Project 2

Nigel Bastian Cendra (s2610920)

1 Part 1: 3D printer

1.1 Compute Probabilistic Predictions of Actual Weight

In this case, we have a model that doesn't fit the basic regression system where we have a constant variance for all the observation, but instead we have a linear model for the expected value and log linear model for the variance. Hence, we need to figure out the parameters since we don't have a closed form expression for the estimates and we also need to figure out it's predictive distribution.

Let

$$Z_E = \begin{bmatrix} 1 & x & 0 & 0 \\ 1 & x & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x & 0 & 0 \end{bmatrix} \in \mathbb{R}^{86\times 4}$$

be the matrix of observation, where the vector of observation expectation can be written as $E_{y|\beta}(y) = Z_E\beta$ where $\beta = [\beta 1, \beta 2, \beta 3, \beta 4]$

The predicted weight for new observations based on model A and B is given by:

$\hat{y} = X\hat{\beta}$

where X is the design matrix for the new data points and $\hat{\beta}$ is the vector of estimated coefficients from the regression model

then we can get the mean and variance of each predicted observation for model A and B by:

$\hat{\mu}_A = exp(\hat{\beta}_1 + \hat{\beta}_2 x), \hat{\mu}_B = exp(\hat{\beta}_1 + \hat{\beta}_2 x)$

$\hat{\sigma}_A^2 = exp(\hat{\beta}_3 + \hat{\beta}_4 x), \hat{\sigma}_B^2 = exp(\hat{\beta}_3) + exp(\hat{\beta}_4 x^2)$

Then from both model, the predictions for new data points assume a normal distribution centered around the mean prediction with variance reflecting the prediction error.

The prediction error ($\hat{\epsilon}$):

The prediction error for a new observation is the difference between the actual value and predicted value

$\hat{\epsilon}_i = y_i - \hat{y}_i \sim N(0, Var(y_i) + Var(\hat{y}_i))$

where we get the variance $= Var(y_i) + Var(\hat{y}_i)$ due to the independence of error term and the predicted value.

Computing the prediction interval:

To get the 95% prediction interval, we can find the lower and upper bound by:

Lower Bound $= \hat{y}_i - z_{0.975}\sqrt{Var(y_i) + Var(\hat{y}_i)}$

Upper Bound $= \hat{y}_i + z_{0.975}\sqrt{Var(y_i) + Var(\hat{y}_i)}$

```r
load("filament1.rda")
#model A
predictions_model_A <- filament1_predict(filament1, "A", filament1)
#model B
predictions_model_B <- filament1_predict(filament1, "B", filament1)
```

1.1.1 Inspecting the predictions results

```r
head(predictions_model_A)
```

1.1.1.1 Model A

```
##        mean        sd       lwr       upr
## 1 35.524602 0.9815810 33.600738 37.448465
## 2 43.079083 1.1941889 40.738516 45.419650
## 3 21.494851 0.6859387 20.150436 22.839266
## 4 47.395929 1.3366028 44.776236 50.015623
## 5 19.336428 0.6499262 18.062596 20.610260
## 6  3.148254 0.4460295  2.274052  4.022456
```

```r
head(predictions_model_B)
```

1.1.1.2 Model B

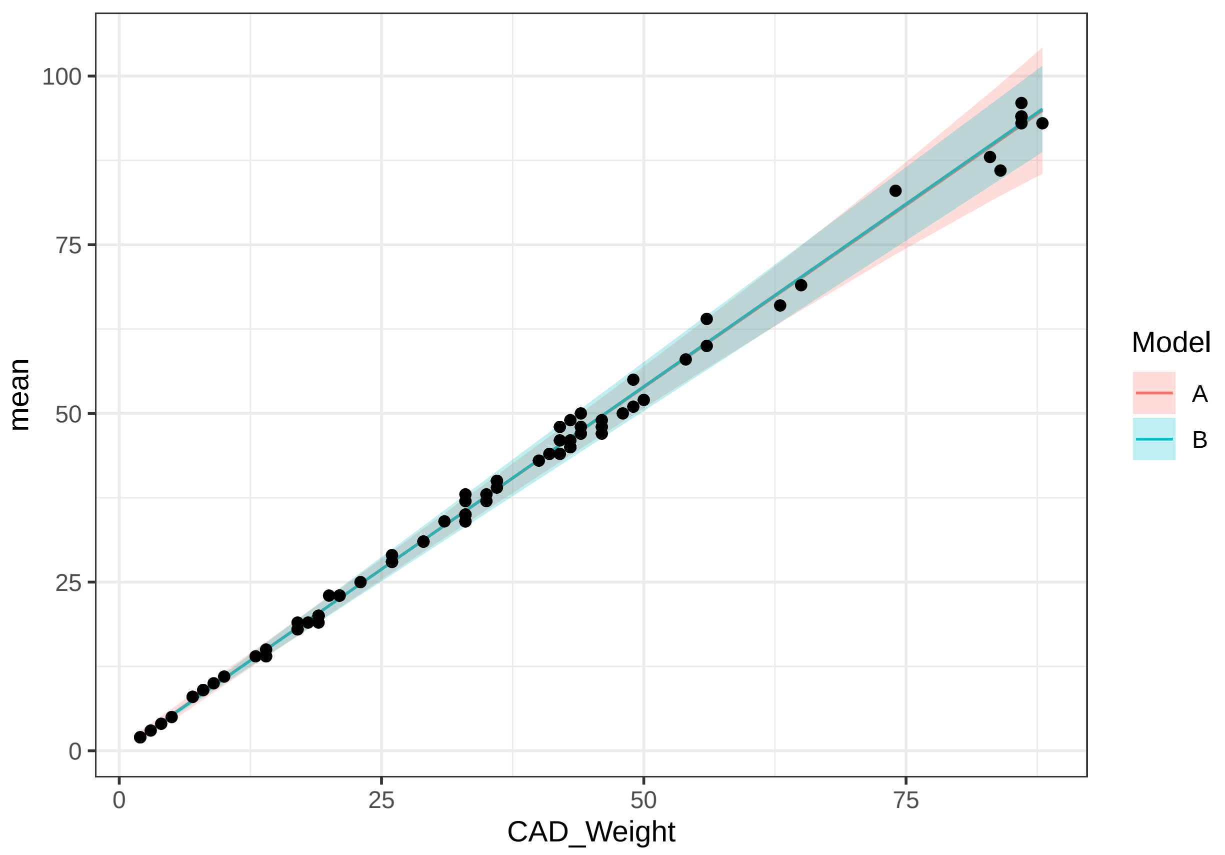
```
##        mean        sd      lwr      upr
## 1 35.573227 1.2237897 33.17464 37.97181
## 2 43.153278 1.4836946 40.24529 46.06127
## 3 21.495990 0.7412875 20.04309 22.94889
## 4 47.484736 1.6322254 44.28563 50.68384
## 5 19.330261 0.6671021 18.02277 20.63776
## 6  3.087295 0.1149282  2.86204  3.31255
```

1.1.2 Inspecting the Predictions Visually

```r
library(ggplot2)

#combine prediction data with filament1 data
predictions_combined <- rbind(cbind(predictions_model_A, filament1, Model = "A"),
                              cbind(predictions_model_B, filament1, Model = "B"))

# Use ggplot to create the plot
ggplot(predictions_combined, aes(x = CAD_Weight)) +
  geom_line(aes(y = mean, color = Model)) +
  geom_ribbon(aes(ymin = lwr, ymax = upr, fill = Model), alpha = 0.25) +
  geom_point(aes(y = Actual_Weight), data = filament1) +
  labs(color = "Model", fill = "Model")
```



Looking at the figure, the prediction line seems to pass through the center of the data points well, indicating that both models have captured the central tendency of the data. However, it seems that model B have a narrower prediction intervals compared to model A. This suggests that model B is more certain about its predictions compared to model A. Hence, look at this preliminary analysis of the figure above, model B may be better at predicting the weight better than model A. Further analysis will be done below.

1.2 Evaluating the Squared Error (ES) and Dawid-Sebastiani (DS) Scores

In this section, we create functions to evaluate the squared error (ES) and Dawid-Sebastiani (DS) scores, which have the following formulas:

- Squared Error (ES) Score:

$$SE = (y - \hat{y}_F)^2$$

and we can get average Squared Error by:

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_{F_i})^2$$

- Dawid-Sebastiani (DS) Score:

$$DS = \frac{(y - \hat{y}_F)^2}{\sigma^2} + log(\sigma_F^2)$$

and we can get the average Dawid_Sebastiani Score by:

$$\frac{1}{N}\sum_{i=1}^{N}\frac{(y_i - \hat{y}_{F_i})^2}{\sigma_i^2} + log(\sigma_{F_i}^2)$$

Model A Scores:

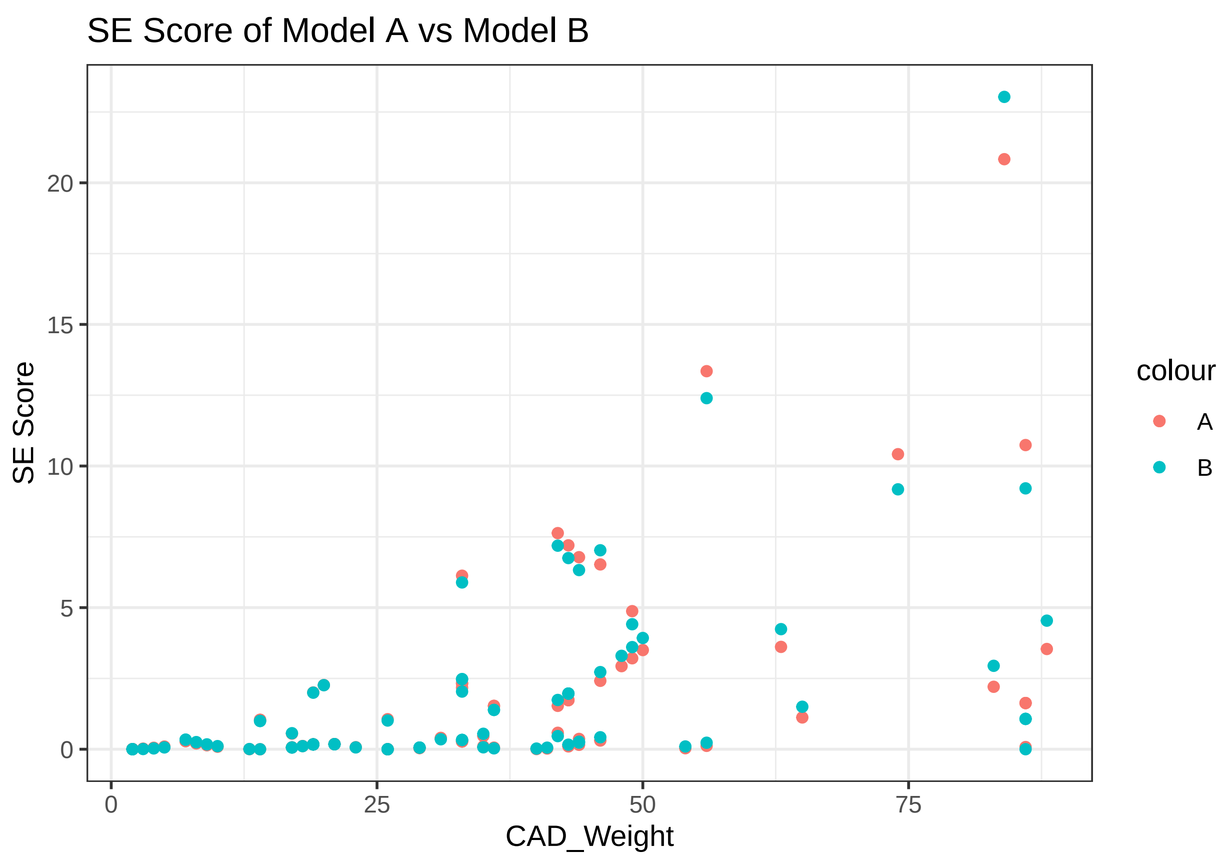
```r
score_A <- fil_score(data = filament1, model = "A", newdata = filament1)
head(score_A)
```

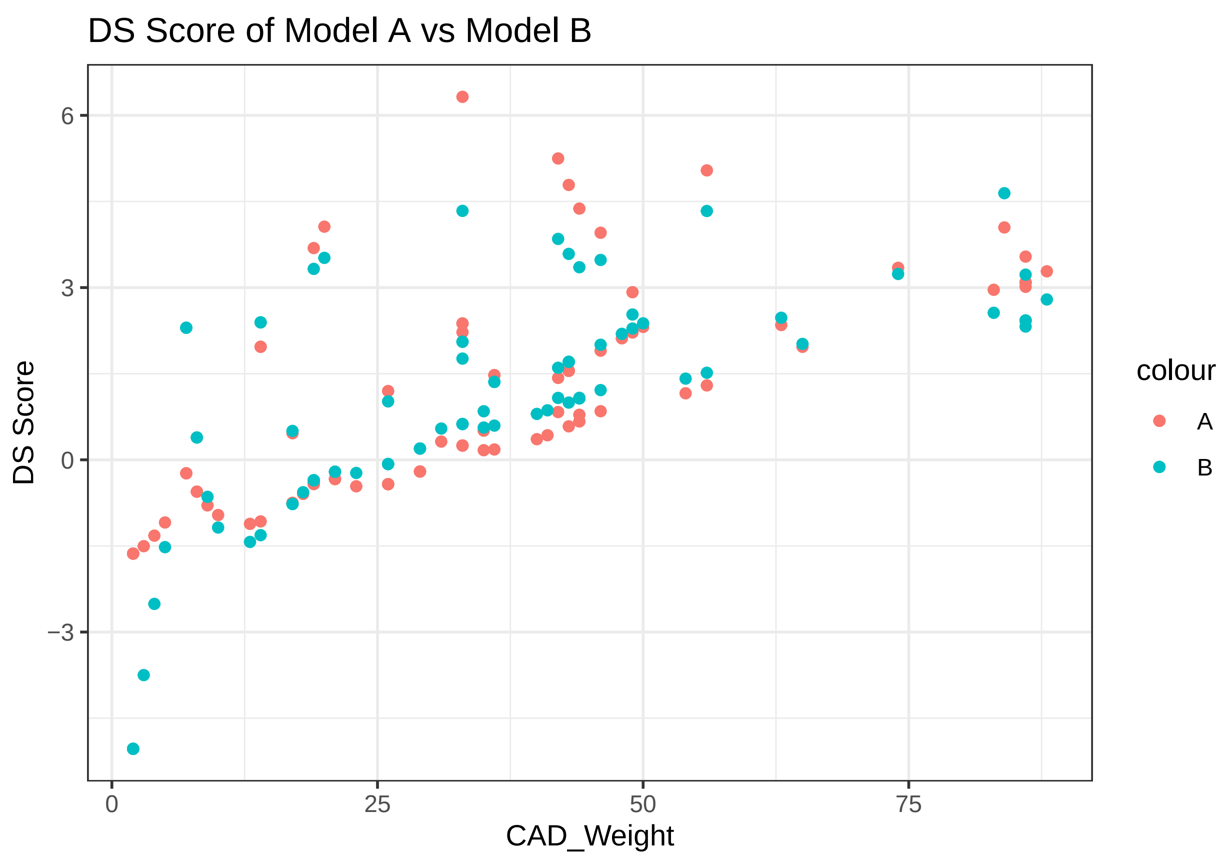
```
##        mean        sd       lwr       upr          se         ds
## 1 35.524602 0.9815810 33.600738 37.448465 6.127596148  6.3225368
## 2 43.079083 1.1941889 40.738516 45.419650 0.006254123  0.3593200
## 3 21.494851 0.6859387 20.150436 22.839266 2.265473411  4.0609784
## 4 47.395929 1.3366028 44.776236 50.015623 0.156760097  0.6680090
## 5 19.336428 0.6499262 18.062596 20.610260 0.113183689 -0.5938416
## 6  3.148254 0.4460295  2.274052  4.022456 0.021979204 -1.5042600
```

Model B Scores:

```r
score_B <- fil_score(data = filament1, model = "B", newdata = filament1)
head(score_B)
```

```
##        mean        sd      lwr      upr         se         ds
## 1 35.573227 1.2237897 33.17464 37.97181 5.88922495  4.3361857
## 2 43.153278 1.4836946 40.24529 46.06127 0.02349426  0.7997433
## 3 21.495990 0.7412875 20.04309 22.94889 2.26204588  3.5177662
## 4 47.484736 1.6322254 44.28563 50.68384 0.23496901  1.0680850
## 5 19.330261 0.6671021 18.02277 20.63776 0.10907249 -0.5645316
## 6  3.087295 0.1149282  2.86204  3.31255 0.00762042 -3.7499619
```


SE Score of Model A vs Model B
SE Score
CAD_Weight
colour
A
B
0
5
10
15
20
0
25
50
75



1.3 Leave One Cross Validation

```r
#leave1out result for model A
res_val_A <- leave1out(filament1, "A")
#leave1out result for model B
res_val_B <- leave1out(filament1, "B")
```

1.3.1 Leave-one-out Average Scores

```r
score_AB <- rbind(cbind(model = "A", se = mean(res_val_A$se), ds = mean(res_val_A$ds)),
                  cbind(model = "B", se = mean(res_val_B$se), ds = mean(res_val_B$ds)))

knitr::kable(score_AB)
```

model	se	ds
A	1.84173404598394	1.12834755606332
B	1.84199459829074	0.939715043805143

Interpretation:

- The se values are very similar for both models, indicating that in terms of average magnitude of errors, they perform similarly, with model A having SE score of 1.8417 and model B with 1.8420. The SE measures the average squared distance between the observed and predicted values, so lower values indicate better model performance. However, due to the slight difference in model A and B, it indicates no substantial difference in performance based on this metric.
- The DS scores here differs more noticeably between model A and B, with model A having DS score of 1.1283 and model B with 0.9397. The DS score is a metric that combines both the precision and accuracy of the predictions, with taking account both the variance of the predictions and squared error. A lower DS score indicates that the model prediction are both more accurate and more precise. In this case, model B have a lower DS score, suggesting that model B appears to be better model in terms of this metric compared to model A.

1.4 Monte Carlo Estimate

1.4.1 Exchangeability Test Discussion

For this part, we want to do the exchangeability test between the model predictions,

Where,

H_0: the scores of the two models are pairwise exchangeable

H_1: Model B is better than A

To do this, we can use the average $T(S_i^{\Delta}) = \frac{1}{N}\sum_{i=1}^{N}(S_i^A - S_i^B) = \frac{1}{N}\sum_{i=1}^{N}(S_i^{\Delta})$ as the test statistic and use the montecarlo estimate of the p value, where S_i^A is the score value of model A while S_i^B is the score value of model B.

More precisely, to construct a formal test, we randomise the scores within each pair. This means swapping the sign of the difference. Where we will use the test statistic mentioned above.

Then, for each $j = 1, ..., J$ and $i = 1, ..., N$, draw $S_i^{\Delta(j)} = S_i^{\Delta}$ with probability 0.5, and $-S_i^{\Delta}$ with probability 0.5. After this, we compute the test statistic $T^{(j)} = T(S_i^{\Delta(j)}, i = 1, ..., N)$. Where the average $\frac{1}{J}\sum_{i=1}^{J} I(T^{(j)} \geq T(S_i^{\Delta}))$ is an unbiased estimator of the one-sided p-value w.rt T for the hypothesis above. Please, note that I here is an indicator function (where the value is 1 if for $I(x)$, $x \in A$, 0 otherwise).

1.4.2 Monte Carlo Standard Error Discussion

Now, let us discuss on how to find the monte carlo standard error

we first deduce that the montecarlo estimate θ is given by:

$\hat{\theta} = \frac{1}{J}\sum_{j=1}^{J}\theta^{(j)}$

The variance of the estimate:

$Var(\hat{\theta}) = \frac{1}{J}\sum_{j=1}^{J} Var(\theta^{(j)}) = \frac{1}{J^2} J Var(\theta^{(j)}) = \frac{Var(\theta^{(j)})}{J}$ as the simulated values are i.i.d

and finally, to get the standard error of the estimate:

$SE(\hat{\theta}) = \sqrt{\frac{Var(\theta^{(j)}}{J}} = \frac{\sigma_{\hat{\theta^{(j)}}}}{\sqrt{J}}$

```r
score_diff <- data.frame(
  se_diff = res_val_A$se - res_val_B$se,
  ds_diff = res_val_A$ds - res_val_B$ds
)
```

```r
#test statistic
test_statistic <- data.frame(
  se = mean(score_diff$se_diff),
  ds = mean(score_diff$ds_diff)
)

set.seed(12345L)
#num of monte carlo simulation
J <- 10000

statistic <- data.frame(se = numeric(J), ds = numeric(J))

for (j in 1:J){
  random_sign <- sample(c(-1, 1), nrow(score_diff), replace = TRUE)
  statistic[j, "se"] <- mean(random_sign * score_diff$se_diff)
  statistic[j, "ds"] <- mean(random_sign * score_diff$ds_diff)

}

p_values <- data.frame(
  se = mean(statistic$se > test_statistic$se),
  ds = mean(statistic$ds > test_statistic$ds)
)


#sd
sd_se <- sd(statistic$se)
sd_ds <- sd(statistic$ds)
#se
se_se <- sd_se/sqrt(J)
se_ds <- sd_ds/sqrt(J)

se_values <- data.frame(
  se_standard_error = se_se,
  ds_standard_error = se_ds
)
```

1.4.3 Monte Carlo Standard Errors Result:

se_standard_error	ds_standard_error
0.0004726	0.0011228

The monte carlo standard error in our context shows the variability of the simulation estimates. It gives an idea of how much the estimated p values would vary if we were to repeat the entire monte carlo multiple times. For our case, we get a standard error of 0.0004726 for the SE score and 0.0011228 for the DS score, suggesting that the monte carlo estimates are quite precise since the monte carlo standard errors are pretty small. This means that, when conducting hypothesis testing, we can be more confident in the result of our p values. Hence, let us now conduct hypothesis testing to investigate whether one model is better at predicting than the other.

1.4.4 Hypothesis Testing: P Val Result

se	ds
0.4989	0.0436

Looking at the P value of both Squared Error and Dawid-Sebastiani scores, the se error term has a p value larger than the significant value (in most case, 0.05), while the ds has a lower p value than the significant value, with the se p values having a score of 0.4989 while the ds have p values of 0.0436. Hence, in terms of se score, the p value is insignificant, suggesting that the model are exchangeable in terms of se score or that we don't have enough evidence to conclude that model B is better than model A in terms of the se score. On the other hand, the p value of ds score (0.0436) is lower than the significant value (in most case, 0.05), meaning that the p value is significant, suggesting that in this case, the model B is better than model A in terms of ds score.

In conclusion, based on the SE score, you cannot conclude that one model is superior to the other. However, the DS score provides evidence that Model B might be the better model. It's important to consider that the DS score takes into account both the variance of the predictions and the scale of the predictions, making it a more comprehensive measure than the SE score.

2 Part 2: Archaeology in the Baltic sea

2.1 Evaluating the combined log-likelihood

```
#test the arch_loglike function
test_df <- data.frame(N = rgeom(1000, 1/1001), phi = rbeta(1000, 2, 2))
y = c(256,237)
head(arch_loglike(test_df, y))
```

```
##      N       phi log_likelihood
## 1  613 0.3580087      -12.54805
## 2   84 0.5495482           -Inf
## 3 2090 0.2284451     -175.21597
## 4  475 0.2321167     -190.30225
## 5   63 0.6568785           -Inf
## 6   68 0.2630601           -Inf
```

2.2 Estimation using Monte Carlo Methods

```
#estimate
monte_carlo_res <- estimate(y=c(237,256), xi=1/1001, a=0.5, b=0.5, K=10000)
```

Estimate Result:

py	EN_y	EP_y
8.2e-06	868.2038	0.4071623

2.2.1 Result Interpretation

From the table above we get:

- $p_Y(y)$ = 8.2e-06, and in this case, $p_Y(y)$ represent the estimated probability of observing the data y given the prior distributions for N and ϕ. This small value (8.2e-06) indicates that under the model and priors given, the probability of observing exactly 256 left and 237 femurs is really low.
- $E[N|y]$ = 868.2038, and in this case $E[N|y]$ represent the expected value of the total number people buried given the observed data y. The value indicates that the average number of people you would expect to be buried given the femur count and the priors are 868.2038. This value is lower than the 1000 individuals initially estimated by the archaeologist. It might mean that some of inividuals femurs were not found or that the initial estimate was too high.
- $E[\phi|y]$ = 0.4071623, and in this case $E[\phi|y]$ represent the probability of finding a femur. The value of the model estimate for the likelihood of finding a femur during the excavations are 0.4071623.

3 Code appendix

```r
#' Nigel Bastian Cendra, S2610920
#' Add your own function definitions on this file.

#' neg_log_lik
#
#' @description Evaluate the negated log-likelihood for model A and B
#' @param beta A vector with the beta parameters
#' @param data A `data.frame` with the same variables as the `filament1` data set.
#' Must have columns `CAD_Weight` and `Actual_Weight`
#' @param model Either "A" for a log-linear variance model, or "B" for a proportional
#' scaling error model

neg_log_lik <- function(beta, data, model){

  mu <- beta[1] + beta[2]*data[["CAD_Weight"]]

  # distinguish between the two models to find the particular standard deviation for the betas
  if(model == "A") {
    sigma <- sqrt(exp(beta[3] + beta[4]*data[["CAD_Weight"]]))
  }else{
    sigma <- sqrt(exp(beta[3])+exp(beta[4]) * (data[["CAD_Weight"]]^2))
  }
  - sum(dnorm(data[["Actual_Weight"]],
              mean = mu,
              sd=sigma,
              log = TRUE))

}

#' filament_estimate
#
#' @description Estimate filament models with different variance structure
#' @param data A `data.frame` with the same variables as the `filament1` data set.
#' Must have columns `CAD_Weight` and `Actual_Weight`
#' @param model Either "A" for a log-linear variance model, or "B" for a proportional
#' scaling error model
#' @return An estimation object suitable for use with [filament1_predict()]

filament1_estimate <- function(data, model) {
  model <- match.arg(model, c("A", "B"))
  if (model == "A") {
    beta_start <- c(-0.1, 1.07, -2, 0.05)
  } else {
    beta_start <- c(-0.15, 1.07, -13.5, -6.5)
  }
  opt <- optim(beta_start,
               neg_log_lik,
               data = data,
               model = model,
               hessian = TRUE,
               method = "Nelder-Mead",
               control = list(maxit = 5000)
  )
  fit <- list(
    model = model,
    par = opt$par,
    hessian = opt$hessian
  )
  class(fit) <- c("filament1_estimate", "list")
  fit
}

#' filament1_aux_EV
#'
#' @description Evaluate the expectation and variance for model A and B
#' @param beta A vector with the beta parameters
#' @param data A `data.frame` containing the required predictors, including `CAD_Weight`
#' @param model Either "A" for a log-linear variance model, or "B" for a proportional
#' scaling error model
#' @param Sigma_beta : If not NULL, an estimate of the covariance matrix for
#                 the uncertainty of estimated betas
#' @return A list with four elements:
#     E : E(y|beta,x)
#     V : Var(y|beta,x)
#     VE : Var(E(y|beta,x)|x) or NULL
#     EV : E(Var(y|beta,x)|x) or NULL

filament1_aux_EV <- function(beta, data, model = c("A", "B"),
                             Sigma_beta = NULL) {

  model <- match.arg(model)
  if (model == "A") {

    ZE.0 <- model.matrix( ~ 1 + CAD_Weight, data = data)
    ZV.0 <- model.matrix( ~ 1 + CAD_Weight, data = data)
    ZE = cbind(ZE.0, ZV.0 * 0)
    ZV = cbind(ZE.0 * 0, ZV.0)

    VE <- EV <- NULL
    if (!is.null(Sigma_beta)) {
      # E(Var(y|beta,x)|x)
      EV <- exp(ZV %*% beta + rowSums(ZV * (ZV %*% Sigma_beta)) / 2)
      # Var(E(y|beta,x)|x)
      VE <- rowSums(ZE * (ZE %*% Sigma_beta))
    }
    out <- list(
      E = ZE %*% beta,
      V = exp(ZV %*% beta),
      VE = VE,
      EV = EV
    )
  } else {

    ZE.0 <- model.matrix( ~ 1 + CAD_Weight, data = data)
    ZV.0 <- model.matrix( ~ 1 + I(CAD_Weight^2), data = data)
    ZE = cbind(ZE.0, ZV.0 * 0)
    ZV = cbind(ZE.0 * 0, ZV.0)

    VE <- EV <- NULL
    if (!is.null(Sigma_beta)) {
      # E(Var(y|beta,x)|x)
      # (pmin: Ignore large Sigma_beta values)
      EV <- ZV %*% exp(beta + pmin(0.5^2, diag(Sigma_beta)) / 2)
      # Var(E(y|beta,x)|x)
      VE <- rowSums(ZE * (ZE %*% Sigma_beta))
    }
    out <- list(
      E = ZE %*% beta,
      V = ZV %*% exp(beta),
      VE = VE,
      EV = EV
    )
  }
  out
}


#' Predict filament weights using specified model
#'
#' @description Compute predictive distributions and prediction intervals for a new dataset using eithe
#' @param data A `data.frame` used to estimate the model parameters.
#' @param model A character string specifying the model to use, either "A" or "B".
#' @param newdata A `data.frame` containing new observations for prediction.
#' @param alpha The significance level for prediction interval computation (default is 0.05).
#' @return A `data.frame` with the predicted mean, standard deviation, lower and upper bounds of the pr

filament1_predict <- function(data, model, newdata, alpha = 0.05){
  fit <- filament1_estimate(data, model)
  theta <- fit$par #beta
  #cov matrix = inverse hessian
  sigma_theta <- solve(fit$hessian) #sigma_beta

  #use filament1_aux_EV to get expectation and variance of model
  aux <- filament1_aux_EV(beta = theta, data = newdata,
                          model = model, Sigma_beta = sigma_theta)

  pred.df <- data.frame(
    mean = aux$E,
    sd = sqrt(aux$EV + aux$VE),
    lwr = NA,
    upr = NA
  )
  z_score <- qt(1-alpha/2, df = Inf)
  #upper and lower prediction interval
  pred.df$lwr <- pred.df$mean - z_score * pred.df$sd
  pred.df$upr <- pred.df$mean + z_score * pred.df$sd

  return (pred.df)
}

#' Calculate SE and DS Scores
#'
#' @description This function fits a specified model to the given data, makes predictions
#'              on new data, and then calculates the Squared Error (SE) and Dawid-Sebastiani (DS)
#'              scores for the predictions.
#' @param data A `data.frame` used to fit the model.
#' @param model A character string specifying the model to use, either "A" or "B".
#' @param newdata A `data.frame` with the actual weights and other predictors for calculating
#'                the scores.
#' @return A `data.frame` with predicted mean, standard deviation, lower and upper bounds of
#'         the prediction interval, as well as the calculated SE and DS scores for each observation.

fil_score <- function(data, model, newdata){
  #fit the model and get the prediction result
  fit <- filament1_predict(data = data, model = model, newdata = newdata)

  #calculate ES and DS score
  act_weight <- newdata$Actual_Weight #actual weight
  mean <- fit$mean
  sd <- fit$sd
  se <- (act_weight - mean)^2
  ds <- (act_weight - mean)^2 / sd^2 + 2*log(sd)
  fit$se <- se
  fit$ds <- ds

  return(fit)
}

#' Leave-One-Out Cross-Validation
#'
#' @description Perform leave-one-out cross-validation using the specified model.
#' @param data A `data.frame` containing the data to be used in cross-validation.
#' @param model A character string specifying the model to use, either "A" or "B".
#' @return A `data.frame` with added columns for mean, standard deviation, squared error, and Dawid-Seb

leave1out <- function(data, model){
  n <- nrow(data)
  mean <- numeric(n)
  sd <- numeric(n)

  for (i in 1:n){
    training_data <- data[-i, ]
    validation_data <- data[i, ]

    fit <- filament1_predict(data = training_data, model = model, newdata = validation_data)

    mean[i] <- fit$mean
    sd[i] <- fit$sd
  }

  res_df <- data.frame(
    mean = mean, sd = sd
  )

  score_res <- cbind(data, res_df) %>%
    mutate(
      se = (Actual_Weight - mean)^2,
      ds = ((Actual_Weight - mean)^2 / sd^2 + 2*log(sd))
    )
  return (score_res)
}


#part 2 codes
#' Calculate log-likelihood for Archaeological Model
#'
#' @description Computes the log-likelihood for a set of observations based on
#'              the binomial model for archaeological counts.
#' @param df A `data.frame` containing the parameters 'N' and 'phi' for which
#'           to calculate log-likelihoods.
#' @param y A vector containing the count of left and right femurs, respectively.
#' @return A modified `data.frame` that includes log-likelihoods for each set of parameters.

arch_loglike <- function(df, y) {
  log_likelihoods <- numeric(nrow(df))

  for (i in 1:nrow(df)) {
    N <- df$N[i]
    phi <- df$phi[i]
    #get y1 and y2 from the vector y
    y1 <- y[1]
    y2 <- y[2]

    #calculate the log-likelihood using lgamma for log(gamma)
    log_likelihoods[i] <- -lgamma(y1 + 1) - lgamma(y2 + 1) -
```

```r
      lgamma(N - y1 + 1) - lgamma(N - y2 + 1) + 2 * lgamma(N + 1) +
      (y1 + y2) * log(phi) + (2 * N - y1 - y2) * log(1 - phi)
  }

  df$log_likelihood <- log_likelihoods
  return(df)
}

#' Estimate Bayesian Posterior via Monte Carlo
#'
#' @description Estimates the posterior probabilities and expectations of 'N' and 'phi'
#'              using a Monte Carlo integration approach based on a binomial model
#'              for archaeological counts.
#' @param y A vector containing the count of left and right femurs, respectively.
#' @param xi The probability parameter for the geometric distribution used to sample 'N'.
#' @param a The 'a' parameter for the beta distribution used to sample 'phi'.
#' @param b The 'b' parameter for the beta distribution used to sample 'phi'.
#' @param K The number of Monte Carlo samples to generate.
#' @return A `data.frame` with estimated probabilities and expectations for 'N' and 'phi'.

estimate <- function(y, xi, a, b, K){
  #parameters for df
  N <- rgeom(K, xi)
  phi <- rbeta(K, a, b)
  #df for arch_loglike input containing N and phi
  df <- data.frame(N = N, phi = phi)
  #compute exponential of log likelihood
  exp_arch <- exp(arch_loglike(df, y = y)$log_likelihood)
  py <- (1/K)* sum(exp_arch)
  EN_y <- (1/(K*py)) * sum(N * exp_arch)
  EP_y <- (1/(K*py)) * sum(phi * exp_arch)

  df_res <- data.frame(py = py, EN_y = EN_y, EP_y = EP_y)
  return(df_res)
}
```