.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

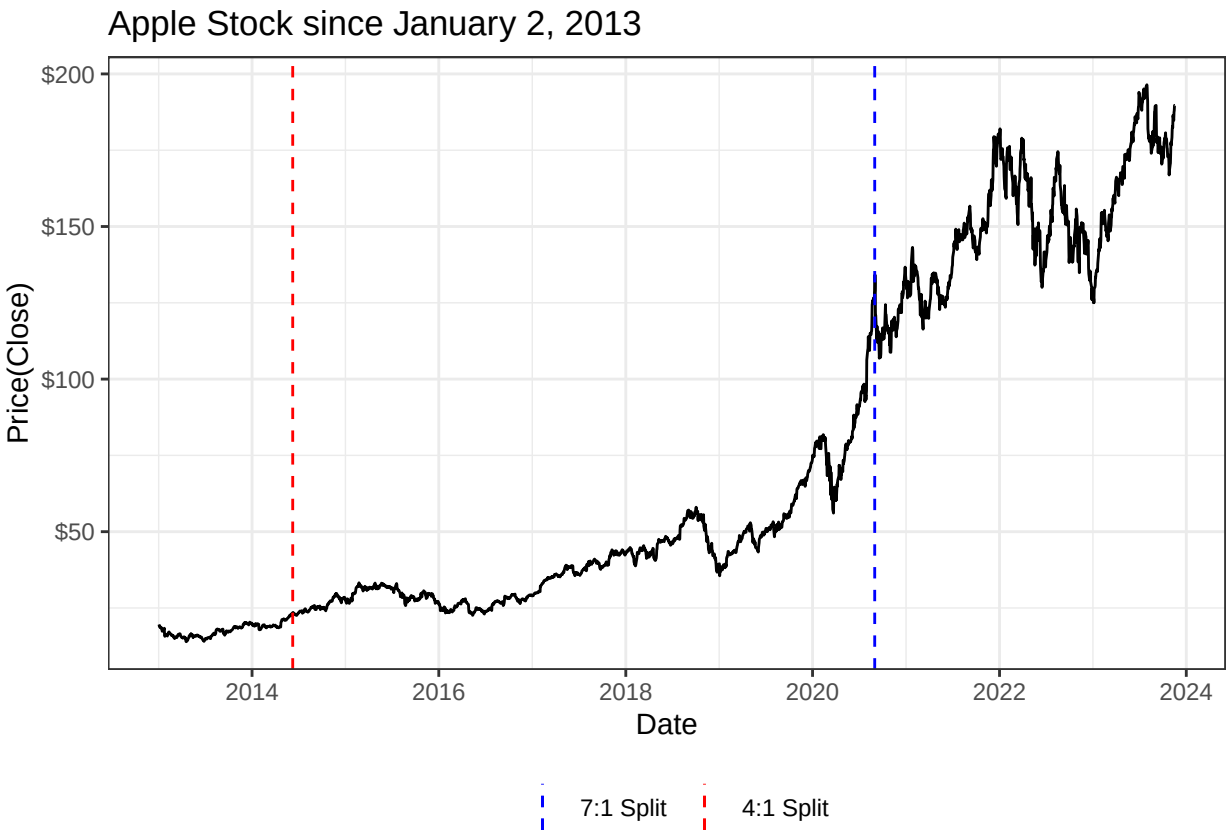
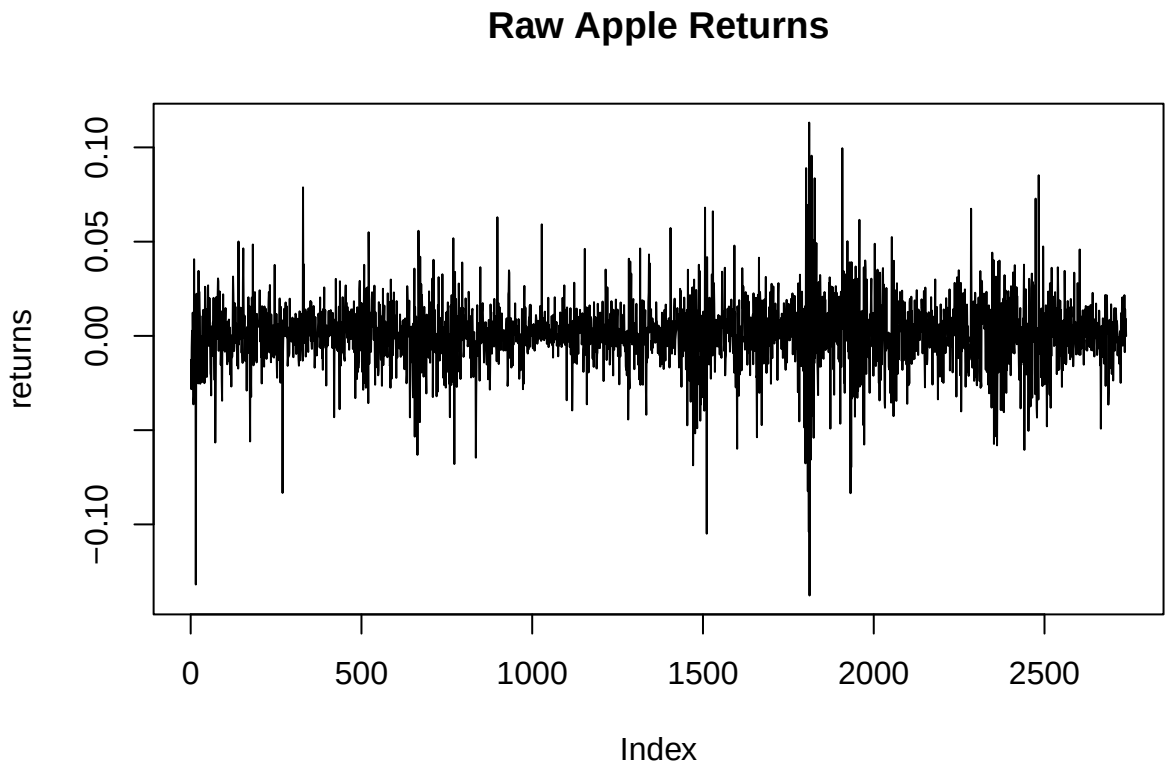


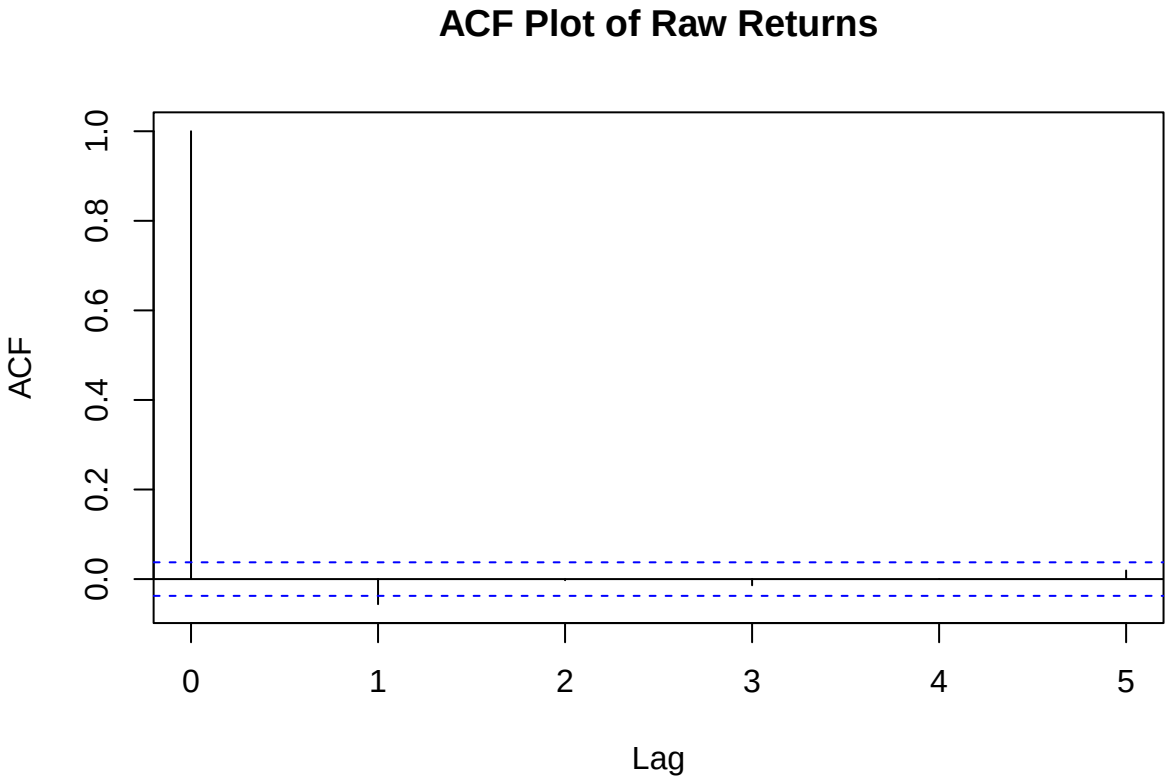
Figure X. Historical trend of APPL

With the help of Tidyquant, historical results can be provided for APPL since January 2, 2013. Overall there has been an upward trend in the daily high stock price. Within this time frame, the stock price started off at $19.60 and closed at $189.69 on November 17, 2023 which is a 162.54% increase in only a decade of data.

During this time frame, the stock split twice. The first occurred on June 9, 2014 where the stock split 7:1. So the number of shares was multiplied by 7 and saw a share price divided by 7. Following this split, there was not much of a jump in price. The second stock 4:1 stock split on August 31, 2020 saw an upward surge in price in the weeks leading up to and after the stock split.

4.2.2 ACF Plots





4.2.3 Fitting an OLS Model

```r
# OLS model
fit_ols <- lm(returns ~ lag(returns,1))
summary(fit_ols)
```

```
##
## Call:
## lm(formula = returns ~ lag(returns, 1))
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.132257 -0.008293 -0.000022  0.009362  0.106461
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)      0.0008799  0.0003437   2.560  0.01051 *
## lag(returns, 1) -0.0559438  0.0190861  -2.931  0.00341 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.01796 on 2736 degrees of freedom
##   (1 observation deleted due to missingness)
## Multiple R-squared:  0.00313,    Adjusted R-squared:  0.002766
## F-statistic: 8.592 on 1 and 2736 DF,  p-value: 0.003405
```

4.3 Data Amputation Prep and Procedure

- A subset of the retrieved data is selected, focusing on the 'close' (closing price) and 'volume' (number of shares traded) columns.
- The `set.seed` function sets the random number generator's seed to ensure reproducibility of the results.

The mice::ampute function artificially introduces missing values into the dataset to simulate incomplete data, which is common in real-world scenarios. The prop argument specifies the proportion of data to be made missing, and the mech argument specifies the missingness mechanism as 'MAR' (Missing At Random).

```r
# Select multiple columns for the amputation process
stocks_for_ampute <- stocks[, c("close", "volume")]

# Introduce missing values
amputed_data <- mice::ampute(stocks_for_ampute, prop = 0.3, mech = "MAR")

# Update the stocks with the amputed data for 'close'
stocks$close <- amputed_data$amp[, "close"]

# Ensure all necessary columns are present after amputation
stocks <- as.data.frame(stocks)
```

4.4 Data Preparation Post-Amputation

- The stocks dataframe is updated with the amputed 'close' prices.

```
## Warning: Removed 1 row containing missing values (`geom_line()`).
```

Figure X. Visualization of the amputated data

In the above figure, one can visually see how mice amputated the data. Although it could be hard to see due to the visualization, it appears there is a sizable gap in data a few months prior to the year 2021.

- New variables 'return' and 'lagged_return' are calculated using the `dplyr` package. These represent the daily returns of the stock and the previous day's return, respectively. The `na.omit` function removes any rows with missing values which can't be used in the subsequent regression analysis.

```
# Calculate 'return' and 'lagged_return' for your model
stocks <- stocks %>%
  mutate(
    return = c(NA, diff(log(close))),
    lagged_return = lag(return, 1)
  ) %>%
  na.omit()  # Remove rows with NAs
```

4.5 Data Imputation

The `mice` function is used to impute missing values in the dataset. It uses Predictive Mean Matching ('pmm') method to fill in missing 'close' prices. Multiple imputed datasets are generated (specified by m = 5) to account for the uncertainty of the imputation process.

```
# Impute missing values using Predictive Mean Matching
imputed <- mice(data = stocks, m = 5, maxit = 20, method = 'pmm', seed = 123)
```

```
##
##  iter imp variable
##   1   1
##   1   2
##   1   3
##   1   4
##   1   5
##   2   1
##   2   2
```

```
##    2    3
##    2    4
##    2    5
##    3    1
##    3    2
##    3    3
##    3    4
##    3    5
##    4    1
##    4    2
##    4    3
##    4    4
##    4    5
##    5    1
##    5    2
##    5    3
##    5    4
##    5    5
##    6    1
##    6    2
##    6    3
##    6    4
##    6    5
##    7    1
##    7    2
##    7    3
##    7    4
##    7    5
##    8    1
##    8    2
##    8    3
##    8    4
##    8    5
##    9    1
##    9    2
##    9    3
##    9    4
```

```
##    9    5
##    10   1
##    10   2
##    10   3
##    10   4
##    10   5
##    11   1
##    11   2
##    11   3
##    11   4
##    11   5
##    12   1
##    12   2
##    12   3
##    12   4
##    12   5
##    13   1
##    13   2
##    13   3
##    13   4
##    13   5
##    14   1
##    14   2
##    14   3
##    14   4
##    14   5
##    15   1
##    15   2
##    15   3
##    15   4
##    15   5
##    16   1
##    16   2
##    16   3
##    16   4
##    16   5
##    17   1
```

```
##   17   2
##   17   3
##   17   4
##   17   5
##   18   1
##   18   2
##   18   3
##   18   4
##   18   5
##   19   1
##   19   2
##   19   3
##   19   4
##   19   5
##   20   1
##   20   2
##   20   3
##   20   4
##   20   5
```

```
## Warning: Number of logged events: 5
```

```r
# Extract imputed datasets
imp_datasets <- lapply(1:5, function(i) complete(imputed, i))
```

The PMM Process for this mice (Multivariate Imputation by Chained Equations) function works as follows:

i) Fitting of the initial model: We have our variables generically as X (predictors) and Y(variable with missing values). We fit a regression model:

$$\hat{Y} = f(X;\theta), \text{ where } \theta \text{ are the estimated parameters}$$

ii) For a missing Y_i, we find the set of cases(called donors in the package) D_i where $\hat{Y}_d \in D_i$ are closest to $\hat{Y}_i$.

iii) We now randomly select a Y_d from D_i to impute Y_i. We have the option to do a straight imputation, or we can choose n of them and take an average. In our case we did straight imputation, and the selection from this set D gives each candidate an equal chance of being selected. From what we could gather from the documentation this probability process was akin to a uniform probability distribution over the pool of potential donors.

This PMM method is a standalone, non-Bayesian imputation method. The imputed datasets generated are then used further using Bayesian methods in this project. By creating multiple imputations (m = 5), `mice` acknowledges and represents the uncertainty inherent in the imputation process.

4.6 Bayesian Regression Model

4.6.1 Setting Priors

- Priors are defined for the Bayesian regression model using the `prior` function. The priors reflect our beliefs about the parameters before seeing the data. In this case, normal priors are set for the intercept and slope (return and lagged_return coefficients).

```
# Define the priors for the Bayesian regression model
priors <- c(
  prior(normal(0, 2), class = "Intercept"), # Intercept Prior
  prior(normal(0, 1), class = "b") # Slope Prior
)
```

Intercept Prior: A normal distribution with a mean of 0 and a standard deviation of 2. This is specified by prior(normal(0, 2), class = "Intercept"). It indicates that before looking at the data, you believe the average daily return when the lagged return is zero is likely to be around 0, but there is some uncertainty, which is captured by the standard deviation of 2.

Slope Prior: A normal distribution with a mean of 0 and a standard deviation of 1. This is specified by prior(normal(0, 1), class = "b"). It suggests that before analyzing the data, you expect the impact of the previous day's return on today's return to be small, as indicated by the mean of 0. The standard deviation of 1 reflects your uncertainty about this expectation.

4.6.2 Fitting a Bayesian Regression Model

The `brm` function from the `brms` package fits a Bayesian regression model to each imputed dataset. The models predict 'return' based on 'lagged_return'. The `iter`, `warmup`, and `chains` arguments

control the Markov Chain Monte Carlo (MCMC) sampling process used to estimate the posterior distribution of the model parameters.

The posterior distribution combines the prior distribution with the likelihood of the observed data to update our beliefs about the model parameters. After fitting the model using `brm`, which internally uses Hamiltonian Monte Carlo (HMC) sampling, a type of MCMC, you obtain a distribution for each parameter that reflects all the information from the priors and the data.

```
# Initialize an empty list to store the models
fits <- vector("list", length = 5)

# Fit a Bayesian regression model to each imputed dataset and check for errors
for (i in seq_along(imp_datasets)) {
  dataset <- imp_datasets[[i]]
  fit <- tryCatch(
    {
      model <- brm(return ~ lagged_return, data = dataset,
                   iter = 1000, warmup = 500, chains = 2, prior = priors)
      model # Return the model
    },
    error = function(e) {
      cat(sprintf("Error in fitting model %d: %s\n", i, e$message))
      NULL # Return NULL if there was an error
    }
  )
  fits[[i]] <- fit
}
```

```
## Compiling Stan program...

## Start sampling

##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 3.9e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.39 seconds
## Chain 1: Adjust your expectations accordingly!
```

```
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 1: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 1: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 1: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 1: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.127 seconds (Warm-up)
## Chain 1:                0.102 seconds (Sampling)
## Chain 1:                0.229 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1.2e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.12 seconds
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 2: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 2: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 2: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 2: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 2: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 2: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 2: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 2: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 2: Iteration: 800 / 1000 [ 80%]  (Sampling)
```

```
## Chain 2: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 2: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.048 seconds (Warm-up)
## Chain 2:                0.106 seconds (Sampling)
## Chain 2:                0.154 seconds (Total)
## Chain 2:

## Compiling Stan program...
## Start sampling

##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 4e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.4 seconds
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 1: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 1: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 1: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 1: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.07 seconds (Warm-up)
## Chain 1:                0.091 seconds (Sampling)
## Chain 1:                0.161 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
```

```
## Chain 2:
## Chain 2: Gradient evaluation took 8e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.08 seconds
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 2: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 2: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 2: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 2: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 2: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 2: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 2: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 2: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 2: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 2: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 2: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.115 seconds (Warm-up)
## Chain 2:                0.087 seconds (Sampling)
## Chain 2:                0.202 seconds (Total)
## Chain 2:

## Compiling Stan program...
## Start sampling

##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 4.2e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.42 seconds
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1: Iteration: 200 / 1000 [ 20%]  (Warmup)
```

```
## Chain 1: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 1: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 1: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 1: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.074 seconds (Warm-up)
## Chain 1:                0.099 seconds (Sampling)
## Chain 1:                0.173 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 8e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.08 seconds
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 2: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 2: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 2: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 2: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 2: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 2: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 2: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 2: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 2: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 2: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 2: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.112 seconds (Warm-up)
## Chain 2:                0.075 seconds (Sampling)
```

```
## Chain 2:                0.187 seconds (Total)
## Chain 2:

## Compiling Stan program...
## Start sampling

##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 4e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.4 seconds
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 1: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 1: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 1: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 1: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.088 seconds (Warm-up)
## Chain 1:                0.101 seconds (Sampling)
## Chain 1:                0.189 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 6e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.06 seconds
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
```

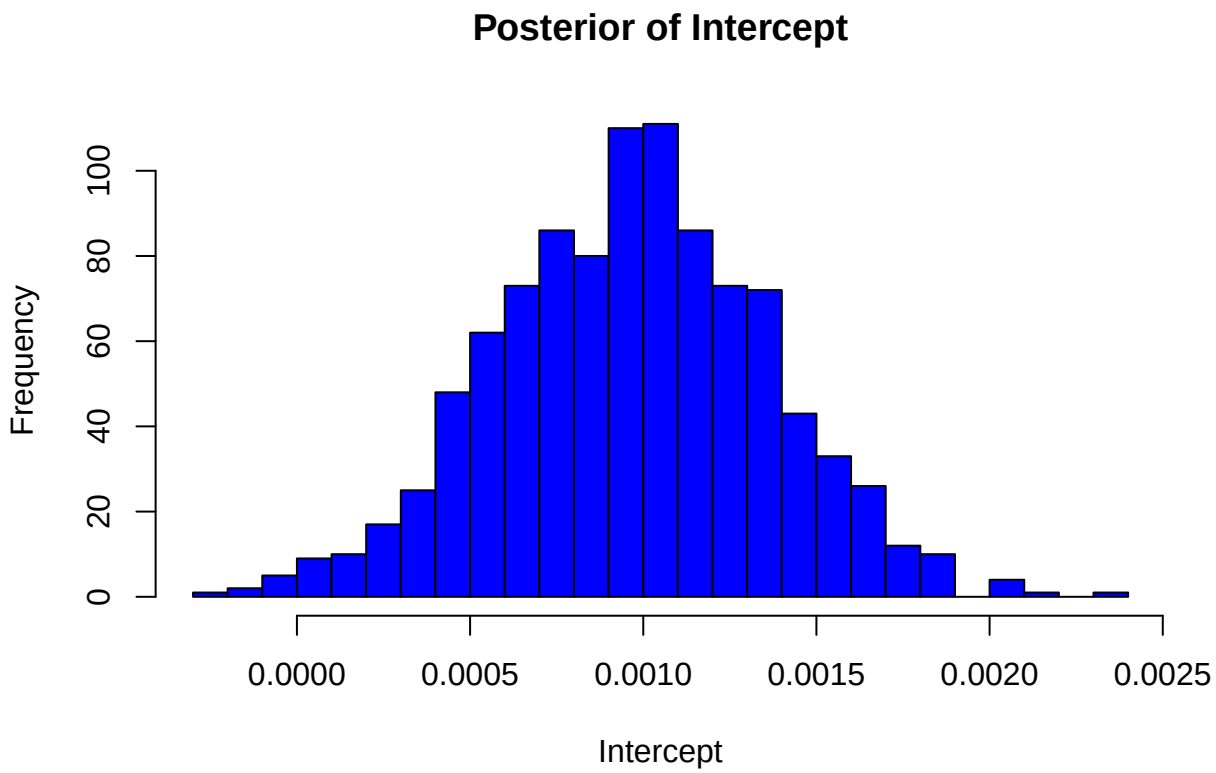
```
## Chain 2:
## Chain 2: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 2: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 2: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 2: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 2: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 2: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 2: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 2: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 2: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 2: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 2: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 2: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.081 seconds (Warm-up)
## Chain 2:                0.073 seconds (Sampling)
## Chain 2:                0.154 seconds (Total)
## Chain 2:

## Compiling Stan program...
## Start sampling

##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 4.5e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.45 seconds
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 1: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 1: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 1: Iteration: 600 / 1000 [ 60%]  (Sampling)
```

```
## Chain 1: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.071 seconds (Warm-up)
## Chain 1:                0.1 seconds (Sampling)
## Chain 1:                0.171 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1.1e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.11 seconds
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 2: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 2: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 2: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 2: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 2: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 2: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 2: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 2: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 2: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 2: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 2: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.08 seconds (Warm-up)
## Chain 2:                0.097 seconds (Sampling)
## Chain 2:                0.177 seconds (Total)
## Chain 2:
```

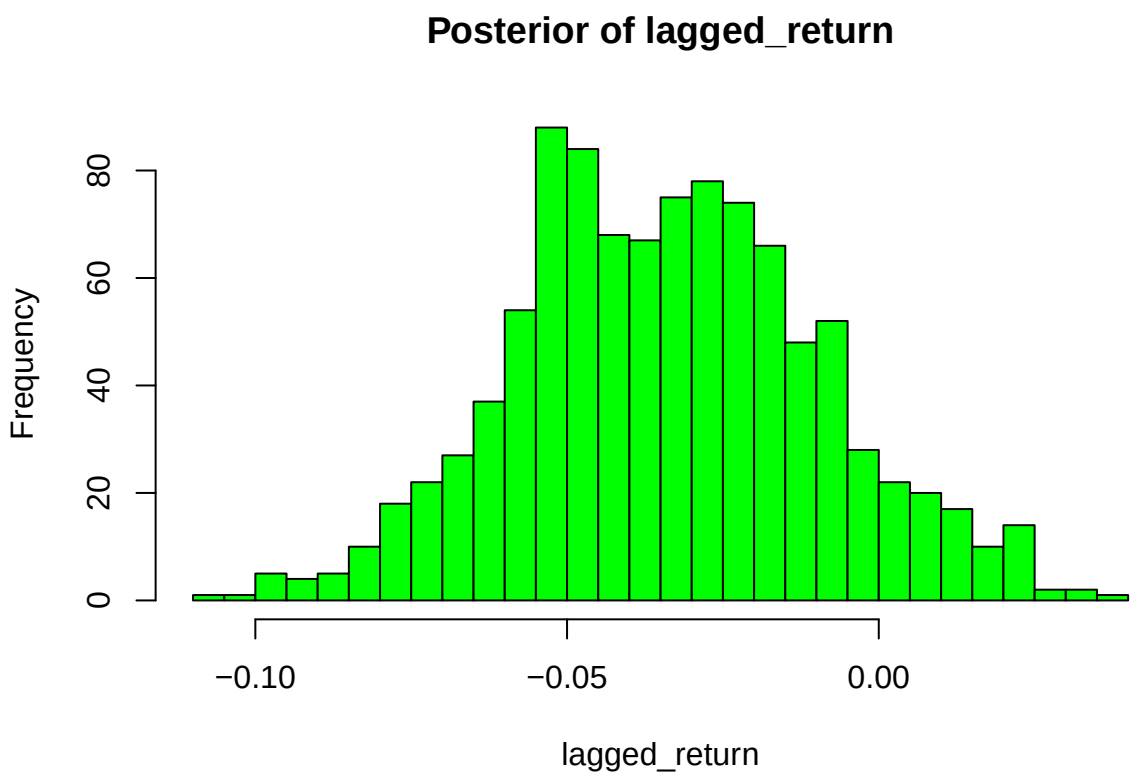
```r
# Extract posterior samples for the first fitted model
post_samples <- posterior_samples(fits[[1]], pars = c("Intercept", "lagged_return", "sigma
```

```
## Warning: Method 'posterior_samples' is deprecated. Please see ?as_draws for
## recommended alternatives.
```

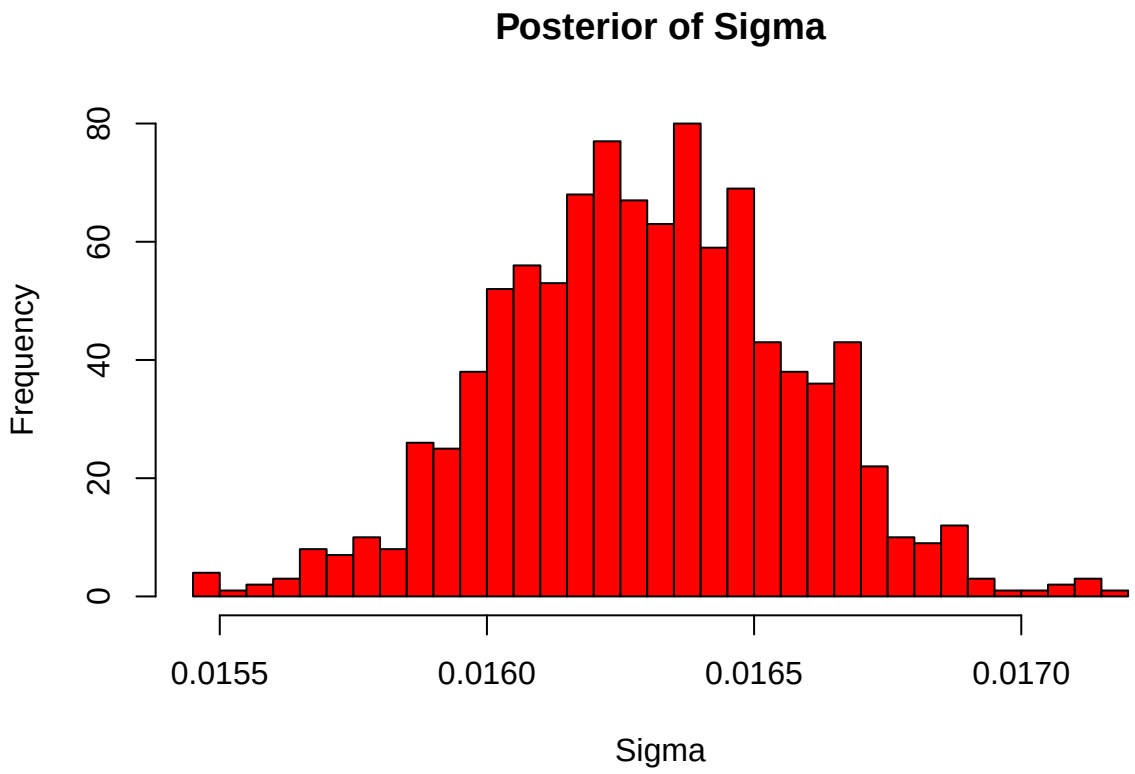
```r
# Histogram for the Intercept
hist(post_samples$b_Intercept, breaks = 30, main = "Posterior of Intercept",
     xlab = "Intercept", col = "blue")
```




```r
# Histogram for lagged_return
hist(post_samples$b_lagged_return, breaks = 30, main = "Posterior of lagged_return",
     xlab = "lagged_return", col = "green")
```




```
# Histogram for sigma
hist(post_samples$sigma, breaks = 30, main = "Posterior of Sigma",
     xlab = "Sigma", col = "red")
```




```r
# Initialize lists to store parameter estimates and standard errors
param_estimates <- list()
param_se <- list()

# Extract parameter estimates and standard errors from each model
for (i in seq_along(fits)) {
  if (!is.null(fits[[i]])) {
    # Extracting estimates
    est <- as.data.frame(summary(fits[[i]])$fixed)[, "Estimate"]
    param_estimates[[i]] <- est

    # Extracting standard errors
    se <- as.data.frame(summary(fits[[i]])$fixed)[, "Est.Error"]
    param_se[[i]] <- se
  }
}

# Calculate the mean of the parameter estimates
```

```r
combined_estimates <- Reduce("+", param_estimates) / length(fits)

# Calculate the pooled standard error if necessary
combined_se <- sqrt(Reduce("+", lapply(param_se, `^`, 2))) / length(fits)

# Combine into a data frame
combined_results <- data.frame(Estimate = combined_estimates, Std.Error = combined_se)

# Print combined results
#print(combined_results)

# Initialize lists to store CIs
ci_lower <- list()
ci_upper <- list()

for (i in seq_along(fits)) {

  if (!is.null(fits[[i]])) {

    # Extract CIs
    ci <- as.data.frame(summary(fits[[i]])$fixed)[, c("l-95% CI", "u-95% CI")]

    ci_lower[[i]] <- ci[,1]
    ci_upper[[i]] <- ci[,2]

  }

}

# Combine CIs

ci_lower_combined <- Reduce("+", ci_lower) / length(fits)
ci_upper_combined <- Reduce("+", ci_upper) / length(fits)

combined_results$ci_lower <- ci_lower_combined
combined_results$ci_upper <- ci_upper_combined
```

```r
print(combined_results)
```

```
##        Estimate    Std.Error      ci_lower    ci_upper
## 1  0.0009641508 0.0001747299  0.0002066848 0.001722092
## 2 -0.0348266417 0.0106010171 -0.0800723292 0.012406600
```

4.7 Combining of Model Results

- Parameter estimates and standard errors are extracted from each fitted model. These estimates are combined by calculating the mean estimate and pooled standard error for each parameter across all imputed datasets.

4.8 Summary and Analysis

The combined results are printed out for interpretation.

This analysis allows for Bayesian inference on time-series data with missing values. The choice of prior values should be justified based on domain knowledge or previous research. Since normal priors are used for both the intercept and slope, it implies a belief that the coefficients are likely to be around 0 but allowing for some variation.

The mice::ampute function is crucial because it enables researchers to understand how well their imputation and analysis methods work when some data is missing, which is a common occurrence in real-world datasets.

The brm function is central to the Bayesian approach, as it fits a model within the Bayesian framework using Hamiltonian Monte Carlo, a type of MCMC sampling. The function returns a wealth of information about the model, including estimates of the posterior distribution of the model parameters, which reflects both the data and the priors.

5 Conclusion

Conclusion: Recap you topic and provide discussion on relevant points from your presentation. Perhaps motivate an extension or different application to which your Model/Process could be utilized.

6 Reference

References: In your presentation you do not need to have a formal reference slide, however, be sure to appropriately reference books/papers/codes that you used in your written report.

1. Lai, Mark. 2019. "Course Handouts for Bayesian Data Analysis Class." Class handouts, December 13, 2019. https://bookdown.org/marklhc/notes_bookdown/missing-data.html

2. Little, Roderick. 2011. "Calibrated Bayes, for Statistics in General, and Missing Data in Particular." Statistical Science 26 (2): 162–74. https://doi.org/10.1214/10-STS318.