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 15, 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. If the missing values are ignored or eliminated, it can introduce biases and lead to invalid statistical inferences. Multiple imputation is a principled approach for handling missing data that involves creating multiple complete versions of an 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 can be specified for the model parameters and missing data, representing initial beliefs before examining 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` (Multivariate Imputation by Chained Equations) in R and `miceforest` 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-life data is messy and almost never "clean" out of the box. Often times one can have records that have non-uniform inputs captured. 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 data professionals will face in their data science careers is the missing data. Perhaps a survey respondent forgot to answer a few questions because it either did not relate to them or they just forgot to answer it. The nature of missing data can come in the following three 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 their drink on their survey accidentally making these entirely unable or some portion unusable. In this scenario, missingness of the data is completely at random as there is not a systematic pattern related to the missing data.

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 by knowing 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 their income level. Missing data of this nature cannot be necessarily accounted for.

For the types of missing data above, it becomes imperative for a data scientist or statistician to have tools to overcome the obstacle of missing data. Up to this point, there are already two solid approaches; removing records entirely for those that have missing data as well as applying the EM algorithm.

With removing entire rows of data, this might be an excellent and easy strategy to implement if dealing with a large dataset. Removing 2,000 rows of data from 2,000,000 rows will not affect the analysis all that much if at all. Working with a small dataset, throwing out incomplete data could result in losing valuable information and bring potential biases into any estimates provided to a stakeholder. The next approach is to apply the expectation maximization (EM) algorithm where one replaces NAs with the conditional average. The details on a third method now follow.

3 Structure of Model/Process

3.1 Overview

When working in a Bayesian setting, one common way is to address missing data is to apply a two-step process:

- First step would be to impute the missing data via a well known package such as 'mice'. 'mice' supports the creation of multiple imputed datasets which 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. In this implementation, predictive mean matching is utilized to impute the datasets in each instance.
- The second step in the process would be to utilize the multiple imputed datasets to train the predictive models, with results appropriately combined across the imputed datasets which allow building more robust models on messy real-world data.

3.2 Predictive Mean Matching Theory

The PMM Process for this 'mice' 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) Randomly select a Y_d from D_i to impute Y_i. There is the option to do a straight imputation or choose n Y_d and take an average. Straight imputation was performed in this analysis and the selection from this set D gives each candidate an equal chance of being selected. Upon exploring the mice documentation, this probability process was akin to a uniform probability distribution over the pool of potential donors.

The 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, `mice` acknowledges and represents the uncertainty inherent in the imputation process. Upon creating multiple imputed datasets, one now can apply whatever modeling technique they are interested in. In this implemntation, the next step was a Bayesian Regression Model.

3.3 Bayesian Regression Model

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
- $\beta_1, \beta_2, \ldots$, etc are the regression coefficients
- σ is the error standard deviation

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

$$\beta_0 \sim N(0, a_1)$$

$$\beta_1 \sim N(0, a_2)$$

$$\beta_2 \sim N(0, a_3)$$

$$\sigma \sim \text{Half-normal}(0, \text{c}) \text{ or } \sigma \sim \text{t-distribution with 3 df}$$

Where $a_1, a_2, a_3, ..., c$ are constants.

4 Data Example

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

$$close_t = \beta_0 + \beta_1 close_{t-1} + \epsilon_t$$

- $close_t$ is the dependent variable, representing the close of the AAPL stock on day t.
- $close_{t-1}$ is the independent variable, which is the close 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 $close_t$, when $close_{t-1}$ is zero.
- β_1 is the slope coefficient, indicating how much $close_t$ is expected to increase when $close_{t-1}$ increases by one unit.
- ϵ_t is the error term, which accounts for the variability in $close_t$ that is not explained by $close_{t-1}$.

It is important to note that this is time series data and as such there likely exists autocorrelation between $close_t$ and $close_{t-1}$. So, autocorrelation violates linear regression assumptions but linear regression can still prove to be a useful modeling approach.

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.

The following sections will delve into the steps taken for this analysis.

4.1 Data Acquisition

The `tq_get` function from the `tidyquant` package retrieves historical stock price data for Apple Inc (APPL) from an online source. The `set.seed` function sets the random number generator's seed to ensure reproducibility of the results.

```r
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'),]

# Store pre-amp
stocks$close_preamp <- stocks$close
```

4.2 Exploratory Data Analysis

4.2.1 Examining APPL

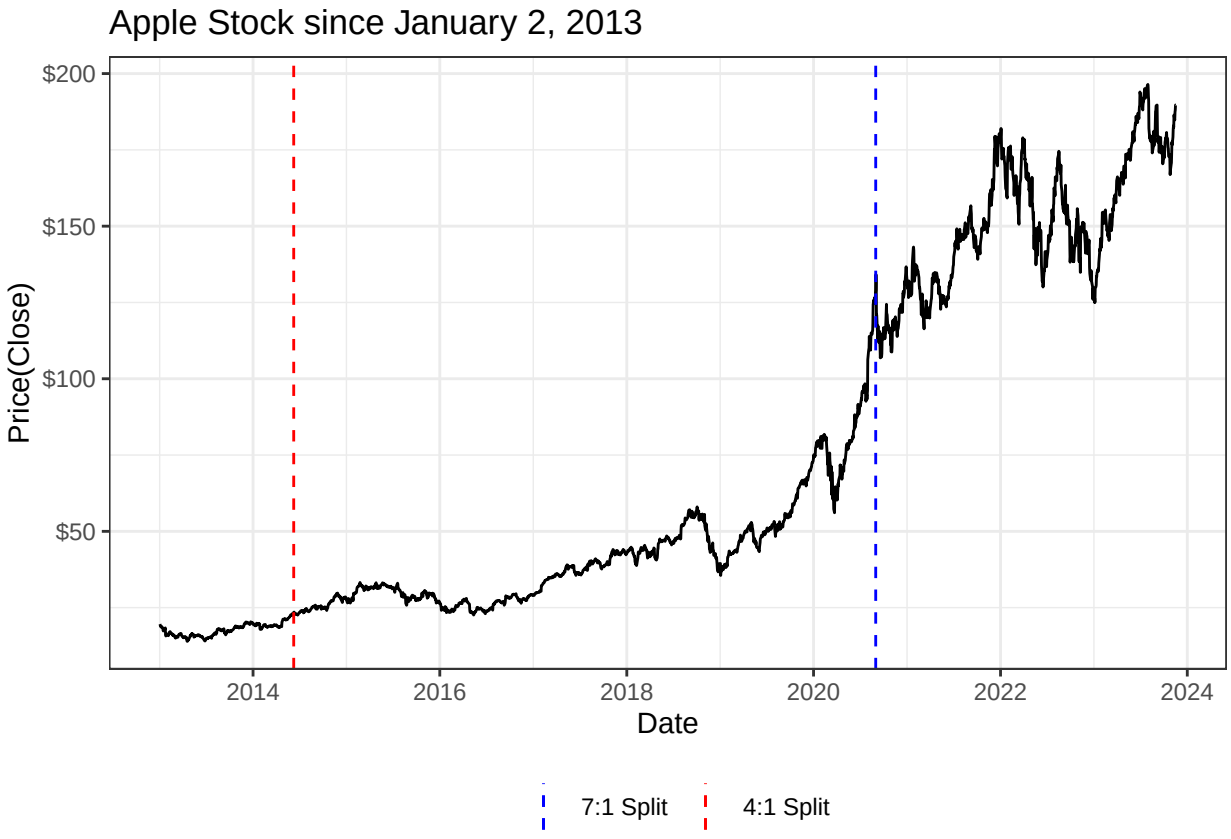


Figure 1. Historical trend of APPL

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 in the time following this split. The second 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.

4.3 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.

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

The mice::ampute function artificially introduces missing values into the dataset to simulate incomplete data, which is common in real-world scenarios. The few key arguments are used as follows:

- The prop argument specifies the proportion of data to be made missing
- The mech argument specifies the missingness mechanism as 'MAR' (Missing At Random)

```
# 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"]

# Add flag for complete data.
# 1 - complete data, 0 - incomplete data
stocks$r <- ifelse(is.na(stocks$close), 0, 1)

# 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 amputated 'close' prices.

Figure 2. Visualization of the amputated data

In the above figure, one can visually see how the 'mice' package 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.

4.5 Data Imputation via Predictive Mean Matching

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

stock_subset <- stocks[, c("symbol", "date", "close")]
```

```
imputed <- mice(data = stock_subset,
                m = 5, # Number of imputed datasets
                maxit = 25, # Max number of iterations
                method = 'pmm', # Imputation method
                seed = 555, # Set seed for reproducibility
                printFlag = FALSE # Hide output
                )
```

```
## Warning: Number of logged events: 1
```

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

#View(imp_datasets)
```

The readouts from this block of code were omitted in an effort to improve readability, but can be viewed within the appendix of this report.

4.5.1 Analyzing imputed Data

First, we will add the results of the 5 imputed datasets back into the main 'stocks' dataframe to allow futher analysis of the imputed data.

```
stocks$imp1 <- imp_datasets[[1]]$close
stocks$imp2 <- imp_datasets[[2]]$close
stocks$imp3 <- imp_datasets[[3]]$close
stocks$imp4 <- imp_datasets[[4]]$close
stocks$imp5 <- imp_datasets[[5]]$close
```

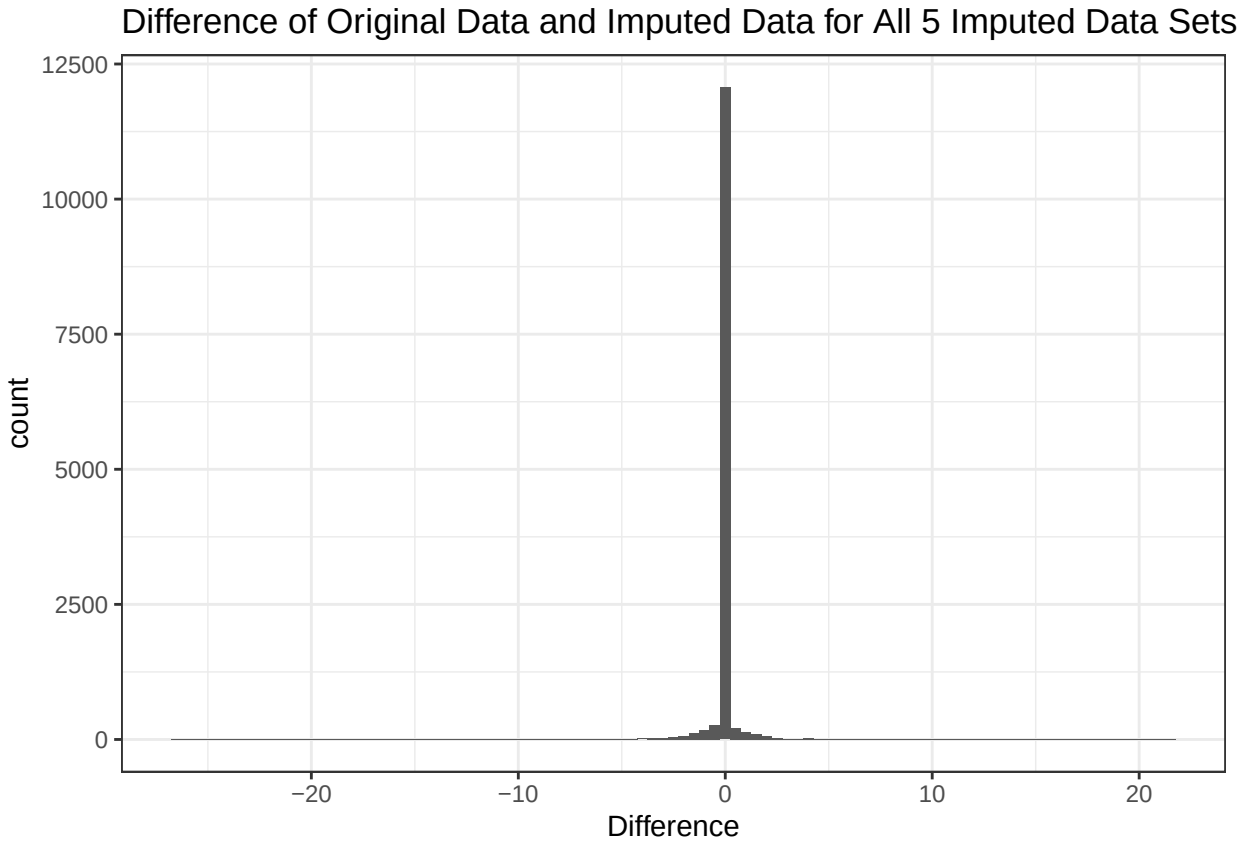


Figure 3. Histogram of the difference of the actual close price vs the imputed data points

Figure 3 shows the difference from the original score vs the imputed score for all 5 multiply imputed datasets combined into one visualization. As one can see, the vast majority of points had a very small difference between the actual and the imputed. There were a few outliers that had imputed a value greater than $20 but overall the imputation was a success. The appendix hosts histograms of differences for all 5 data sets individually.

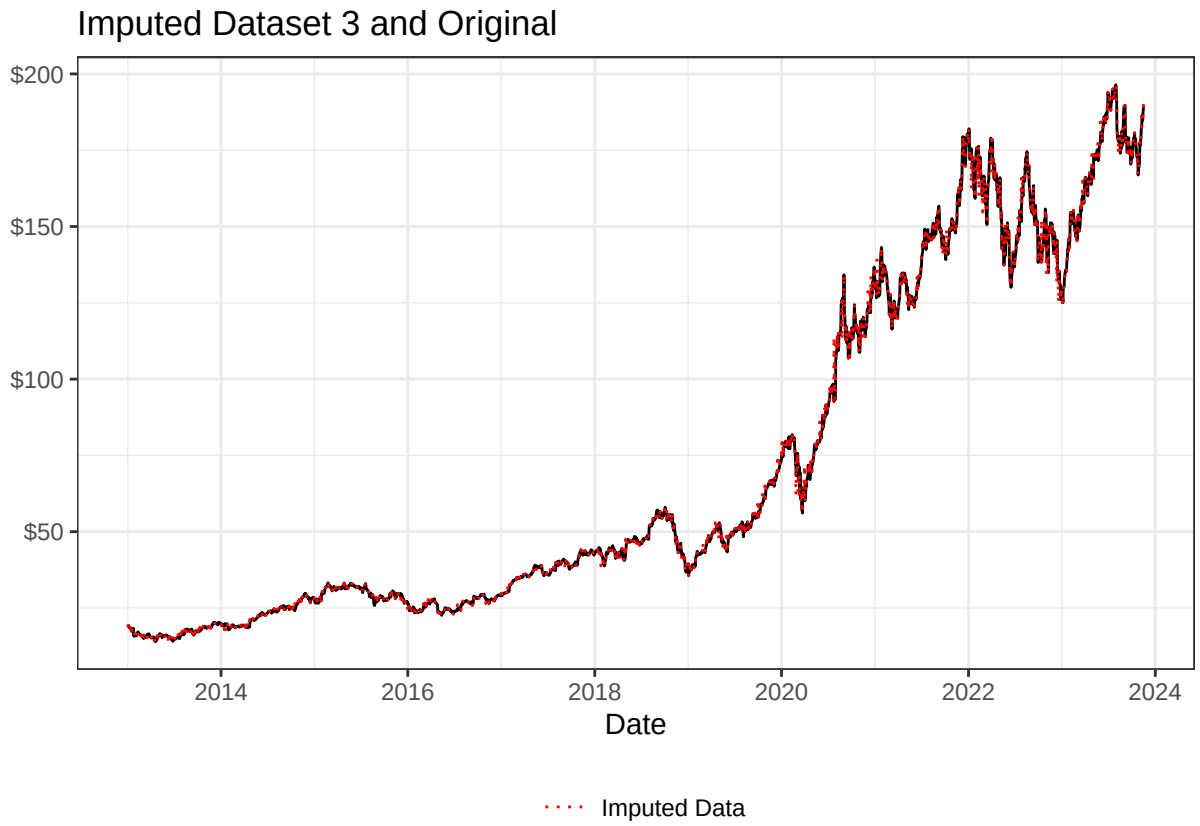


Figure 4. Imputed results for dataset 3 overlayed the actual close price

Figure 4 shows the original data overlayed with the imputed data results for the third imputed dataset. The third dataset happened to be the imputed dataset with the absolute difference between the actual data and the imputed dataset being the third largest. This means there exists imputed results where the difference was actually smaller and worse than this. The other 4 imputed dataset results can be viewed in the appendix.

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 ($close_t$ and $close_{t-1}$ 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"). A prior such as this indicates that before looking at the data, an analyst might believe the average daily return when the lagged return is zero is likely to be around 0 but would like to reflect 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, an analyst might 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 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 $close_t$ based on $close_{t-1}$. 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, one can 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]]
  # Create lag variable
  dataset$lagged_close <- c(NA, head(dataset$close, -1))
  fit <- tryCatch(
    {
```

```
      model <- brm(close ~ lagged_close, data = dataset,
                   iter = 750, 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
}
```

Due to the length the readout, the readout of this code chunk has been relegated to the appendix.

4.6.3 Bayesian Regression Results

Now that the bayesian regression model has been fit, one can now gather inference. First, the posterior samples are extracted.

```
# Extract posterior samples for the first fitted model
post_samples1 <- posterior_samples(fits[[1]], pars = c("Intercept",
                                                       "lagged_close",
                                                       "sigma"))


post_samples <- as.data.frame(rbind(posterior_samples(fits[[1]],
                                                      pars = c("Intercept",
                                                               "lagged_close",
                                                               "sigma")),
      posterior_samples(fits[[2]], pars = c("Intercept", "lagged_close", "sigma")),
      posterior_samples(fits[[3]], pars = c("Intercept", "lagged_close", "sigma")),
      posterior_samples(fits[[4]], pars = c("Intercept", "lagged_close", "sigma")),
      posterior_samples(fits[[5]], pars = c("Intercept", "lagged_close", "sigma"))))
```

The below chunk will calculate a MAP estimate as well as 95% credible interval estimates for the parameters β_0, β_1.

```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
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)

# 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
```

	Point Estimate	CI Low	CI High
β_0	0.05693082	-0.09265884	0.2163004
β_1	0.99943648	0.99766967	1.0011488
σ	2.56325	2.508397	2.622535

Figure 5. Table of finalized parameter estimates

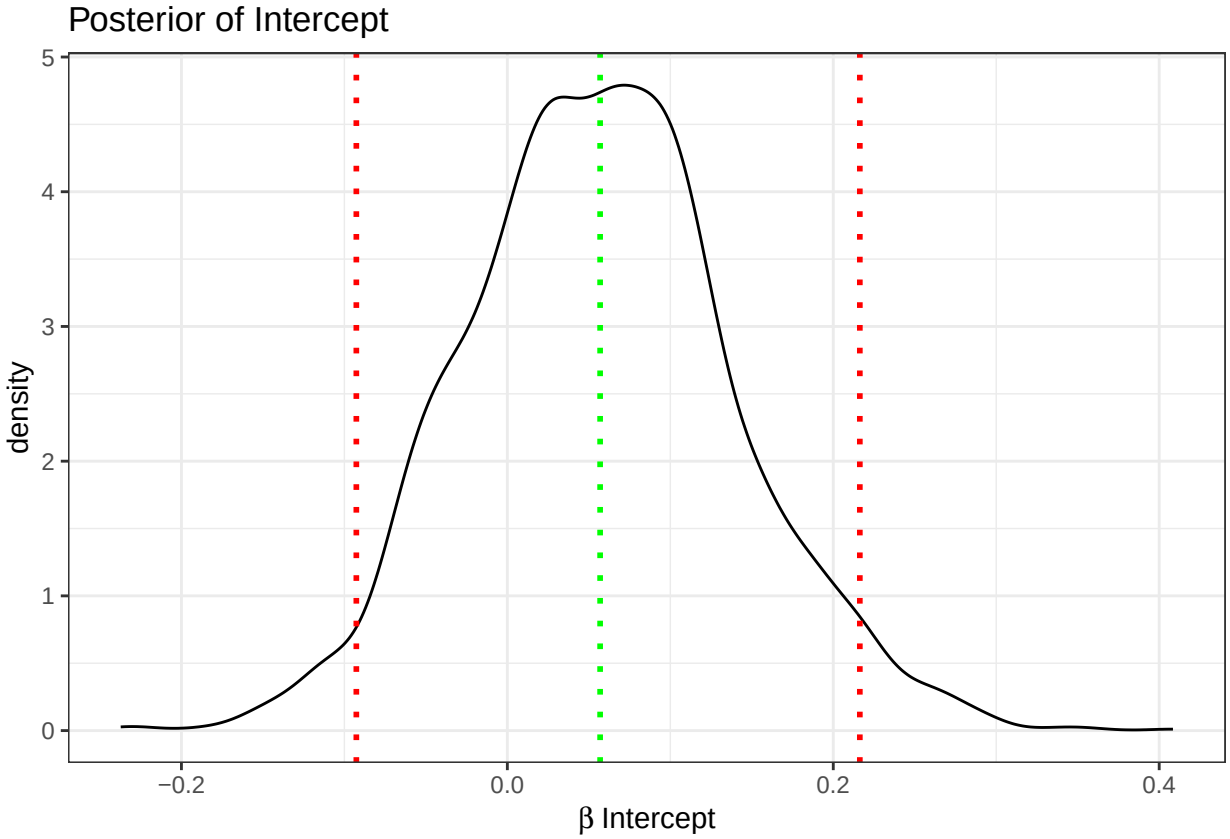


Figure 6. Posterior distribution of intercept

The intercept posterior distribution is centered around 0.05 with most of the density between -0.1 and 0.2, indicating the credible values for the expected return when the lagged close is 0. The tight concentration shows the data and prior belief in a centered intercept updated our beliefs to a quite precisely estimated narrow range. With the bulk of the posterior density above 0, there is slightly more probability weight reflecting a small positive intercept versus negative. But the distribution is still quite symmetric and centered providing evidence the daily Apple stock returns hover around a 0 intercept.

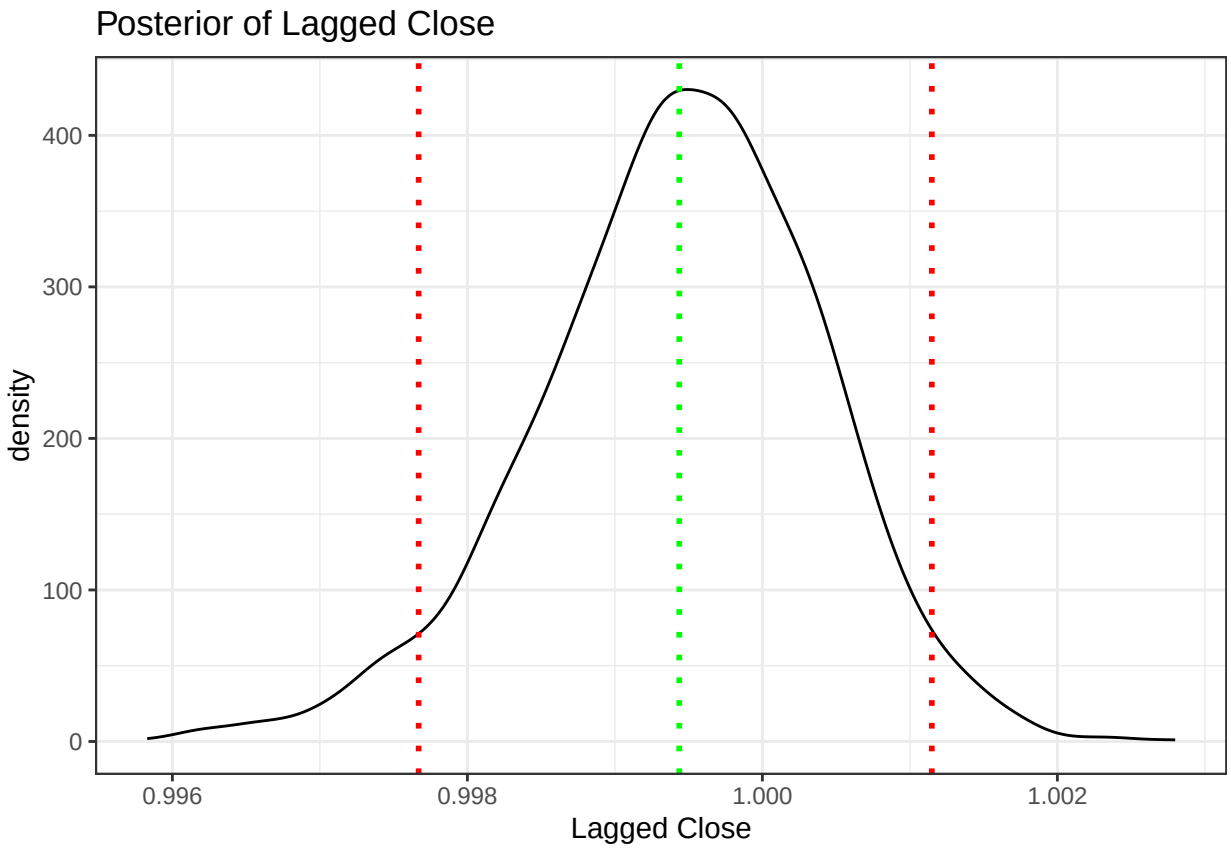


Figure 7. Posterior distribution of β for lagged close

The slope coefficient posterior is extremely tight, with essentially all the probability density between 0.9975 and 1.0025. This precision reflects the very high correlation and effect of the lagged close in predicting today's close. The posterior mean is 0.9994, indicating that for a 0.01 (1%) increase in prior day close, today's close is expected to increase by around 0.01% as well. The narrow credible interval and concentration of density shows the data and prior updated beliefs in the precise predictive effect of lagged closes. With such a peaked distribution, we can be highly certain of an exact unit-proportional relationship in the autocorrelation structure of daily Apple stock closes.

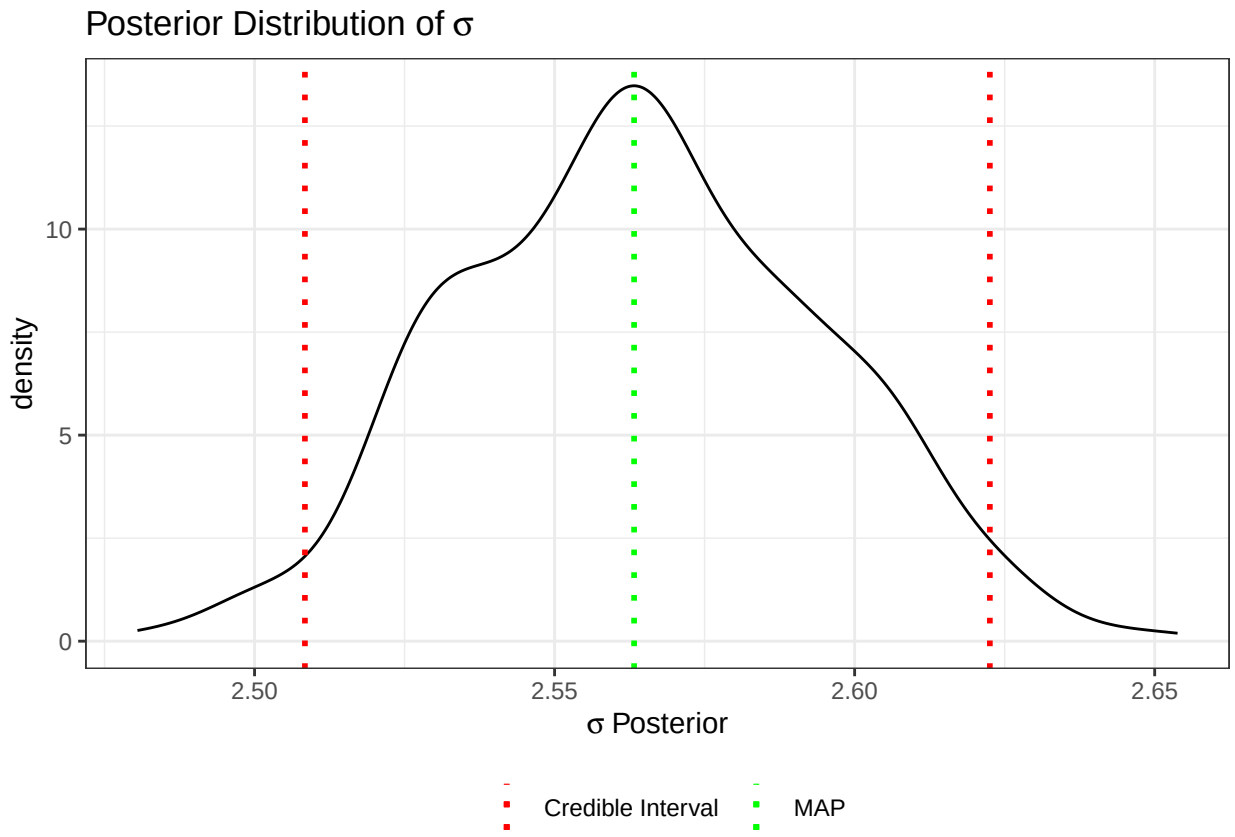


Figure 8. Posterior distribution of sigma

The sigma posterior, which represents the standard deviation of the normally distributed residuals, shows a distribution centered tightly around a value of 2.56. With a narrow 95% credible interval of 2.51 to 2.62, it indicates precision in how much daily variability exists across the time series not explained by the autocorrelation with lagged closes. The bulk of the posterior probability reflects residual standard deviation likely ranging between 2.5 and 2.6. This variability represents day to day fluctuations not accounted for by the high correlation in lagged closes, providing evidence for additional noise factors across the historical Apple stock price changes.

4.7 Comparing with OLS

As a sniff test to validate our findings from the analysis above, traditional OLS was computed for the complete data set.

```
data_all <- tq_get("AAPL", get = "stock.prices", from = "2013-01-01")
data_all$lagged_close <- c(NA, head(data_all$close, -1))
```

```r
# OLS model
fit_ols <- lm(data_all$close ~ data_all$lagged_close)
summary(fit_ols)
```

```
##
## Call:
## lm(formula = data_all$close ~ data_all$lagged_close)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.6018  -0.3957  -0.0230   0.3995  11.9172
##
## Coefficients:
##                          Estimate Std. Error  t value Pr(>|t|)
## (Intercept)             0.0425944  0.0513614    0.829    0.407
## data_all$lagged_close 1.0002981  0.0005653 1769.623   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.656 on 2751 degrees of freedom
##   (1 observation deleted due to missingness)
## Multiple R-squared:  0.9991, Adjusted R-squared:  0.9991
## F-statistic: 3.132e+06 on 1 and 2751 DF,  p-value: < 2.2e-16
```

Figure 9. OLS parameter estimates using complete data

From the figure above, one can see that the intercept and the coefficient are close to the multiple imputed Bayesian regression results.

4.8 Summary and Analysis

The combined results of the five imputed datasets were printed out for interpretation and could be considered successful as the imputed datasets performed just as well as OLS.

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. Furthermore, 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

This analysis demonstrated a Bayesian approach for handling missing data, specifically using multiple imputation to fill gaps while quantifying uncertainty. The 'mice' R package enabled missing data simulation on the AAPL stock dataset. Five imputed complete datasets were then generated using predictive mean matching. Bayesian regression models were fitted to predict daily returns based on prior day returns, combining prior beliefs about the coefficients with likelihood from the observed data to estimate posterior distributions. Hamilton Monte Carlo sampling enabled convenient fitting of the posteriors. Parameter estimates were finally pooled across the multiple imputations to obtain overall inference, incorporating missing data uncertainty.

The multi-step process provides a blueprint for principled missing data handling - imputing while preserving relationships in the data, fitting appropriate Bayesian models, and aggregating estimates for final inference. The methodology can generalize across applications with missing observations, from financial time series forecasting to political polls, epidemiology, econometrics, or climate data analysis. The multiple imputation framework allows proceeding with modeling and prevents relying only on complete cases. Use of MCMC sampling handles high-dimensional posteriors despite analytic intractability. Thus Bayesian missing data techniques serve as broadly applicable tools for overcoming an omnipresent analysis challenge to enable robust modeling from incomplete real-world data. Missing data is bound to come across an analyst's desk so the more tools they have at their disposal to work with missing data, the better the data analysis and inference can be performed.

6 References

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.

3. Gelman, Andrew et al. 2103. "Bayesian Data Analysis. (Chapman & Hall/CRC Texts in Statistical Science)"

4. Buerkner P. C. 2017. brms: An R Package for Bayesian Multilevel Models using Stan. Journal of Statistical Software. 80(1), 1-28. doi.org/10.18637/jss.v080.i01

5. Buerkner P. C. 2018. Advanced Bayesian Multilevel Modeling with the R Package brms. The R Journal. 10(1), 395-411. doi.org/10.32614/RJ-2018-017

6. Buerkner P. C. 2021. Bayesian Item Response Modeling in R with brms and Stan. Journal of Statistical Software, 100(5), 1-54. doi.org/10.18637/jss.v100.i05

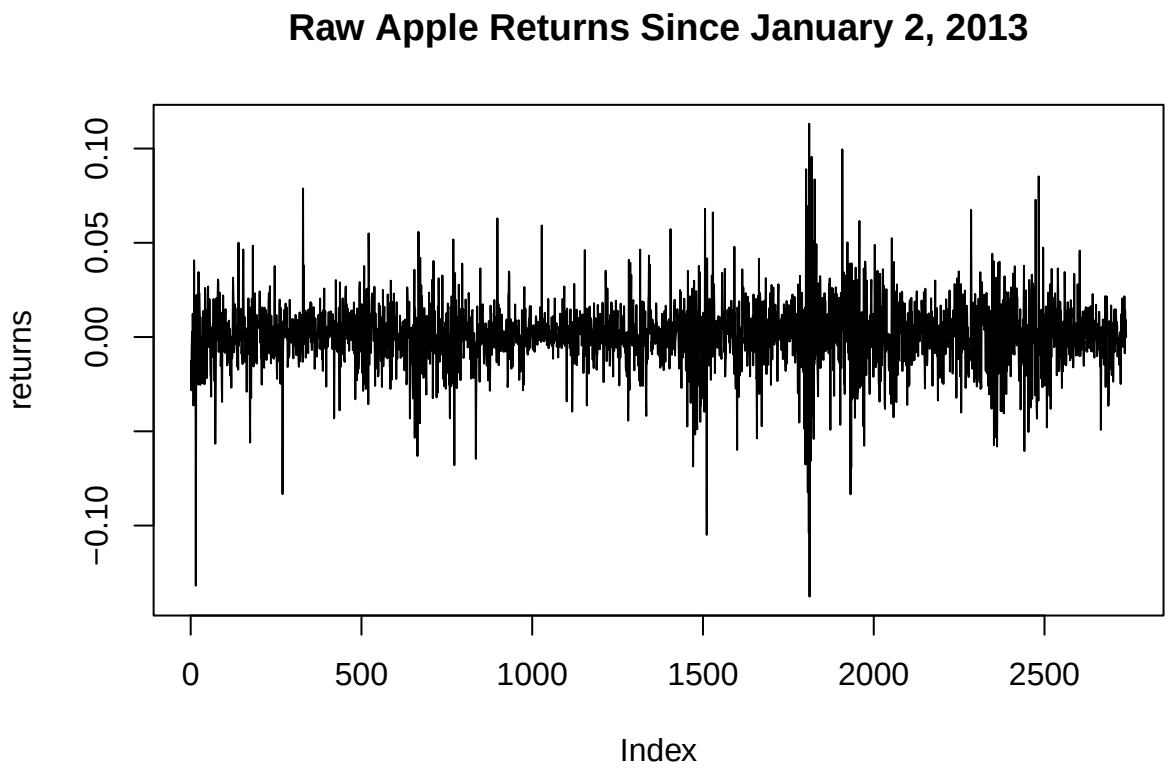
7 Appendix

7.1 Additional Exploratory Data Analysis

7.1.1 Raw Returns Analysis

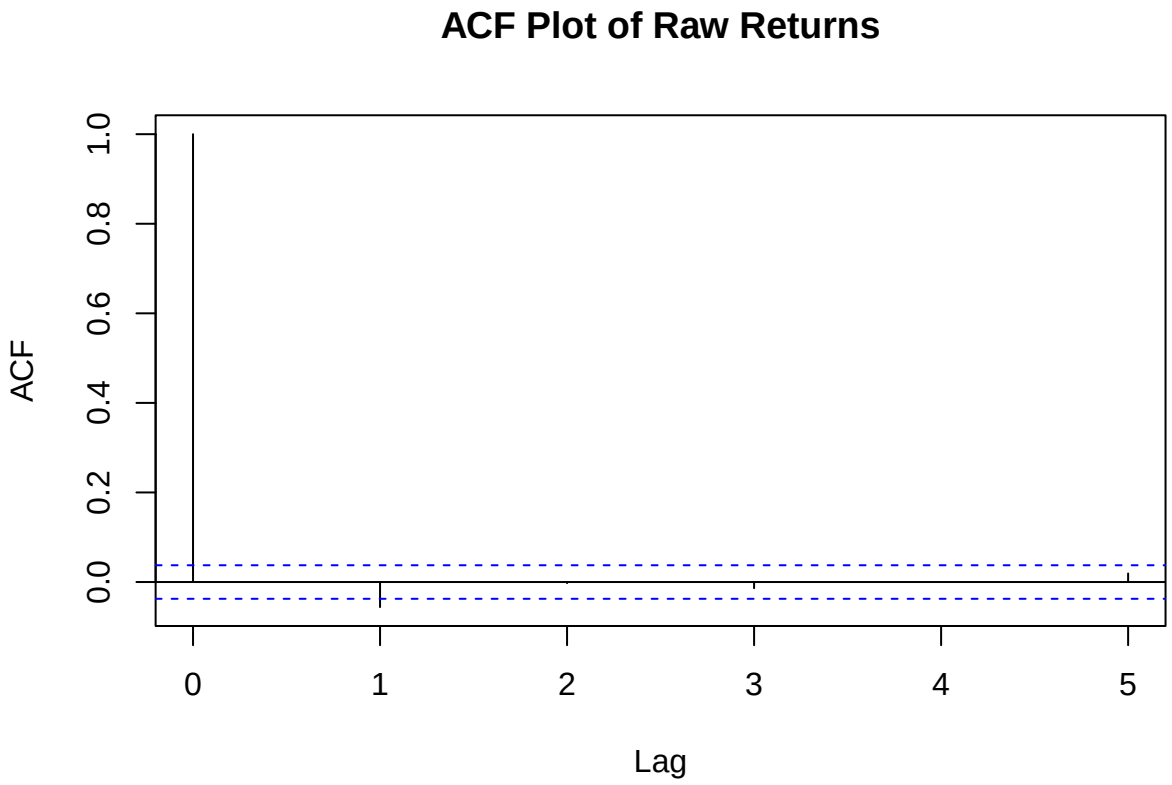
.

```
# Calculate raw closes
returns =diff(log(stocks$close_preamp))

# Plot time series
plot(returns, type="l", main="Raw Apple Returns Since January 2, 2013")
```



In the above figure, one can see the historical trend of raw returns for APPL since January 1, 2013. Taking the log of the returns is a common application in finance used for volatility analysis. Applying the 'diff' function calculates the difference between consecutive element. Within the context of this analysis, the function will compute the daily log returns by calculating the difference between each day's log closing price and the previous day closing's prices.

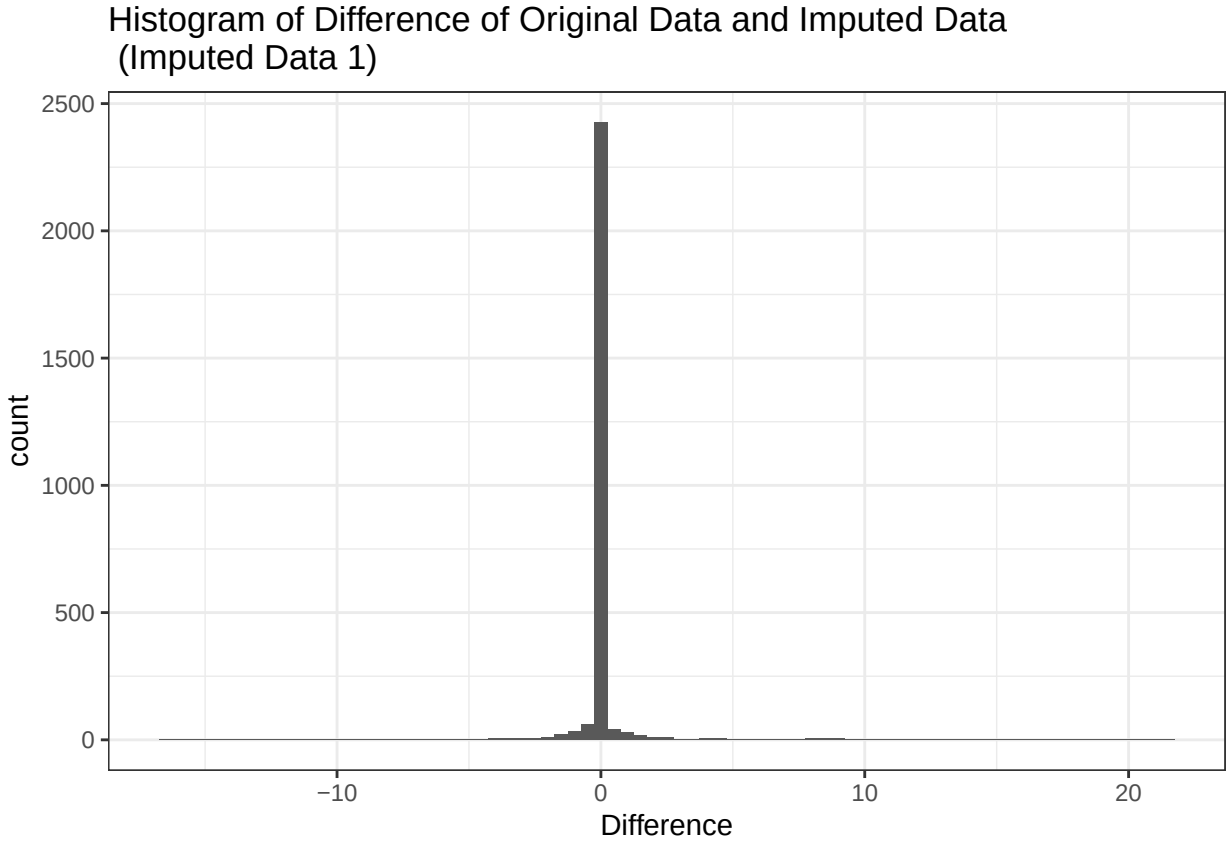
```r
# Calculate autocorrelations
acf_returns = acf(returns, lag.max=5, plot=FALSE)

# Plot ACF
plot(acf_returns, main="ACF Plot of Raw Returns")
```

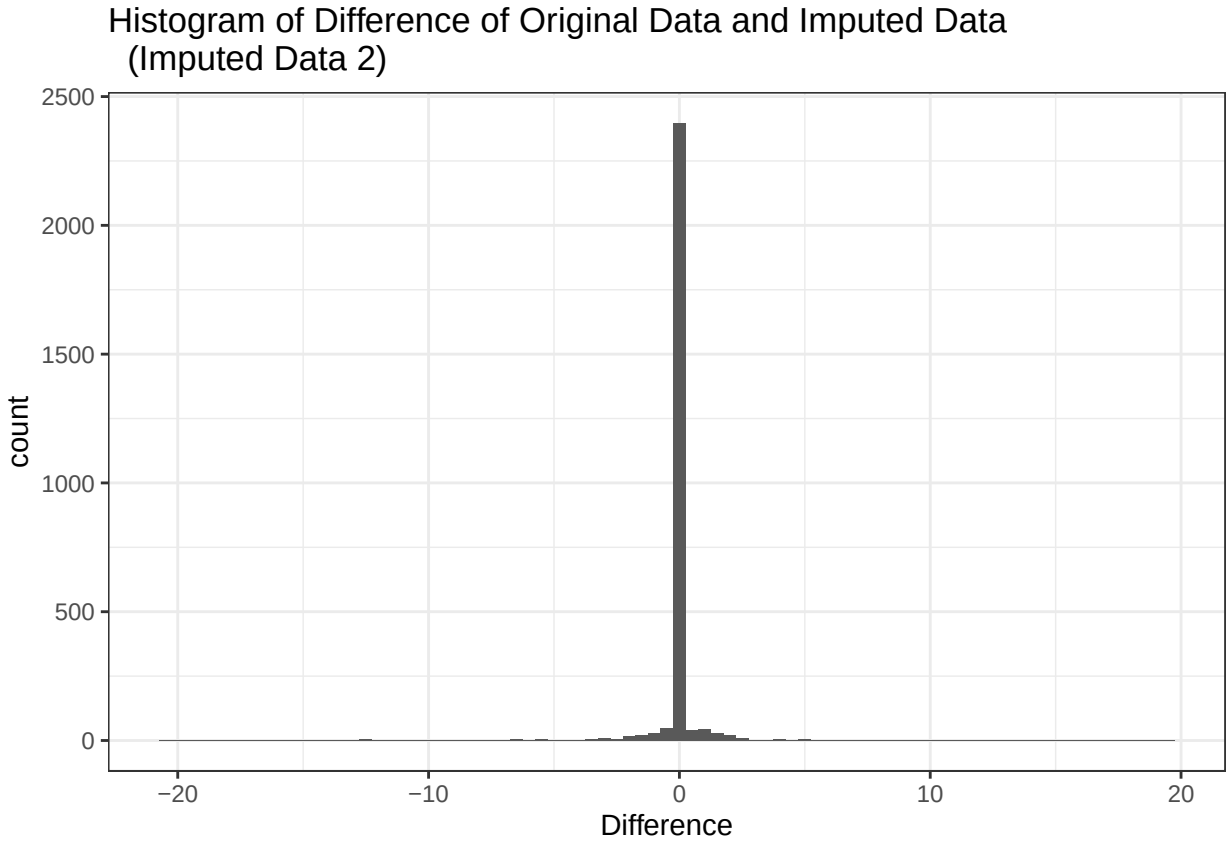


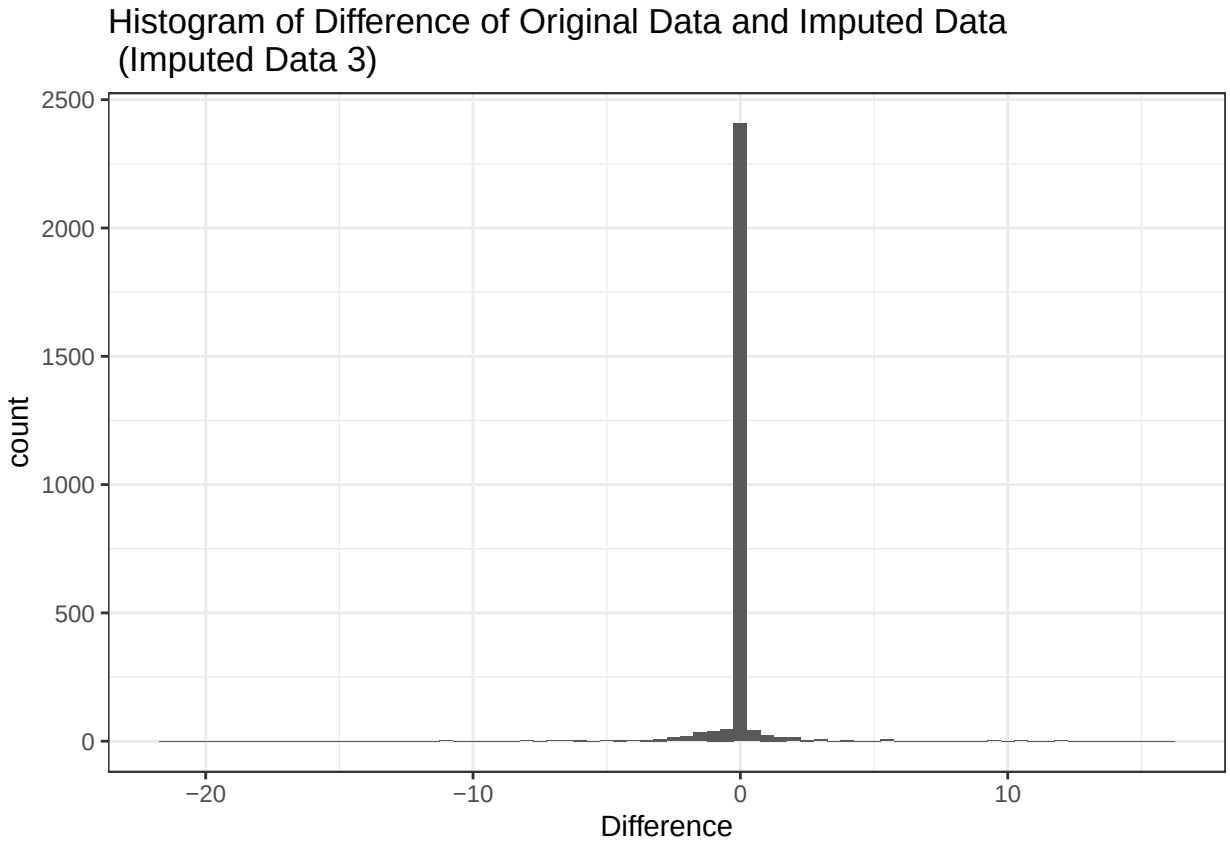
With the ACF plots shown in the above figure, the autocorrelation structure of the raw returns. With the autocorrelation falling within the blue horizontal bands, this would mean that the there is independence within the log returns.

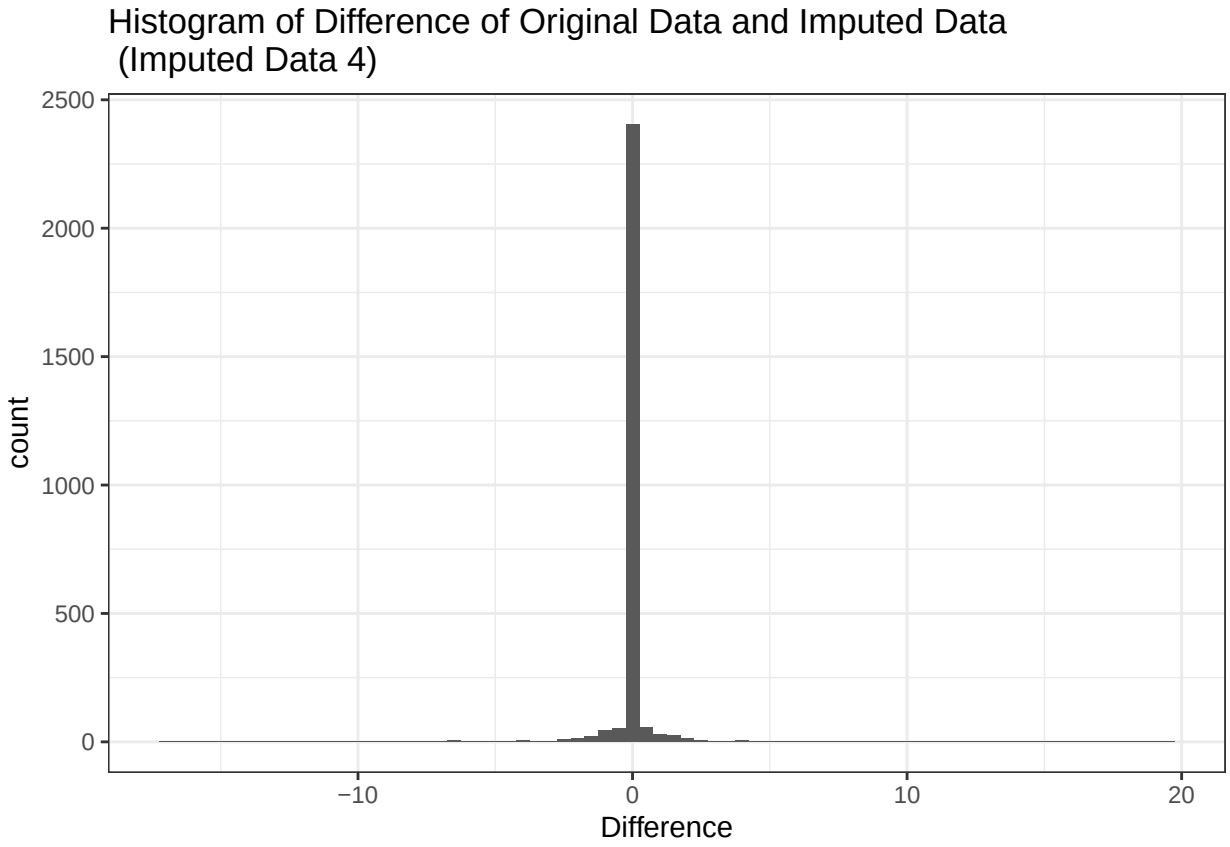
7.2 Imputed Data Results

7.2.1 Histograms for the Difference of Original Data and Imputed Data

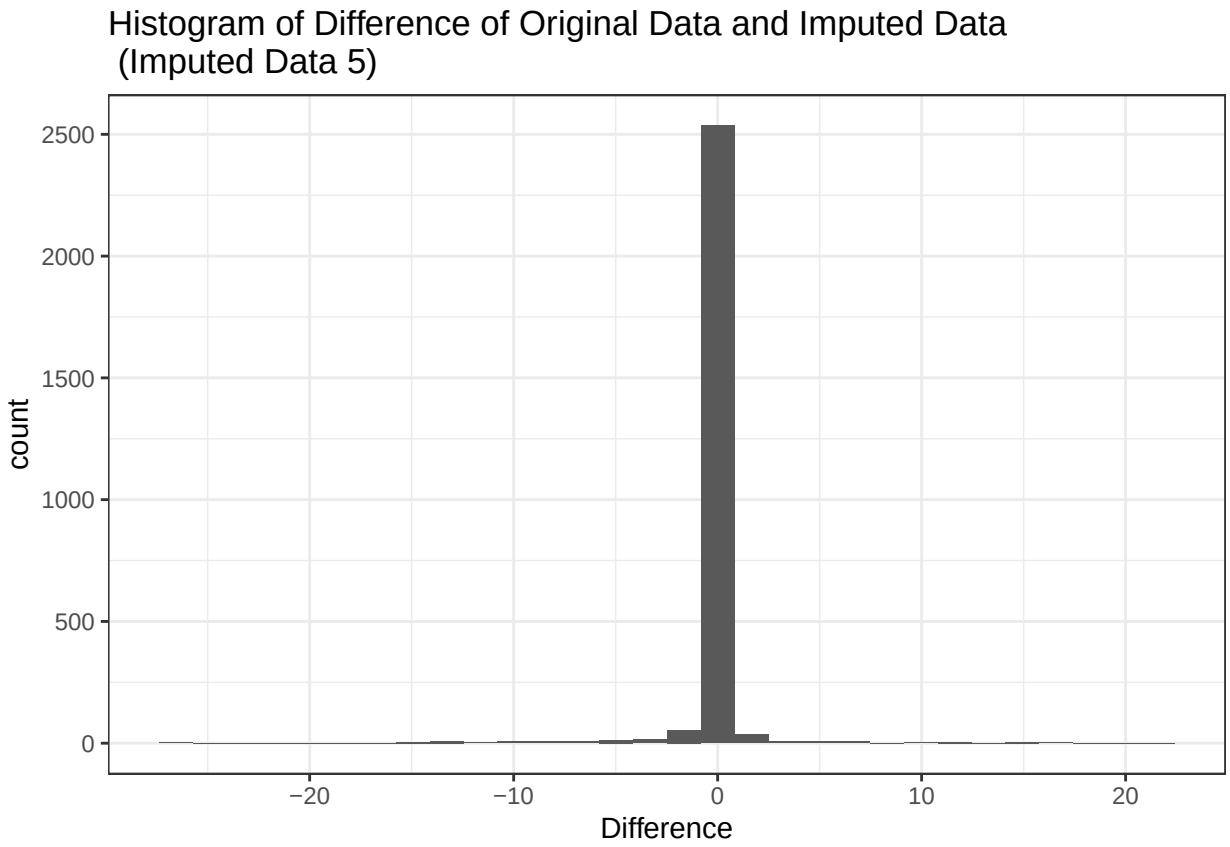




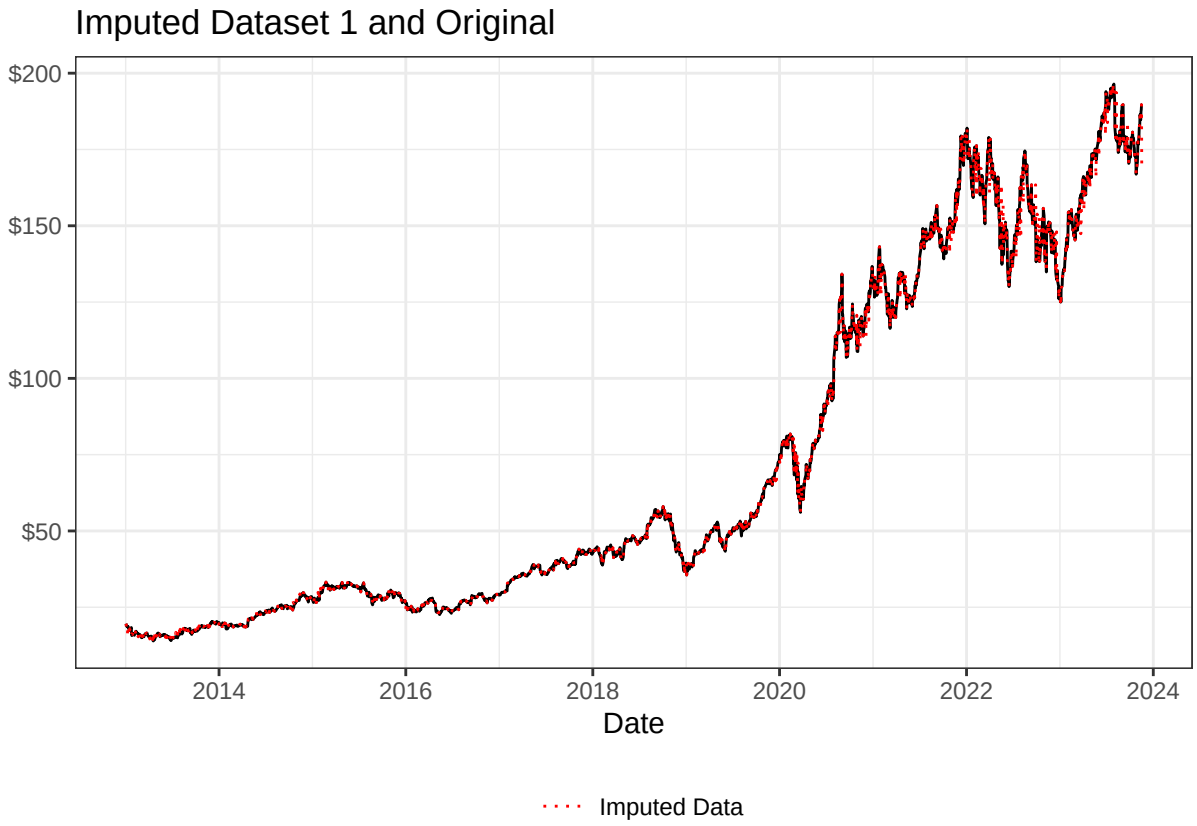


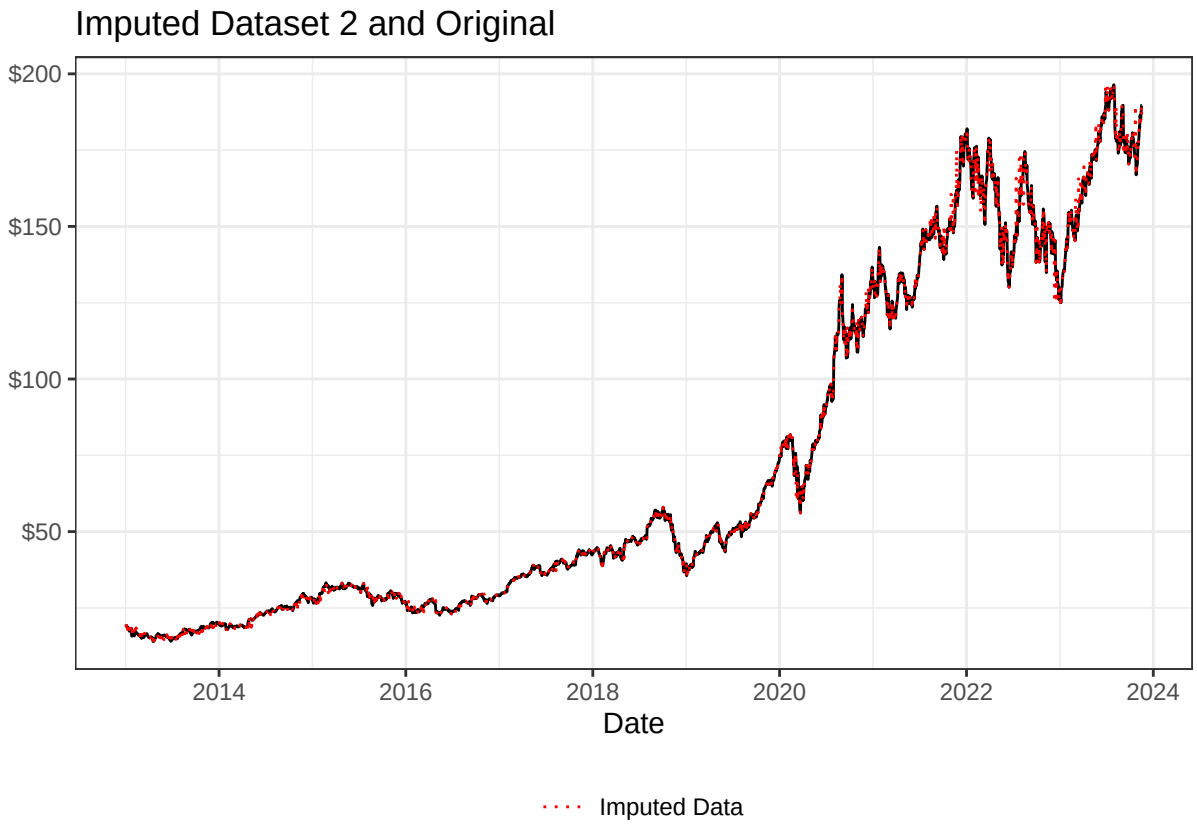


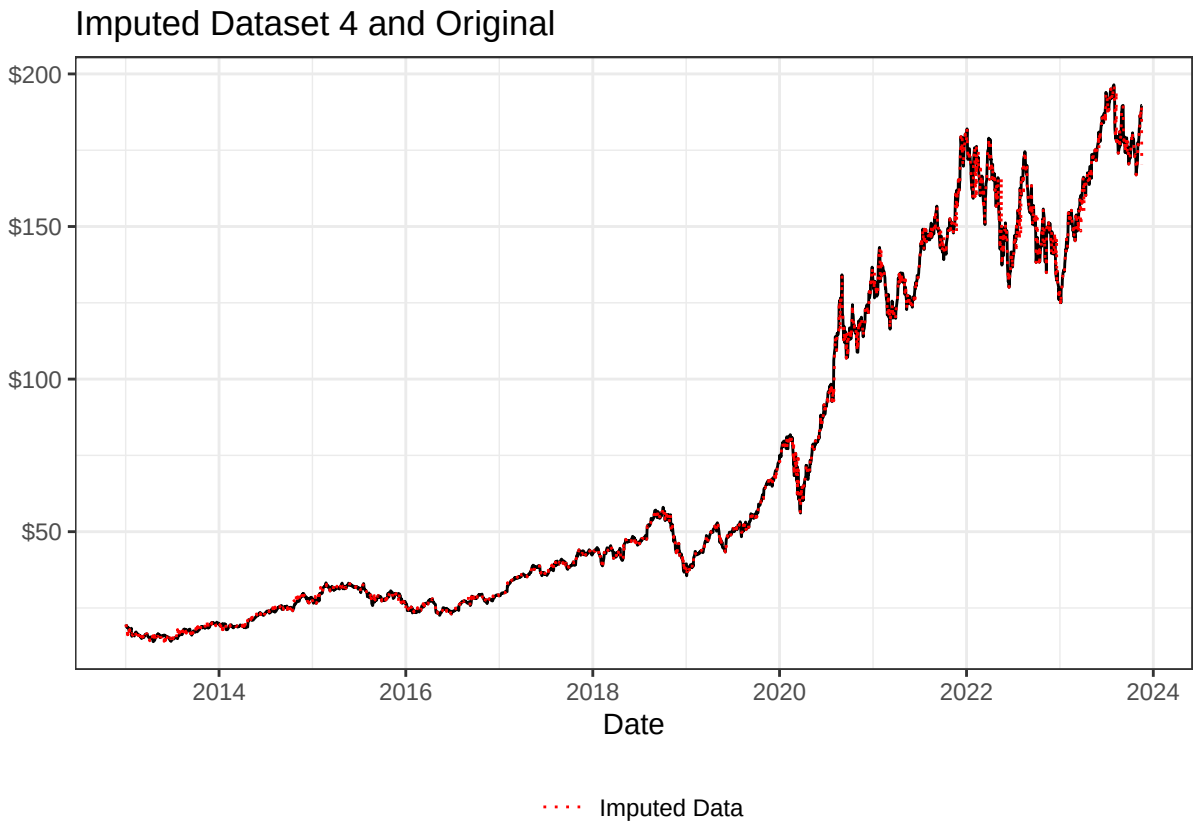

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

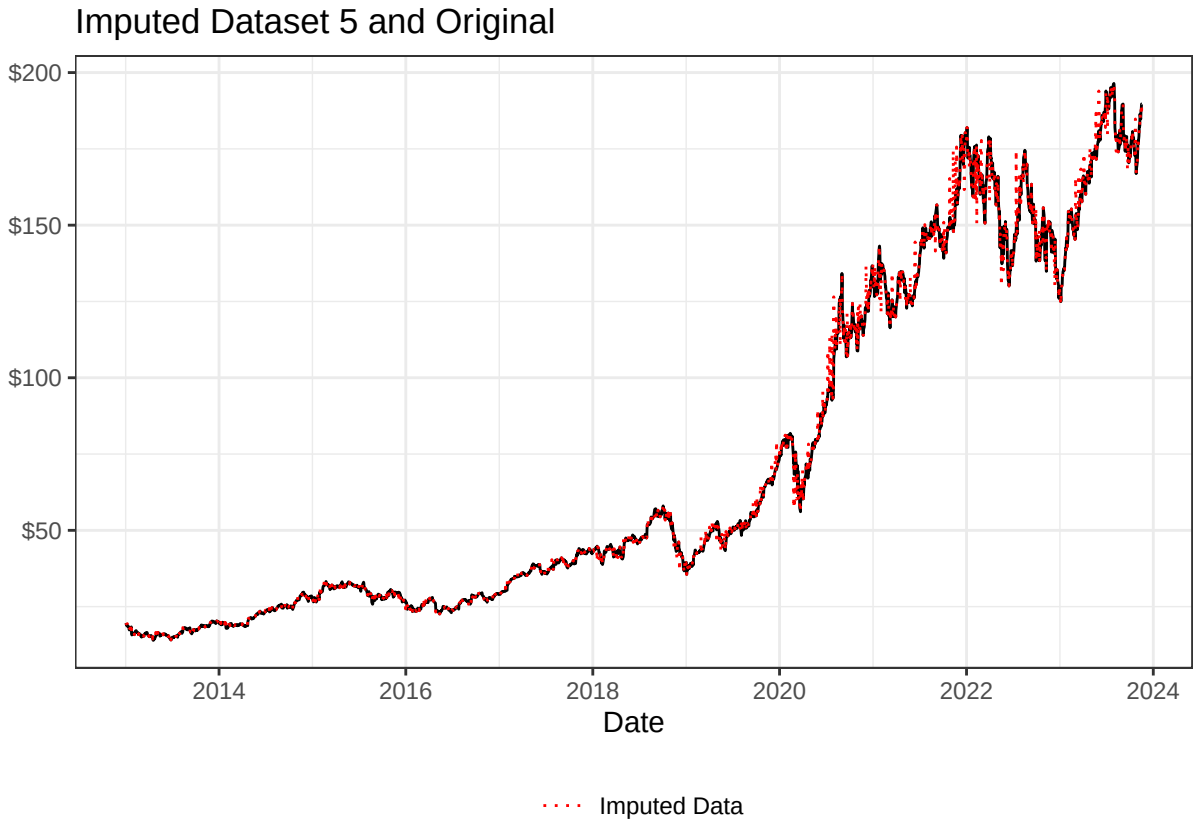

Histogram of Difference of Original Data and Imputed Data
(Imputed Data 5)
count
2500
2000
1500
1000
500
0
−20
−10
0
10
20
Difference

7.2.2 Graph of Imputed Data and Original Data






Imputed Dataset 4 and Original
$200
$150
$100
$50
2014
2016
2018
2020
2022
2024
Date
Imputed Data


Imputed Dataset 5 and Original
$200
$150
$100
$50
2014
2016
2018
2020
2022
2024
Date
Imputed Data

7.3 mice Readout

```
# Impute missing values using Predictive Mean Matching

stock_subset <- stocks[, c("symbol", "date", "close")]

imputed <- mice(data = stock_subset,
                m = 5, # Number of imputed datasets
                maxit = 25, # Max number of iterations
                method = 'pmm', # Imputation method
                seed = 555, # Set seed for reproducibility
                printFlag = TRUE
                )
```

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

```
##   5   2  close
##   5   3  close
##   5   4  close
##   5   5  close
##   6   1  close
##   6   2  close
##   6   3  close
##   6   4  close
##   6   5  close
##   7   1  close
##   7   2  close
##   7   3  close
##   7   4  close
##   7   5  close
##   8   1  close
##   8   2  close
##   8   3  close
##   8   4  close
##   8   5  close
##   9   1  close
##   9   2  close
##   9   3  close
##   9   4  close
##   9   5  close
##   10   1  close
##   10   2  close
##   10   3  close
##   10   4  close
##   10   5  close
##   11   1  close
##   11   2  close
##   11   3  close
##   11   4  close
##   11   5  close
##   12   1  close
##   12   2  close
##   12   3  close
```

```
##   12   4  close
##   12   5  close
##   13   1  close
##   13   2  close
##   13   3  close
##   13   4  close
##   13   5  close
##   14   1  close
##   14   2  close
##   14   3  close
##   14   4  close
##   14   5  close
##   15   1  close
##   15   2  close
##   15   3  close
##   15   4  close
##   15   5  close
##   16   1  close
##   16   2  close
##   16   3  close
##   16   4  close
##   16   5  close
##   17   1  close
##   17   2  close
##   17   3  close
##   17   4  close
##   17   5  close
##   18   1  close
##   18   2  close
##   18   3  close
##   18   4  close
##   18   5  close
##   19   1  close
##   19   2  close
##   19   3  close
##   19   4  close
##   19   5  close
```

```
##   20   1  close
##   20   2  close
##   20   3  close
##   20   4  close
##   20   5  close
##   21   1  close
##   21   2  close
##   21   3  close
##   21   4  close
##   21   5  close
##   22   1  close
##   22   2  close
##   22   3  close
##   22   4  close
##   22   5  close
##   23   1  close
##   23   2  close
##   23   3  close
##   23   4  close
##   23   5  close
##   24   1  close
##   24   2  close
##   24   3  close
##   24   4  close
##   24   5  close
##   25   1  close
##   25   2  close
##   25   3  close
##   25   4  close
##   25   5  close

## Warning: Number of logged events: 1
```

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

#View(imp_datasets)
```

7.4 brms Readout

```r
# 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]]
  # Create lag variable
  dataset$lagged_close <- c(NA, head(dataset$close, -1))
  fit <- tryCatch(
    {
      model <- brm(close ~ lagged_close, data = dataset,
                   iter = 750, 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
}
```

```
##
## 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 / 750 [  0%]  (Warmup)
## Chain 1: Iteration:  75 / 750 [ 10%]  (Warmup)
## Chain 1: Iteration: 150 / 750 [ 20%]  (Warmup)
## Chain 1: Iteration: 225 / 750 [ 30%]  (Warmup)
## Chain 1: Iteration: 300 / 750 [ 40%]  (Warmup)
```

```
## Chain 1: Iteration: 375 / 750 [ 50%]  (Warmup)
## Chain 1: Iteration: 450 / 750 [ 60%]  (Warmup)
## Chain 1: Iteration: 501 / 750 [ 66%]  (Sampling)
## Chain 1: Iteration: 575 / 750 [ 76%]  (Sampling)
## Chain 1: Iteration: 650 / 750 [ 86%]  (Sampling)
## Chain 1: Iteration: 725 / 750 [ 96%]  (Sampling)
## Chain 1: Iteration: 750 / 750 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.078 seconds (Warm-up)
## Chain 1:                0.035 seconds (Sampling)
## Chain 1:                0.113 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.1 seconds
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:   1 / 750 [  0%]  (Warmup)
## Chain 2: Iteration:  75 / 750 [ 10%]  (Warmup)
## Chain 2: Iteration: 150 / 750 [ 20%]  (Warmup)
## Chain 2: Iteration: 225 / 750 [ 30%]  (Warmup)
## Chain 2: Iteration: 300 / 750 [ 40%]  (Warmup)
## Chain 2: Iteration: 375 / 750 [ 50%]  (Warmup)
## Chain 2: Iteration: 450 / 750 [ 60%]  (Warmup)
## Chain 2: Iteration: 501 / 750 [ 66%]  (Sampling)
## Chain 2: Iteration: 575 / 750 [ 76%]  (Sampling)
## Chain 2: Iteration: 650 / 750 [ 86%]  (Sampling)
## Chain 2: Iteration: 725 / 750 [ 96%]  (Sampling)
## Chain 2: Iteration: 750 / 750 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.07 seconds (Warm-up)
## Chain 2:                0.044 seconds (Sampling)
## Chain 2:                0.114 seconds (Total)
## Chain 2:
```

```
##
## 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 / 750 [  0%]  (Warmup)
## Chain 1: Iteration:  75 / 750 [ 10%]  (Warmup)
## Chain 1: Iteration: 150 / 750 [ 20%]  (Warmup)
## Chain 1: Iteration: 225 / 750 [ 30%]  (Warmup)
## Chain 1: Iteration: 300 / 750 [ 40%]  (Warmup)
## Chain 1: Iteration: 375 / 750 [ 50%]  (Warmup)
## Chain 1: Iteration: 450 / 750 [ 60%]  (Warmup)
## Chain 1: Iteration: 501 / 750 [ 66%]  (Sampling)
## Chain 1: Iteration: 575 / 750 [ 76%]  (Sampling)
## Chain 1: Iteration: 650 / 750 [ 86%]  (Sampling)
## Chain 1: Iteration: 725 / 750 [ 96%]  (Sampling)
## Chain 1: Iteration: 750 / 750 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.083 seconds (Warm-up)
## Chain 1:                0.03 seconds (Sampling)
## Chain 1:                0.113 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 / 750 [  0%]  (Warmup)
## Chain 2: Iteration:  75 / 750 [ 10%]  (Warmup)
## Chain 2: Iteration: 150 / 750 [ 20%]  (Warmup)
## Chain 2: Iteration: 225 / 750 [ 30%]  (Warmup)
```

```
## Chain 2: Iteration: 300 / 750 [ 40%]  (Warmup)
## Chain 2: Iteration: 375 / 750 [ 50%]  (Warmup)
## Chain 2: Iteration: 450 / 750 [ 60%]  (Warmup)
## Chain 2: Iteration: 501 / 750 [ 66%]  (Sampling)
## Chain 2: Iteration: 575 / 750 [ 76%]  (Sampling)
## Chain 2: Iteration: 650 / 750 [ 86%]  (Sampling)
## Chain 2: Iteration: 725 / 750 [ 96%]  (Sampling)
## Chain 2: Iteration: 750 / 750 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.076 seconds (Warm-up)
## Chain 2:                0.039 seconds (Sampling)
## Chain 2:                0.115 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 4.3e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.43 seconds
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 750 [  0%]  (Warmup)
## Chain 1: Iteration:  75 / 750 [ 10%]  (Warmup)
## Chain 1: Iteration: 150 / 750 [ 20%]  (Warmup)
## Chain 1: Iteration: 225 / 750 [ 30%]  (Warmup)
## Chain 1: Iteration: 300 / 750 [ 40%]  (Warmup)
## Chain 1: Iteration: 375 / 750 [ 50%]  (Warmup)
## Chain 1: Iteration: 450 / 750 [ 60%]  (Warmup)
## Chain 1: Iteration: 501 / 750 [ 66%]  (Sampling)
## Chain 1: Iteration: 575 / 750 [ 76%]  (Sampling)
## Chain 1: Iteration: 650 / 750 [ 86%]  (Sampling)
## Chain 1: Iteration: 725 / 750 [ 96%]  (Sampling)
## Chain 1: Iteration: 750 / 750 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.083 seconds (Warm-up)
## Chain 1:                0.044 seconds (Sampling)
## Chain 1:                0.127 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 / 750 [  0%]  (Warmup)
## Chain 2: Iteration:  75 / 750 [ 10%]  (Warmup)
## Chain 2: Iteration: 150 / 750 [ 20%]  (Warmup)
## Chain 2: Iteration: 225 / 750 [ 30%]  (Warmup)
## Chain 2: Iteration: 300 / 750 [ 40%]  (Warmup)
## Chain 2: Iteration: 375 / 750 [ 50%]  (Warmup)
## Chain 2: Iteration: 450 / 750 [ 60%]  (Warmup)
## Chain 2: Iteration: 501 / 750 [ 66%]  (Sampling)
## Chain 2: Iteration: 575 / 750 [ 76%]  (Sampling)
## Chain 2: Iteration: 650 / 750 [ 86%]  (Sampling)
## Chain 2: Iteration: 725 / 750 [ 96%]  (Sampling)
## Chain 2: Iteration: 750 / 750 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.077 seconds (Warm-up)
## Chain 2:                0.036 seconds (Sampling)
## Chain 2:                0.113 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 5.5e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.55 seconds
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 750 [  0%]  (Warmup)
## Chain 1: Iteration:  75 / 750 [ 10%]  (Warmup)
## Chain 1: Iteration: 150 / 750 [ 20%]  (Warmup)
```

```
## Chain 1: Iteration: 225 / 750 [ 30%]  (Warmup)
## Chain 1: Iteration: 300 / 750 [ 40%]  (Warmup)
## Chain 1: Iteration: 375 / 750 [ 50%]  (Warmup)
## Chain 1: Iteration: 450 / 750 [ 60%]  (Warmup)
## Chain 1: Iteration: 501 / 750 [ 66%]  (Sampling)
## Chain 1: Iteration: 575 / 750 [ 76%]  (Sampling)
## Chain 1: Iteration: 650 / 750 [ 86%]  (Sampling)
## Chain 1: Iteration: 725 / 750 [ 96%]  (Sampling)
## Chain 1: Iteration: 750 / 750 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.083 seconds (Warm-up)
## Chain 1:                0.035 seconds (Sampling)
## Chain 1:                0.118 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.1 seconds
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:   1 / 750 [  0%]  (Warmup)
## Chain 2: Iteration:  75 / 750 [ 10%]  (Warmup)
## Chain 2: Iteration: 150 / 750 [ 20%]  (Warmup)
## Chain 2: Iteration: 225 / 750 [ 30%]  (Warmup)
## Chain 2: Iteration: 300 / 750 [ 40%]  (Warmup)
## Chain 2: Iteration: 375 / 750 [ 50%]  (Warmup)
## Chain 2: Iteration: 450 / 750 [ 60%]  (Warmup)
## Chain 2: Iteration: 501 / 750 [ 66%]  (Sampling)
## Chain 2: Iteration: 575 / 750 [ 76%]  (Sampling)
## Chain 2: Iteration: 650 / 750 [ 86%]  (Sampling)
## Chain 2: Iteration: 725 / 750 [ 96%]  (Sampling)
## Chain 2: Iteration: 750 / 750 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.076 seconds (Warm-up)
## Chain 2:                0.033 seconds (Sampling)
```

```
## Chain 2:                0.109 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 4.1e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.41 seconds
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 750 [  0%]  (Warmup)
## Chain 1: Iteration:  75 / 750 [ 10%]  (Warmup)
## Chain 1: Iteration: 150 / 750 [ 20%]  (Warmup)
## Chain 1: Iteration: 225 / 750 [ 30%]  (Warmup)
## Chain 1: Iteration: 300 / 750 [ 40%]  (Warmup)
## Chain 1: Iteration: 375 / 750 [ 50%]  (Warmup)
## Chain 1: Iteration: 450 / 750 [ 60%]  (Warmup)
## Chain 1: Iteration: 501 / 750 [ 66%]  (Sampling)
## Chain 1: Iteration: 575 / 750 [ 76%]  (Sampling)
## Chain 1: Iteration: 650 / 750 [ 86%]  (Sampling)
## Chain 1: Iteration: 725 / 750 [ 96%]  (Sampling)
## Chain 1: Iteration: 750 / 750 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.059 seconds (Warm-up)
## Chain 1:                0.029 seconds (Sampling)
## Chain 1:                0.088 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 / 750 [  0%]  (Warmup)
## Chain 2: Iteration:  75 / 750 [ 10%]  (Warmup)
```

```
## Chain 2: Iteration: 150 / 750 [ 20%]  (Warmup)
## Chain 2: Iteration: 225 / 750 [ 30%]  (Warmup)
## Chain 2: Iteration: 300 / 750 [ 40%]  (Warmup)
## Chain 2: Iteration: 375 / 750 [ 50%]  (Warmup)
## Chain 2: Iteration: 450 / 750 [ 60%]  (Warmup)
## Chain 2: Iteration: 501 / 750 [ 66%]  (Sampling)
## Chain 2: Iteration: 575 / 750 [ 76%]  (Sampling)
## Chain 2: Iteration: 650 / 750 [ 86%]  (Sampling)
## Chain 2: Iteration: 725 / 750 [ 96%]  (Sampling)
## Chain 2: Iteration: 750 / 750 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.056 seconds (Warm-up)
## Chain 2:                0.026 seconds (Sampling)
## Chain 2:                0.082 seconds (Total)
## Chain 2:
```