2.Gene_expresion

Fay

2022-05-27

Libraries:

```
library(tidyverse)
```

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

```
## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.7     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
```

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

```
library(tidyr)
library(dplyr)
library(cowplot)
library(randomForest)
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: ’randomForest’
```

```
## The following object is masked from ’package:dplyr’:
##
##     combine
```

```
## The following object is masked from ’package:ggplot2’:
##
##     margin
```

```
library(ggplot2)
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: ’caret’
```

```
## The following object is masked from ’package:purrr’:
##
##     lift
```

```
library(VIM) # visualizing missing data
```

```
## Loading required package: colorspace
```

```
## Loading required package: grid
```

```
## VIM is ready to use.
```

```
## Suggestions and bug-reports can be submitted at: https://github.com/statistikat/VIM/issues
```

```
##
## Attaching package: ’VIM’
```

```
## The following object is masked from ’package:datasets’:
##
##     sleep
```

```
library(mice) # imputing missing data without predictors
```

```
##
## Attaching package: ’mice’
```

```
## The following object is masked from ’package:stats’:
##
##     filter
```

```
## The following objects are masked from ’package:base’:
##
##     cbind, rbind
```

```
library(ggpubr)
```

```
##
## Attaching package: ’ggpubr’
```

```
## The following object is masked from ’package:cowplot’:
##
##     get_legend
```

```r
library(optimx)
```

Import the data:

```r
# Here we import the cleaned data set from the previous script derived from the
# data set challenge infections
g <- read.csv("output_data/gene_expression/data_products/clean_gene_expression.csv")


# vectors for selecting gene columns
Genes <- c("IFNy", "CXCR3_bio", "IL.6", "IL.10", "IL.13", "IL.10", "IL.13", "IL1RN",
           "CASP1", "CXCL9", "IDO1", "IRGM1", "MPO", "MUC2", "MUC5AC", "MYD88",
           "NCR1", "PRF1", "RETNLB", "SOCS1", "TICAM1", "TNF")
```

Data cleaning

```r
# we need to change the  in challenge infections to a factor
g$Parasite_challenge <- as.factor(g$Parasite_challenge)
g$Eim_MC <- as.factor(g$Eim_MC)

# Here I create a new column, where we get the actual infection status
# According to the melting curve for eimeria
g <- g %>%
  dplyr::mutate(current_infection = case_when(
    Parasite_challenge == "E_ferrisi" & Eim_MC == "TRUE" ~ "E_ferrisi",
    Parasite_challenge == "E_ferrisi" & Eim_MC == "FALSE" ~ "uninfected",
    Parasite_challenge == "E_falciformis" & Eim_MC == "TRUE" ~ "E_falciformis",
    Parasite_challenge == "E_falciformis" & Eim_MC == "FALSE" ~ "uninfected",
    Parasite_challenge == "uninfected" & Eim_MC == "TRUE" ~ "infected_eimeria",
    Parasite_challenge == "uninfected" & Eim_MC == "FALSE" ~ "uninfected",
    TRUE ~ ""
  ))


# how to impute delta? Replacing with 0 the ones with negative melting curve
# open for other solutions!
g <- g %>%
  dplyr::mutate(Intensity = case_when(
    Eim_MC == "TRUE" ~ delta,
    Eim_MC == "FALSE" ~ 0))

# create variable maximum weight loss instead of maximum relative weight loss
g <- g %>% dplyr::mutate(max_WL = max_WL - 100)
```

Imputing missing data + cleaning

```
#Start by selecting only the genes and the maximum weight loss for each mouse
# Apparently the relative end weight doesn't work so well for predictions

g.1 <- g %>%
  dplyr::select(c(max_WL, all_of(Genes)))

# to get reproducible results we use a seed
set.seed(42)

# We want the maximum weight loss to be predicted by the data ina ll of the other columns

# iter = how many random forests are needed, in theory 6 are enough
g.imputed <- rfImpute(max_WL ~ ., data = g.1, iter = 6)
```

```
##      |      Out-of-bag   |
## Tree |      MSE  %Var(y) |
##  300 |     26.1    61.11 |
##      |      Out-of-bag   |
## Tree |      MSE  %Var(y) |
##  300 |    27.08    63.39 |
##      |      Out-of-bag   |
## Tree |      MSE  %Var(y) |
##  300 |    27.97    65.49 |
##      |      Out-of-bag   |
## Tree |      MSE  %Var(y) |
##  300 |    28.27    66.19 |
##      |      Out-of-bag   |
## Tree |      MSE  %Var(y) |
##  300 |    28.26    66.17 |
##      |      Out-of-bag   |
## Tree |      MSE  %Var(y) |
##  300 |    28.24    66.11 |
```

```
g.imputed <- g.imputed %>% dplyr::select(-max_WL)

g_minus <- g %>%
  dplyr::select(-all_of(Genes))

#full data set containing the imputed gene expression data
g.imputed <- cbind(g_minus, g.imputed)
```

How many mice are in the infection planning?

```
g.imputed %>%
  filter(infection == "challenge") %>%
  group_by(Parasite_challenge) %>%
  summarize(length(EH_ID))
```

```
## # A tibble: 3 x 2
##   Parasite_challenge ‘length(EH_ID)‘
##   <fct>                        <int>
## 1 E_falciformis                      22
## 2 E_ferrisi                          47
## 3 uninfected                         47
```

How many mice are indeed infected?

```
g.imputed %>%
  filter(infection == "challenge") %>%
  group_by(current_infection) %>%
  summarize(length(EH_ID))
```

```
## # A tibble: 4 x 2
##   current_infection ‘length(EH_ID)‘
##   <chr>                       <int>
## 1 E_falciformis                  22
## 2 E_ferrisi                      39
## 3 infected_eimeria                9
## 4 uninfected                     46
```

I guess mice got mixed up here?

Splitting data into training and testing sets

Splitting between training and testing: - Assess model performance on unseen data - Avoid over-fitting

```
#select the relevant columns:
g.imputed <- g.imputed %>%
  dplyr::select(c(max_WL, all_of(Genes))) %>%
  dplyr::select(- CXCR3_bio)

# split data into training and test

set.seed(123) # this will help us reproduce this random assignment

# in this way we can pick the random numbers

training.samples <- g.imputed$max_WL%>%
  createDataPartition(p = .7, # this is the partiicition! In this case 0.7 = training data and 0.3 = te
                      list = FALSE) # we don't want to get a list in return

train.data <- g.imputed[training.samples, ] #include all the randomly selected rows
test.data <- g.imputed[-training.samples, ]
```

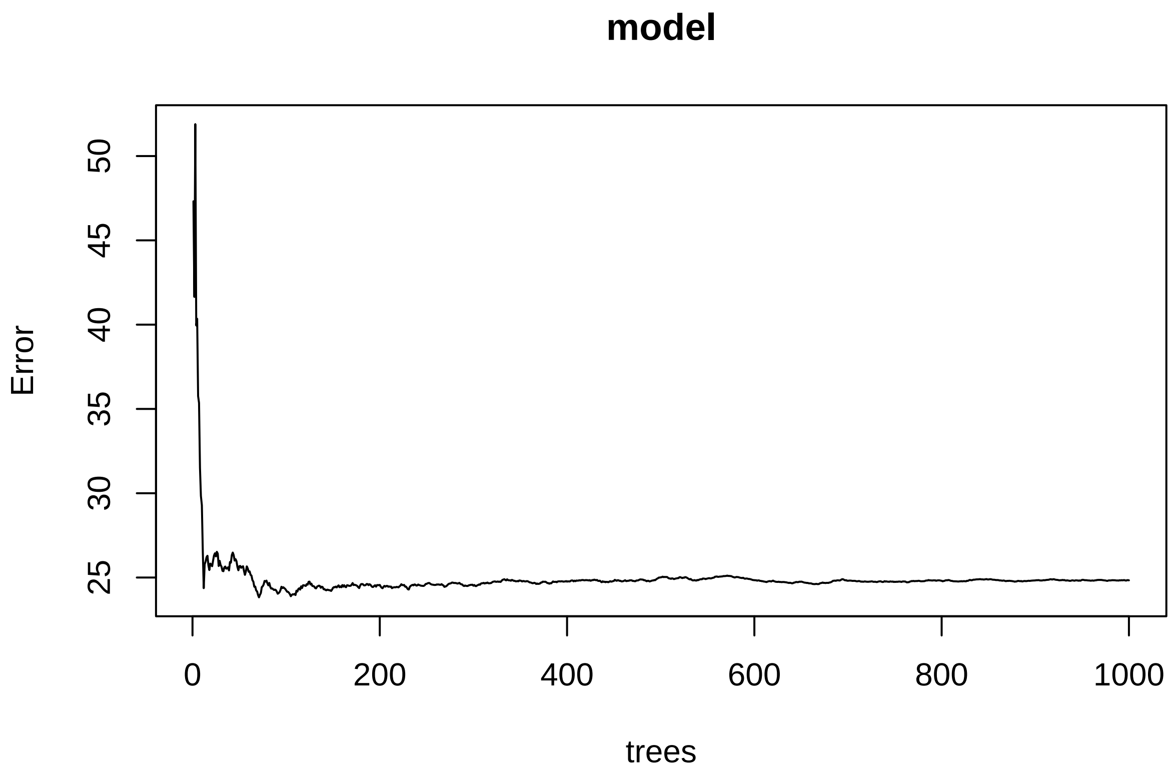
Building the model

```
#train the model
model <- randomForest(max_WL ~., data = train.data, proximity = TRUE,
                      ntree = 1000) # number of trees



print(model)
```

```
##
## Call:
##  randomForest(formula = max_WL ~ ., data = train.data, proximity = TRUE,      ntree = 1000)
##                Type of random forest: regression
##                      Number of trees: 1000
## No. of variables tried at each split: 6
##
##           Mean of squared residuals: 24.8344
##                     % Var explained: 40.55
```

Plotting the model will illustrate the error rate as we average across more trees and shows that our error rate stabalizes with around 200 trees.

```
plot(model)
```



The plotted error rate above is based on the OOB sample error and can be accessed directly at m1$mse. Thus, we can find which number of trees providing the lowest error rate, which is 257 trees providing an weight error of 5.024738.

```
# number of trees with lowest MSE
which.min(model$mse)
```

```
## [1] 71
## [1] 257

# RMSE of this optimal random forest
sqrt(model$mse[which.min(model$mse)])
```

```
## [1] 4.88103
```

```
## [1] 5.024738
```

https://uc-r.github.io/s

RandomForest also allows us to use a validation set to measure predictive accuracy if we did not want to use the OOB samples.

Tutorial: https://hackernoon.com/random-forest-regression-in-r-code-and-interpretation Random forest regression in R provides two outputs: decrease in mean square error (MSE) and node purity. Prediction error described as MSE is based on permuting out-of-bag sections of the data per individual tree and predictor, and the errors are then averaged. In the regression context, Node purity is the total decrease in residual sum of squares when splitting on a variable averaged over all trees (i.e. how well a predictor decreases variance). MSE is a more reliable measure of variable importance. If the two importance metrics show different results, listen to MSE. If all of your predictors are numerical, then it shouldn't be too much of an issue

Mean Decrease Gini (IncNodePurity) - This is a measure of variable importance based on the Gini impurity index used for the calculating the splits in trees.

Improving Your Model Your model depends on the quality of your dataset and the type of Machine Learning algorithm used. Therefore, to improve the accuracy of your model, you should:

Check what attributes affect our model the most and what variables to leave out in future analysis Find out what other attributes affect a person's wage; we can use as predictors in future analysis Tweak the algorithm (e.g. change the ntree value) Use a different machine learning algorithm If any of these reduces the RMSE significantly, you have succeeded in improving your model!

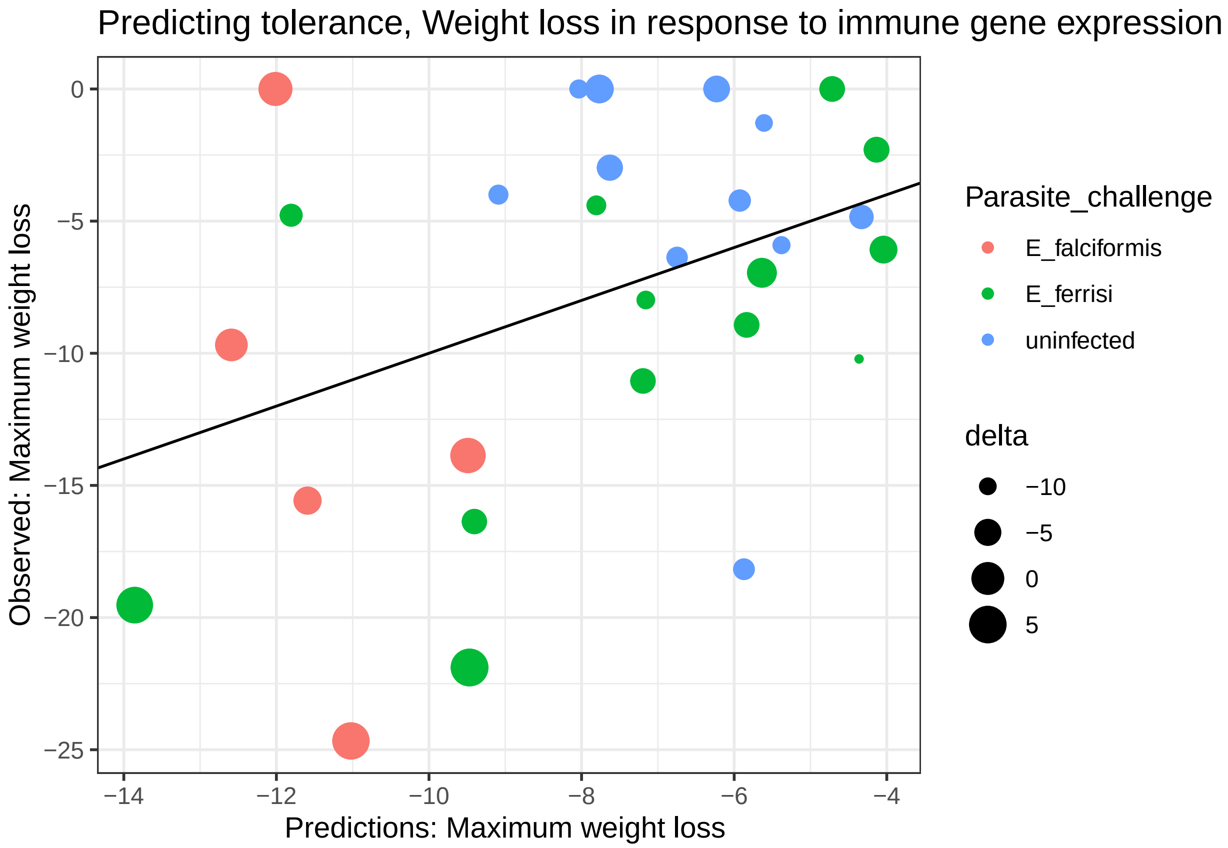
Making predictions

```
#The predict() function in R is used to predict the values based on the input data.
predictions <- predict(model, test.data)

# assign test.data to a new object, so that we can make changes
result <- test.data

#add the new variable of predictions to the result object
result <- cbind(result, predictions)

#add the results to a data frame containing test data and the prediction
result <- cbind(g[row.names(result), ], predictions)
```

Visualizations

```
result   %>%
  ggplot() +
  geom_point(aes(x = predictions, y = max_WL, color = Parasite_challenge, size = delta)) +
  geom_abline() +
  labs(x = "Predictions: Maximum weight loss ", y = "Observed: Maximum weight loss",
         title = "Predicting tolerance, Weight loss in response to immune gene expression") +
    theme_bw()
```

```
## Warning: Removed 3 rows containing missing values (geom_point).
```



Predicting eimeria species according to gene expression

```
g$current_infection <- as.factor(g$current_infection)

#now select the genes and the actual infection of the mice in the new mutate column
#infection
g.2 <- g %>%
    dplyr::select(c(current_infection, delta, all_of(Genes)))

# to get reproducible results we use a seed
set.seed(42)
# We want the current infection to be predicted by the data ina ll of the other columns
# iter = how many random forests are needed, in theory 6 are enough


#now we can impute our data
g.imputed_parasite <- rfImpute(current_infection ~ ., data = g.2, iter = 6)
```

```
## ntree      OOB      1      2      3      4
##   300:  34.48% 40.91% 33.33%100.00% 19.57%
## ntree      OOB      1      2      3      4
##   300:  35.34% 45.45% 33.33%100.00% 19.57%
## ntree      OOB      1      2      3      4
##   300:  40.52% 50.00% 35.90%100.00% 28.26%
## ntree      OOB      1      2      3      4
##   300:  36.21% 40.91% 33.33%100.00% 23.91%
## ntree      OOB      1      2      3      4
##   300:  36.21% 36.36% 33.33%100.00% 26.09%
## ntree      OOB      1      2      3      4
##   300:  36.21% 40.91% 35.90%100.00% 21.74%
```

```
g.imputed_parasite <- g.imputed_parasite %>% dplyr::select(- current_infection)

g_minus <- g %>% dplyr::select(-c((all_of(Genes)), delta))

#full data set containing the imputed gene expression data
g.imputed_parasite <- cbind(g_minus, g.imputed_parasite)
```

Now split the data again into training and testing

```
g.imputed_parasite$current_infection <- as.factor(g.imputed_parasite$current_infection)

#select the relevant columns:
g.imputed_parasite <- g.imputed_parasite %>%
  dplyr::select(c(current_infection, delta, all_of(Genes)))

# split data into training and test
set.seed(123) # this will help us reproduce this random assignment
# in this way we can pick the random numbers
training.samples_parasite <- g.imputed_parasite$current_infection%>%
  createDataPartition(p = .7, # this is the partiicition! In this case 0.7 = training data and 0.3 = te
                      list = FALSE) # we don't want to get a list in return
train.data_parasite <- g.imputed_parasite[training.samples, ] #include all the randomly selected rows
test.data_parasite <- g.imputed_parasite[-training.samples, ]
```

Building the model

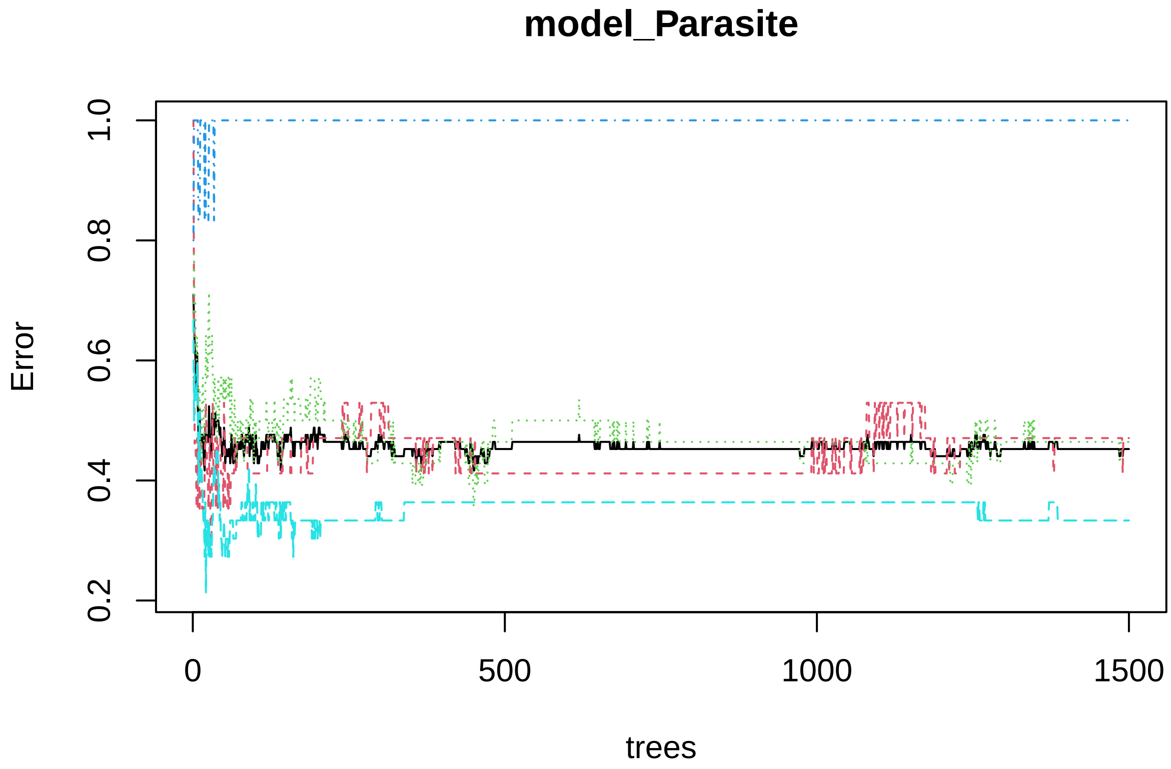
```
#train the model
model_Parasite <- randomForest(current_infection ~., data = train.data_parasite, proximity = TRUE,
                      ntree = 1500) # number of trees
```

```
print(model_Parasite)
```

```
##
## Call:
##  randomForest(formula = current_infection ~ ., data = train.data_parasite,      proximity = TRUE, nt
##                Type of random forest: classification
##                      Number of trees: 1500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 45.24%
## Confusion matrix:
##                   E_falciformis E_ferrisi infected_eimeria uninfected
## E_falciformis                 9         6                0          2
## E_ferrisi                     6        15                0          7
## infected_eimeria              0         1                0          5
## uninfected                    1        10                0         22
##                   class.error
## E_falciformis       0.4705882
## E_ferrisi           0.4642857
## infected_eimeria    1.0000000
## uninfected          0.3333333
```

OOB = 46.43, this means that only 53 % of our predictions are accurate

```
plot(model_Parasite)
```



Test the model

Making predictions

```
#The predict() function in R is used to predict the values based on the input data.
predictions_parasite <- predict(model_Parasite, test.data_parasite)
# assign test.data to a new object, so that we can make changes
result_parasite <- test.data_parasite
#add the new variable of predictions to the result object
result_parasite <- cbind(result_parasite, predictions_parasite)
#add the results to a data frame containing test data and the prediction
result_parasite <- cbind(g[row.names(result_parasite), ], predictions_parasite)
```

same but for parasite challenge instead of current infection

Predicting eimeria species according to gene expression

```
g$Parasite_challenge <- as.factor(g$Parasite_challenge)

#now select the genes and the actual infection of the mice in the new mutate column
#infection
g.2 <- g %>%
    dplyr::select(c(Parasite_challenge, delta, all_of(Genes)))

# to get reproducible results we use a seed
set.seed(42)
# We want the current infection to be predicted by the data ina ll of the other columns
# iter = how many random forests are needed, in theory 6 are enough


#now we can impute our data
g.imputed_parasite <- rfImpute(Parasite_challenge ~ ., data = g.2, iter = 6)
```

```
## ntree      OOB      1      2      3
##   300:  31.03% 54.55% 31.91% 19.15%
## ntree      OOB      1      2      3
##   300:  30.17% 50.00% 29.79% 21.28%
## ntree      OOB      1      2      3
##   300:  27.59% 45.45% 29.79% 17.02%
## ntree      OOB      1      2      3
##   300:  27.59% 50.00% 27.66% 17.02%
## ntree      OOB      1      2      3
##   300:  29.31% 45.45% 31.91% 19.15%
## ntree      OOB      1      2      3
##   300:  27.59% 50.00% 27.66% 17.02%
```

```
g.imputed_parasite <- g.imputed_parasite %>% dplyr::select(- Parasite_challenge)

g_minus <- g %>% dplyr::select(-c((all_of(Genes)), delta))
```

```
#full data set containing the imputed gene expression data
g.imputed_parasite <- cbind(g_minus, g.imputed_parasite)
```

Now split the data again into training and testing

```
g.imputed_parasite$Parasite_challenge <- as.factor(g.imputed_parasite$Parasite_challenge)

#select the relevant columns:
g.imputed_parasite <- g.imputed_parasite %>%
  dplyr::select(c(Parasite_challenge, Eim_MC, all_of(Genes)))

# to use in the next model
parasite_data <- g.imputed_parasite

g.imputed_parasite <- g.imputed_parasite %>%
  dplyr::select(-Eim_MC)

# split data into training and test
set.seed(123) # this will help us reproduce this random assignment
# in this way we can pick the random numbers
training.samples_parasite <- g.imputed_parasite$Parasite_challenge%>%
  createDataPartition(p = .7, # this is the partiicition! In this case 0.7 = training data and 0.3 = te
                      list = FALSE) # we don't want to get a list in return
train.data_parasite <- g.imputed_parasite[training.samples, ] #include all the randomly selected rows
test.data_parasite <- g.imputed_parasite[-training.samples, ]
```

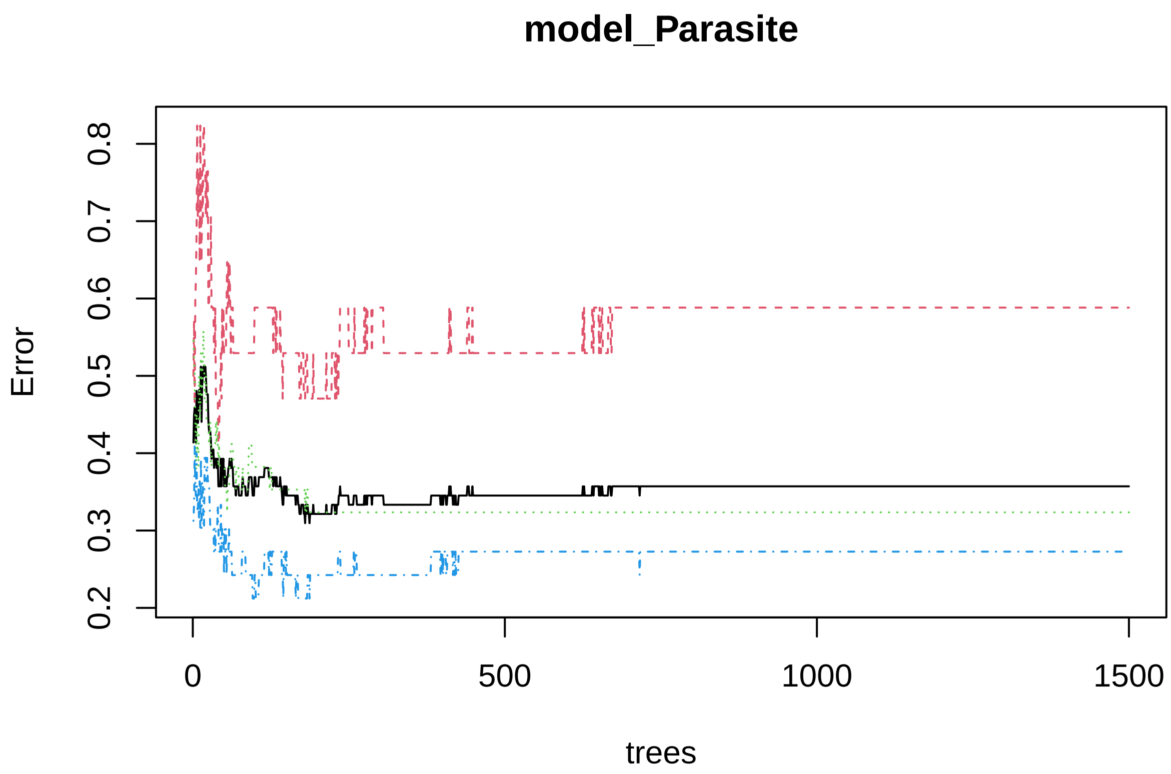
Building the model

```
#train the model
model_Parasite <- randomForest(Parasite_challenge ~., data = train.data_parasite, proximity = TRUE,
                      ntree = 1500) # number of trees

print(model_Parasite)
```

```
##
## Call:
##  randomForest(formula = Parasite_challenge ~ ., data = train.data_parasite,      proximity = TRUE, n
##                Type of random forest: classification
##                      Number of trees: 1500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 35.71%
## Confusion matrix:
##               E_falciformis E_ferrisi uninfected class.error
## E_falciformis             7         7          3   0.5882353
## E_ferrisi                 4        23          7   0.3235294
## uninfected                1         8         24   0.2727273
```

OOB = 46.43, this means that only 53 % of our predictions are accurate

```
plot(model_Parasite)
```



Test the model

Making predictions

```
#The predict() function in R is used to predict the values based on the input data.
predictions_parasite <- predict(model_Parasite, test.data_parasite)
# assign test.data to a new object, so that we can make changes
result_parasite <- test.data_parasite
#add the new variable of predictions to the result object
result_parasite <- cbind(result_parasite, predictions_parasite)
#add the results to a data frame containing test data and the prediction
result_parasite <- cbind(g[row.names(result_parasite), ], predictions_parasite)
```

Visualizations

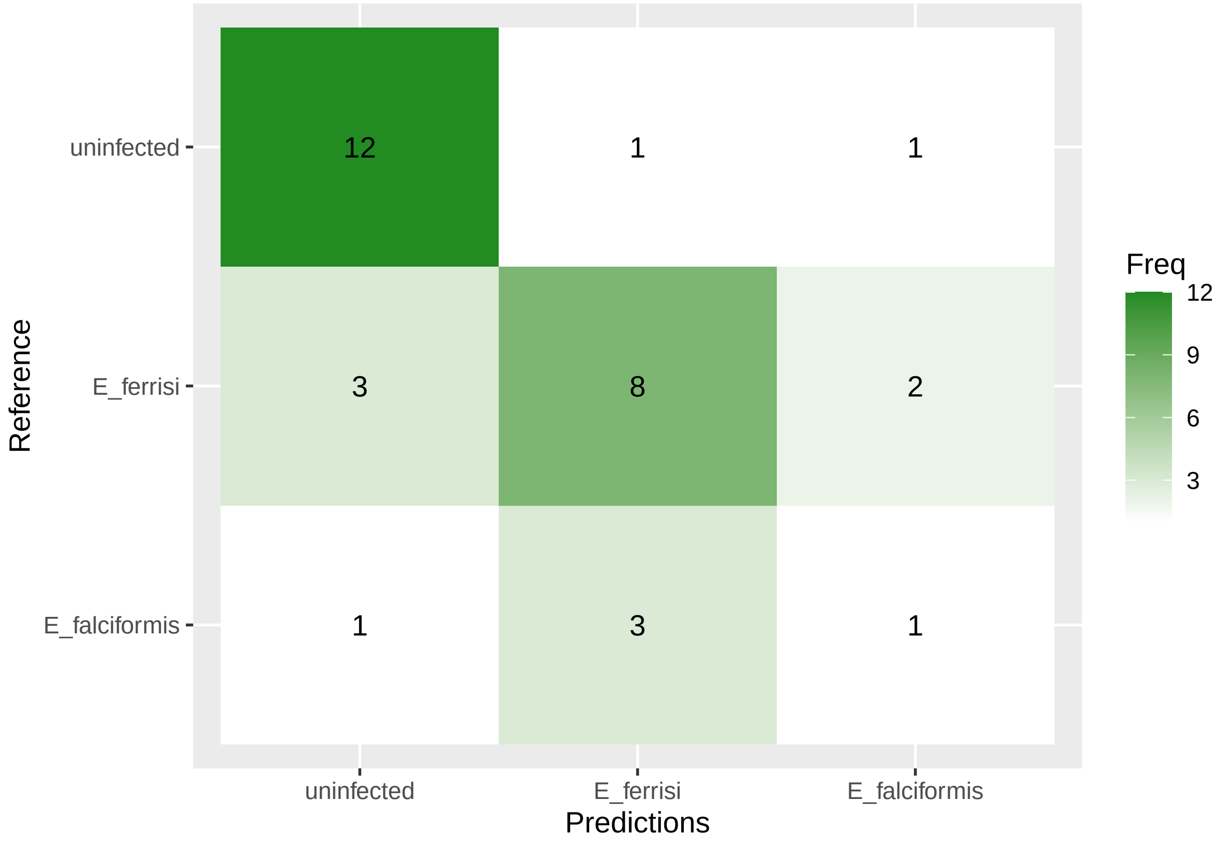
```
conf_matrix_parasite <- confusionMatrix(result_parasite$predictions_parasite, reference = result_parasi

print(conf_matrix_parasite)
```

```
## Confusion Matrix and Statistics
##
##                 Reference
## Prediction      E_falciformis E_ferrisi uninfected
##   E_falciformis             1         2          1
##   E_ferrisi                 3         8          1
##   uninfected                1         3         12
##
## Overall Statistics
##
##                 Accuracy : 0.6562
##                   95% CI : (0.4681, 0.8143)
##     No Information Rate : 0.4375
##     P-Value [Acc > NIR] : 0.01044
##
##                    Kappa : 0.4359
##
##  Mcnemar's Test P-Value : 0.75300
##
## Statistics by Class:
##
##                      Class: E_falciformis Class: E_ferrisi Class: uninfected
## Sensitivity                       0.20000           0.6154            0.8571
## Specificity                       0.88889           0.7895            0.7778
## Pos Pred Value                    0.25000           0.6667            0.7500
## Neg Pred Value                    0.85714           0.7500            0.8750
## Prevalence                        0.15625           0.4062            0.4375
## Detection Rate                    0.03125           0.2500            0.3750
## Detection Prevalence              0.12500           0.3750            0.5000
## Balanced Accuracy                 0.54444           0.7024            0.8175
```

```
conf_matrix_parasite$table
```

```
##                 Reference
## Prediction      E_falciformis E_ferrisi uninfected
##   E_falciformis             1         2          1
##   E_ferrisi                 3         8          1
##   uninfected                1         3         12
```

```
plt <- as.data.frame(conf_matrix_parasite$table)
plt$Prediction <- factor(plt$Prediction, levels=rev(levels(plt$Prediction)))

ggplot(plt, aes(x = Prediction, y =  Reference, fill= Freq)) +
        geom_tile() + geom_text(aes(label=Freq)) +
        scale_fill_gradient(low="white", high="forestgreen") +
        labs(x = "Predictions",y = "Reference")
```



```
train.data_parasite %>%
  group_by(Parasite_challenge) %>%
  summarize(length(Parasite_challenge))
```

```
## # A tibble: 3 x 2
##   Parasite_challenge ‘length(Parasite_challenge)‘
##   <fct>                                    <int>
## 1 E_falciformis                               17
## 2 E_ferrisi                                   34
## 3 uninfected                                  33
```

Repeat the previous model, this time testing for

infected with Eimeria or not. ###

```
# to use in the next model
parasite_data <- parasite_data %>%
  dplyr::select(-Parasite_challenge)


# split data into training and test
set.seed(123) # this will help us reproduce this random assignment
# in this way we can pick the random numbers
training.samples_melting <- parasite_data$Eim_MC%>%
  createDataPartition(p = .7, # this is the partiicition! In this case 0.7 = training data and 0.3 = te
                      list = FALSE) # we don't want to get a list in return
train.data_melting <- parasite_data[training.samples, ] #include all the randomly selected rows
test.data_melting <- parasite_data[-training.samples, ]
```

Building the model

```
#train the model
model_melting <- randomForest(Eim_MC ~., data = train.data_melting, proximity = TRUE,
                      ntree = 1500) # number of trees

print(model_melting)
```

```
##
## Call:
##  randomForest(formula = Eim_MC ~ ., data = train.data_melting,      proximity = TRUE, ntree = 1500)
##                Type of random forest: classification
##                      Number of trees: 1500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 30.95%
## Confusion matrix:
##       FALSE TRUE class.error
## FALSE    16   17   0.5151515
## TRUE      9   42   0.1764706
```

Test the model

Making predictions

```
#The predict() function in R is used to predict the values based on the input data.
predictions_melting <- predict(model_melting, test.data_melting)
# assign test.data to a new object, so that we can make changes
result_melting <- test.data_melting
#add the new variable of predictions to the result object
result_melting <- cbind(result_melting, predictions_melting)
#add the results to a data frame containing test data and the prediction
result_melting <- cbind(g[row.names(result_melting), ], predictions_melting)
```

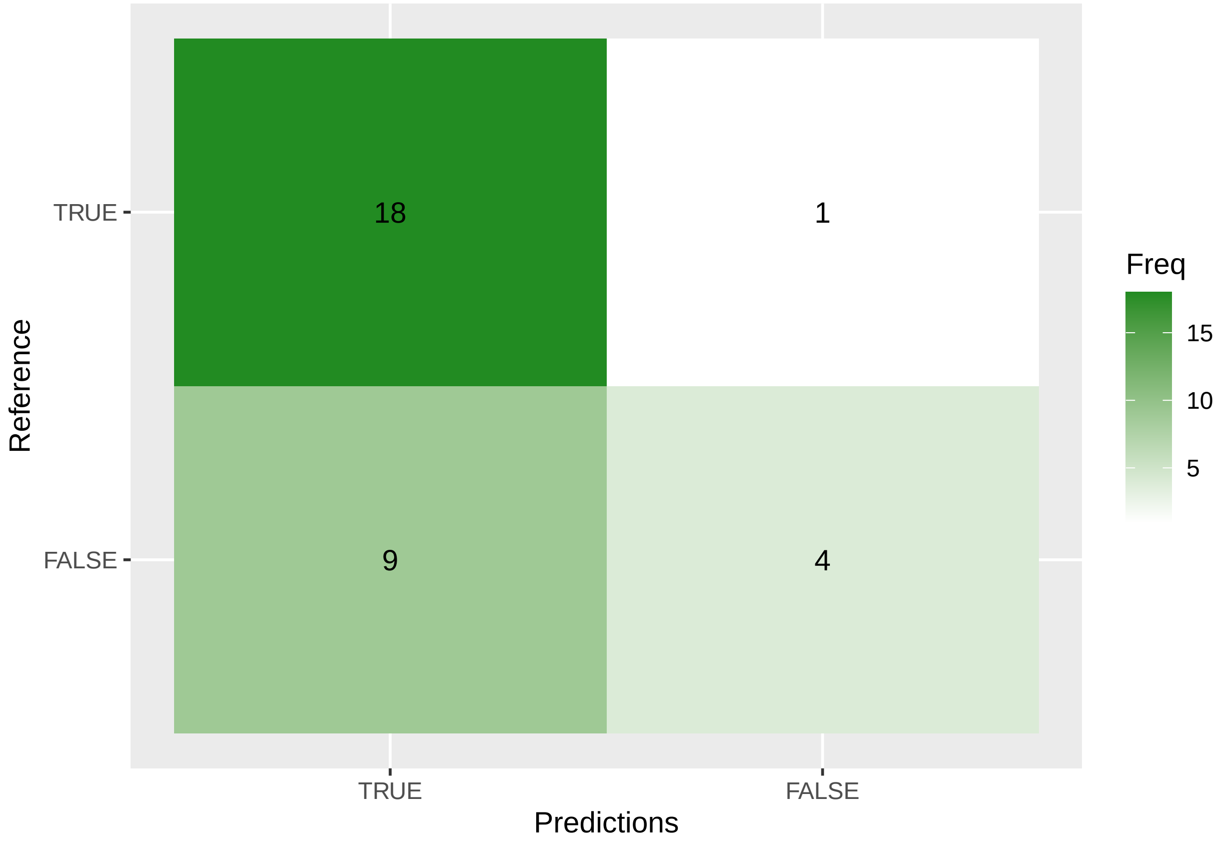
```
conf_matrix_melting <- confusionMatrix(result_melting$predictions_melting, reference = result_melting$E

print(conf_matrix_melting)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE     4    1
##      TRUE      9   18
##
##                Accuracy : 0.6875
##                  95% CI : (0.4999, 0.8388)
##     No Information Rate : 0.5938
##     P-Value [Acc > NIR] : 0.18482
##
##                   Kappa : 0.2825
##
##  Mcnemar's Test P-Value : 0.02686
##
##             Sensitivity : 0.3077
##             Specificity : 0.9474
##          Pos Pred Value : 0.8000
##          Neg Pred Value : 0.6667
##              Prevalence : 0.4062
##          Detection Rate : 0.1250
##    Detection Prevalence : 0.1562
##       Balanced Accuracy : 0.6275
##
##        'Positive' Class : FALSE
##
```

```
conf_matrix_melting$table
```

```
##           Reference
## Prediction FALSE TRUE
##      FALSE     4    1
##      TRUE      9   18
```

```
plt <- as.data.frame(conf_matrix_melting$table)
plt$Prediction <- factor(plt$Prediction, levels=rev(levels(plt$Prediction)))

ggplot(plt, aes(x = Prediction, y =  Reference, fill= Freq)) +
        geom_tile() + geom_text(aes(label=Freq)) +
        scale_fill_gradient(low="white", high="forestgreen") +
        labs(x = "Predictions",y = "Reference")
```



Field data

Importing field data

```
Field <- read.csv("https://raw.githubusercontent.com/derele/Mouse_Eimeria_Field/master/data_products/SO
```

Summary statistics for the field data

```
Field %>% summarise(length(Mouse_ID))
```

```
##   length(Mouse_ID)
## 1             1921
```

We have 1921 mice in total.

```
EqPCR.cols      <- c("delta_ct_cewe_MminusE", "MC.Eimeria", "Ct.Eimeria") #,"Ct.Mus""delta_ct_ilwe_Mmin

EimGeno.cols    <- c("n18S_Seq", "COI_Seq", "ORF470_Seq", "eimeriaSpecies")

Gene.Exp.cols   <- c("IFNy",  "CXCR3", "IL.6", #"GBP2", "IL.12", "IRG6",
                     "IL.10", "IL.13", "IL.10", "IL.13", "IL1RN",
                     "CXCR3", "CASP1", "CXCL9",
                     "IDO1", "IRGM1", "MPO", "MUC2", "MUC5AC", "MYD88",
                     "NCR1", "PRF1", "RETNLB", "SOCS1", "TICAM1", "TNF")

House.Keeping.cols <- c("GAPDH", "PPIB", "B.actin", "B-actin")
```

```
#which are the numbers of the columns of Field
names <- data.frame(colnames(Field))

f <- Field[ , c(76:78, 80:97)]

#how many nas in each column
sapply(f, function(x) sum(is.na(x)))
```

```
##  CASP1  CXCL9  CXCR3   IDO1   IFNy  IL.10  IL.13  IL1RN  IRGM1    MPO   MUC2
##   1713   1627   1695   1614   1592   1802   1613   1616   1596   1624   1599
## MUC5AC  MYD88   NCR1   PPIB   PRF1 RETNLB  SOCS1 TICAM1    TNF   IL.6
##   1615   1605   1711   1797   1720   1693   1596   1703   1625   1684
```

```
#remove rows with only nas
f <- f[rowSums(is.na(f)) != ncol(f), ]

Field <- Field %>%
  dplyr::select(-c(76:78, 80:97))

#merge the data frame to keep only the selected rows
f <- merge(Field, f, by = "row.names")
```

Imputing missing data

For the lab data I have used the function rfimpute from the package random forest. I can't use the same function for our lab data as the function requires the data set to contain predictor variable and response variables.

Therefore I will be using the package MICE (multivariate Imputation by chained Equations) which only requires a data frame of missing observations.

Description: *Multiple imputation using Fully Conditional Specification (FCS)*

implemented by the MICE algorithm as described in Van Buuren and Groothuis-Oudshoorn (2011) doi:10.18637/jss.v045.i03. Each variable has its own imputation model. Built-in imputation models are provided for continuous data (predictive mean matching, normal), binary data (logistic regression), unordered categorical data (polytomous logistic regression) and ordered categorical data (proportional odds). MICE can also impute continuous two-level data (normal model, pan, second-level variables). Passive imputation can be used to maintain consistency between variables. Various diagnostic plots are available to inspect the quality of the imputations.

https://www.jstatsoft.org/article/view/v045i03

tutorial: https://www.youtube.com/watch?v=WPiYOS3qK70

https://datascienceplus.com/imputing-missing-data-with-r-mice-package/

https://datascienceplus.com/handling-missing-data-with-mice-package-a-simple-approach/

Missing data can be classified into three categories:

1. Missing completely at random (MCAR)

We can't probably predict that value from any other value in the data. MCAR implies the reason for the missingness of a field is completely random, and that we probably can't predict that value from any other value in the data.

2. Missing at Random (MAR)

Missingess can be explained by other values in other columns, but not from that column.

3. Missing NOT at random (MNAR)

The basic MICE assumption is that the data is missing at random, and that we can make a guess about its true value by looking at other data samples.

Let's start by cleaning and checking the missing data points in our field data.

```
library(mice)

f <- f %>% dplyr::select(-"Row.names")


#turn the eimeria species into logical
f$eimeriaSpecies <- as.factor(f$eimeriaSpecies)

field_genes <- f %>%
  dplyr::select(Gene.Exp.cols)
```

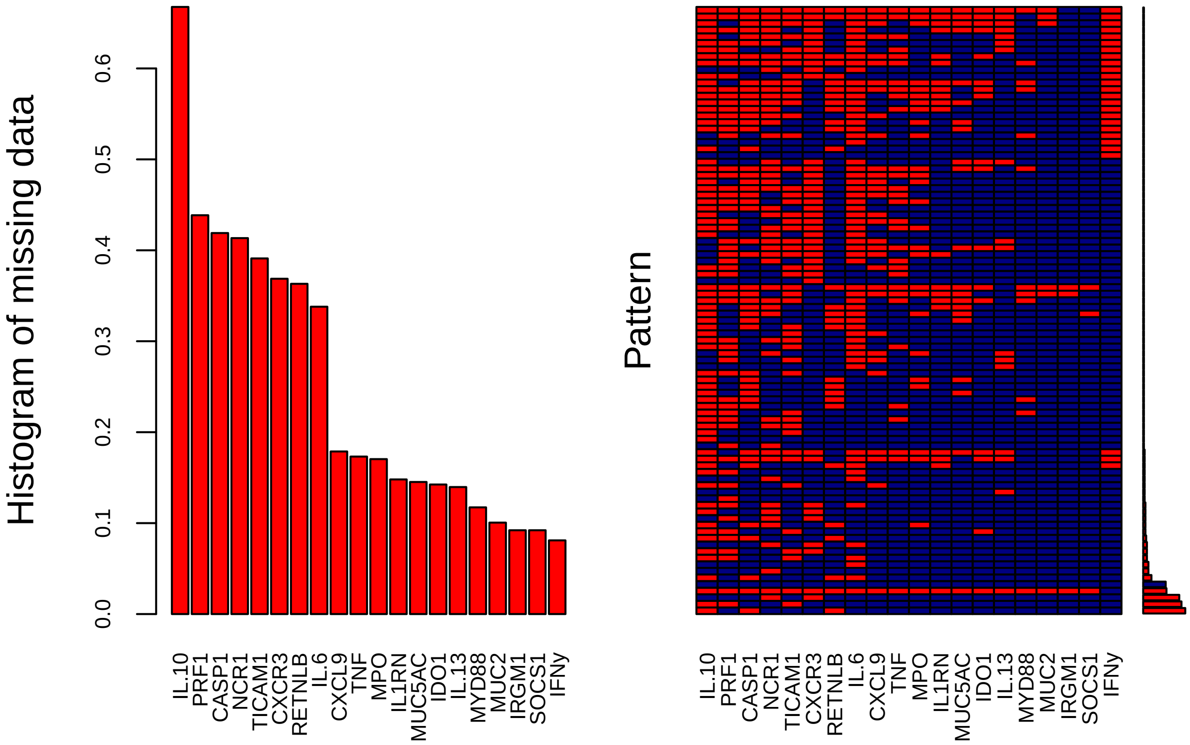
```
## Note: Using an external vector in selections is ambiguous.
## i Use ‘all_of(Gene.Exp.cols)‘ instead of ‘Gene.Exp.cols‘ to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.
```

```
# check the data for missing values
sapply(field_genes, function(x) sum(is.na(x)))
```

```
##   IFNy  CXCR3   IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9    IDO1  IRGM1    MPO
##     29    132    121    239     50     53    150     64      51     33     61
##   MUC2 MUC5AC  MYD88   NCR1   PRF1 RETNLB  SOCS1 TICAM1     TNF
##     36     52     42    148    157    130     33    140      62
```

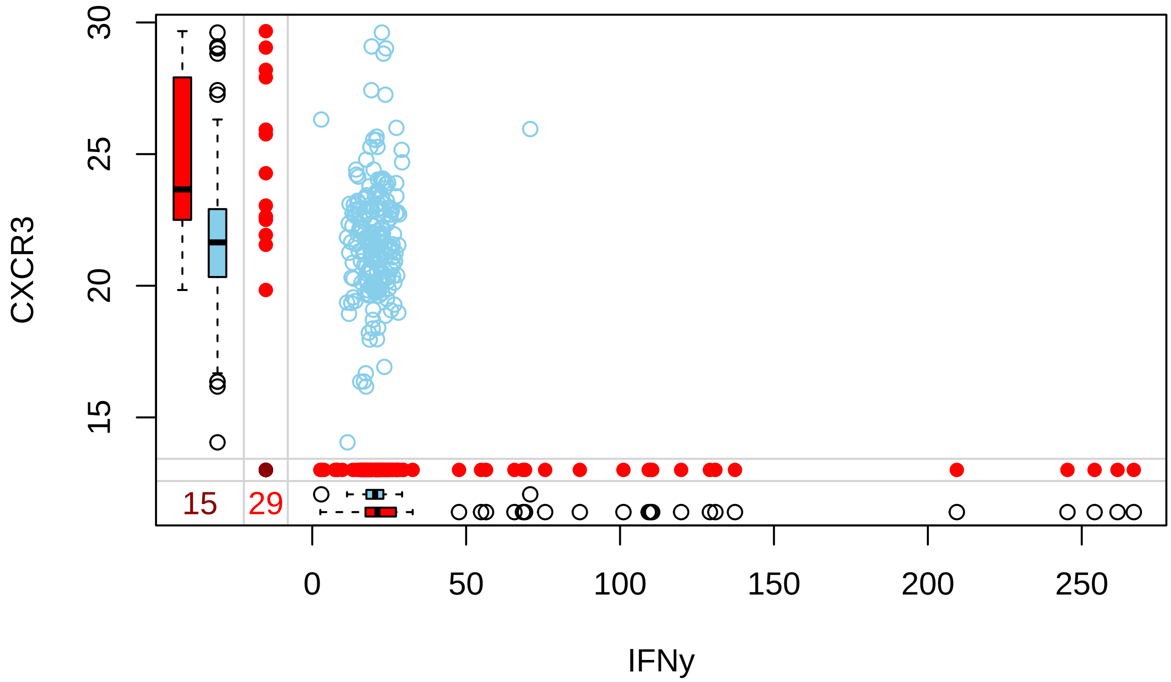
```
field_genes %>%
  aggr(col = c('navyblue', 'red'), numbers = TRUE, sortVars = TRUE, labels=names(field_genes), cex.axis=
```

```
## Warning in plot.aggr(res, ...): not enough vertical space to display frequencies
## (too many combinations)
```



```
##
##  Variables sorted by number of missings:
##  Variable      Count
##     IL.10 0.66759777
##      PRF1 0.43854749
##     CASP1 0.41899441
##      NCR1 0.41340782
##    TICAM1 0.39106145
##     CXCR3 0.36871508
##    RETNLB 0.36312849
##      IL.6 0.33798883
##     CXCL9 0.17877095
##       TNF 0.17318436
##       MPO 0.17039106
##     IL1RN 0.14804469
##    MUC5AC 0.14525140
##      IDO1 0.14245810
##     IL.13 0.13966480
##     MYD88 0.11731844
##      MUC2 0.10055866
##     IRGM1 0.09217877
##     SOCS1 0.09217877
##      IFNy 0.08100559
```

```
marginplot(field_genes[c(1,2)])
```



Now let's coninue by ussing the package MICE to impute the data

```
# The frequency distribution of the missing cases per variable can be obtained as:
init <- mice(field_genes, maxit = 0)

# table of amount of variables with the amount of missing values
table(init$nmis)
```

```
##
##  29  33  36  42  50  51  52  53  61  62  64 121 130 132 140 148 150 157 239
##   1   2   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
```

```
# which method is used for imputation? In this case the package mice
# uses the default method for continuous variable,
# which is pmm, or predictive mean matching

meth <- init$method


# now impute the immune gene expression for the field and save it as the oject:
```

```
# igf
# m=5 refers to the number of imputed datasets. Five is the default value.
igf <- mice(field_genes, method = meth, m = 5, seed = 500)
```

```
## 
##  iter imp variable
##   1   1  IFNy  CXCR3  IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9  IDO1  IRGM1  MPO  MUC2  MUC5AC  MYD88
##   1   2  IFNy  CXCR3  IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9  IDO1  IRGM1  MPO  MUC2  MUC5AC  MYD88
##   1   3  IFNy  CXCR3  IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9  IDO1  IRGM1  MPO  MUC2  MUC5AC  MYD88
##   1   4  IFNy  CXCR3  IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9  IDO1  IRGM1  MPO  MUC2  MUC5AC  MYD88
##   1   5  IFNy  CXCR3  IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9  IDO1  IRGM1  MPO  MUC2  MUC5AC  MYD88
##   2   1  IFNy  CXCR3  IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9  IDO1  IRGM1  MPO  MUC2  MUC5AC  MYD88
##   2   2  IFNy  CXCR3  IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9  IDO1  IRGM1  MPO  MUC2  MUC5AC  MYD88
##   2   3  IFNy  CXCR3  IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9  IDO1  IRGM1  MPO  MUC2  MUC5AC  MYD88
##   2   4  IFNy  CXCR3  IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9  IDO1  IRGM1  MPO  MUC2  MUC5AC  MYD88
##   2   5  IFNy  CXCR3  IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9  IDO1  IRGM1  MPO  MUC2  MUC5AC  MYD88
##   3   1  IFNy  CXCR3  IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9  IDO1  IRGM1  MPO  MUC2  MUC5AC  MYD88
##   3   2  IFNy  CXCR3  IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9  IDO1  IRGM1  MPO  MUC2  MUC5AC  MYD88
##   3   3  IFNy  CXCR3  IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9  IDO1  IRGM1  MPO  MUC2  MUC5AC  MYD88
##   3   4  IFNy  CXCR3  IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9  IDO1  IRGM1  MPO  MUC2  MUC5AC  MYD88
##   3   5  IFNy  CXCR3  IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9  IDO1  IRGM1  MPO  MUC2  MUC5AC  MYD88
##   4   1  IFNy  CXCR3  IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9  IDO1  IRGM1  MPO  MUC2  MUC5AC  MYD88
##   4   2  IFNy  CXCR3  IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9  IDO1  IRGM1  MPO  MUC2  MUC5AC  MYD88
##   4   3  IFNy  CXCR3  IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9  IDO1  IRGM1  MPO  MUC2  MUC5AC  MYD88
##   4   4  IFNy  CXCR3  IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9  IDO1  IRGM1  MPO  MUC2  MUC5AC  MYD88
##   4   5  IFNy  CXCR3  IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9  IDO1  IRGM1  MPO  MUC2  MUC5AC  MYD88
##   5   1  IFNy  CXCR3  IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9  IDO1  IRGM1  MPO  MUC2  MUC5AC  MYD88
##   5   2  IFNy  CXCR3  IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9  IDO1  IRGM1  MPO  MUC2  MUC5AC  MYD88
##   5   3  IFNy  CXCR3  IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9  IDO1  IRGM1  MPO  MUC2  MUC5AC  MYD88
##   5   4  IFNy  CXCR3  IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9  IDO1  IRGM1  MPO  MUC2  MUC5AC  MYD88
##   5   5  IFNy  CXCR3  IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9  IDO1  IRGM1  MPO  MUC2  MUC5AC  MYD88
```

```
summary(igf)
```

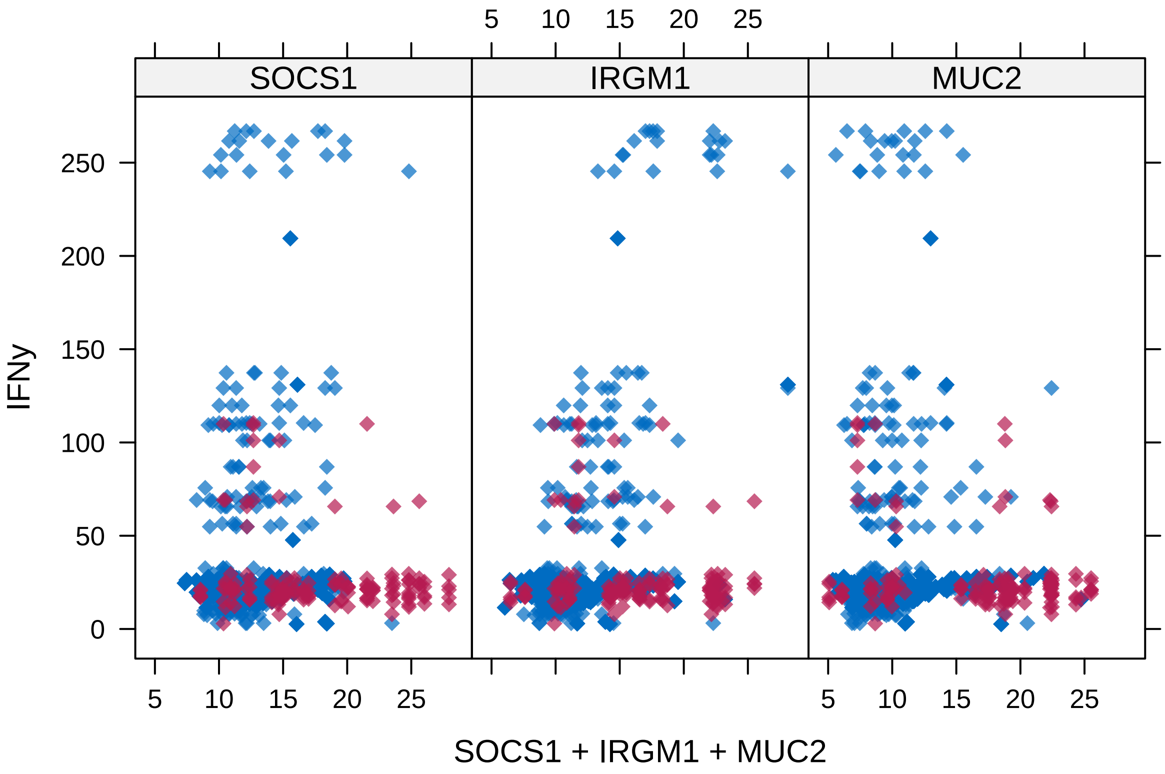
```
## Class: mids
## Number of multiple imputations:  5 
## Imputation methods:
##   IFNy  CXCR3   IL.6  IL.10  IL.13  IL1RN  CASP1  CXCL9   IDO1  IRGM1    MPO 
##  "pmm"  "pmm"  "pmm"  "pmm"  "pmm"  "pmm"  "pmm"  "pmm"  "pmm"  "pmm"  "pmm" 
##   MUC2 MUC5AC  MYD88   NCR1   PRF1 RETNLB  SOCS1 TICAM1    TNF 
##  "pmm"  "pmm"  "pmm"  "pmm"  "pmm"  "pmm"  "pmm"  "pmm"  "pmm" 
## PredictorMatrix:
##       IFNy CXCR3 IL.6 IL.10 IL.13 IL1RN CASP1 CXCL9 IDO1 IRGM1 MPO MUC2 MUC5AC
## IFNy     0     1    1     1     1     1     1     1    1     1   1    1      1
## CXCR3    1     0    1     1     1     1     1     1    1     1   1    1      1
## IL.6     1     1    0     1     1     1     1     1    1     1   1    1      1
## IL.10    1     1    1     0     1     1     1     1    1     1   1    1      1
## IL.13    1     1    1     1     0     1     1     1    1     1   1    1      1
## IL1RN    1     1    1     1     1     0     1     1    1     1   1    1      1
##       MYD88 NCR1 PRF1 RETNLB SOCS1 TICAM1 TNF
## IFNy      1    1    1      1     1      1   1
## CXCR3     1    1    1      1     1      1   1
## IL.6       1    1    1      1     1      1   1
## IL.10      1    1    1      1     1      1   1
## IL.13      1    1    1      1     1      1   1
## IL1RN      1    1    1      1     1      1   1
```

```
# to check each column with imputed data
## igf$imp$IFNy

#Now we can get back the completed dataset using the complete()
completeField <- complete(igf, 1)
```

Predictive mean matching with $d = 5$ is the default in mice() for continuous data. The method is robust against misspecification of the imputation model, yet performs as well as theoretically superior methods. In the context of missing covariate data, Marshall, Altman, and Holder (2010) concluded that predictive mean matching "produced the least biased estimates and better model performance measures." Another simulation study that addressed skewed data concluded that predictive mean matching "may be the preferred approach provided that less than 50% of the cases have missing data and the missing data are not MNAR" (Marshall et al. 2010). Kleinke (2017) found that the method works well across a wide variety of scenarios, but warned the default cannot address severe skewness or small samples.

Let's compare the distributions of original and imputed data using a some useful plots. First of all we can use a scatterplot and plot Ozone against all the other variables Let's first plot the variables for which we have few missing values

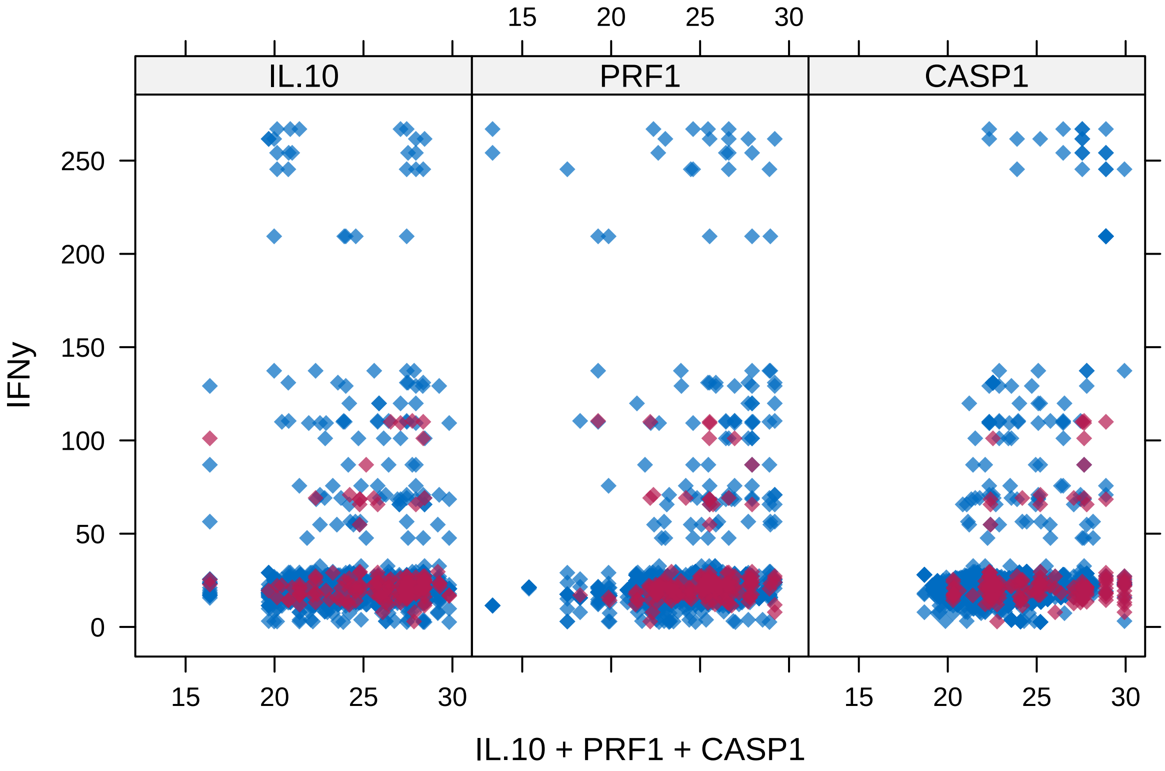
```
xyplot(igf,IFNy ~ SOCS1 + IRGM1 + MUC2, pch=18,cex=1)
```



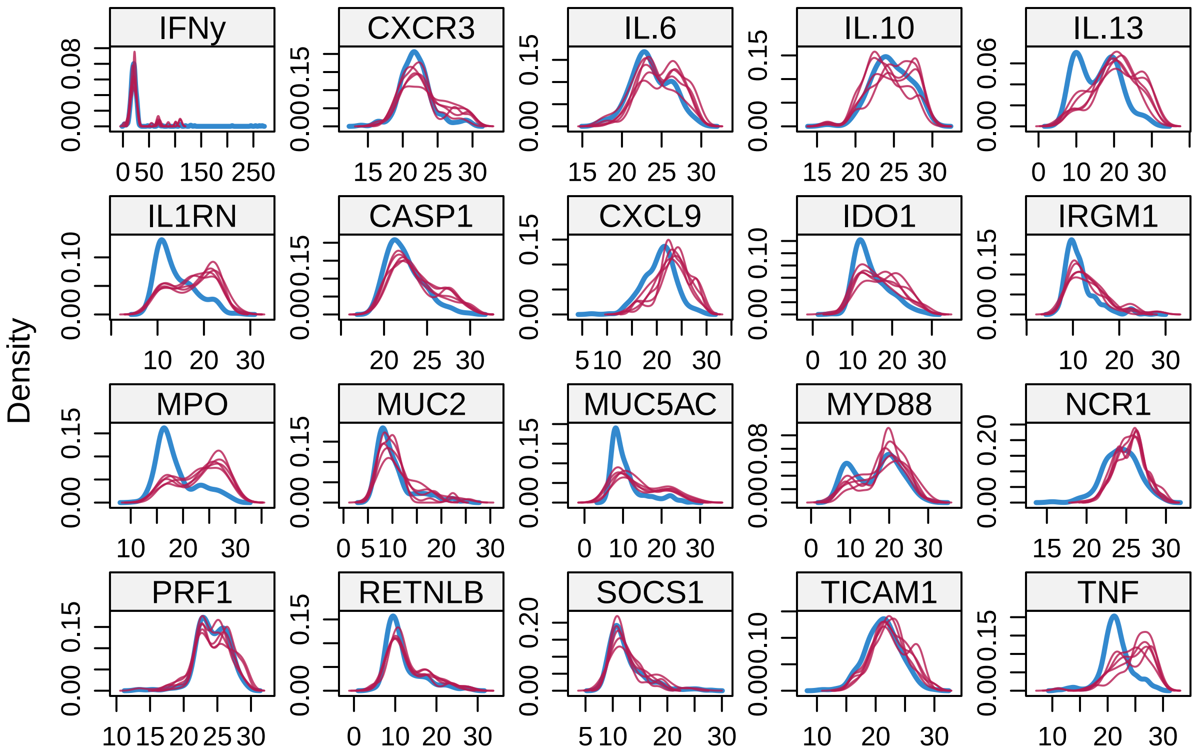
What we would like to see is that the shape of the magenta points (imputed) matches the shape of the blue ones (observed). The matching shape tells us that the imputed values are indeed "plausible values".

Now let's plot the variables with many missing data points.

```
xyplot(igf,IFNy ~ IL.10 + PRF1 + CASP1, pch=18,cex=1)
```



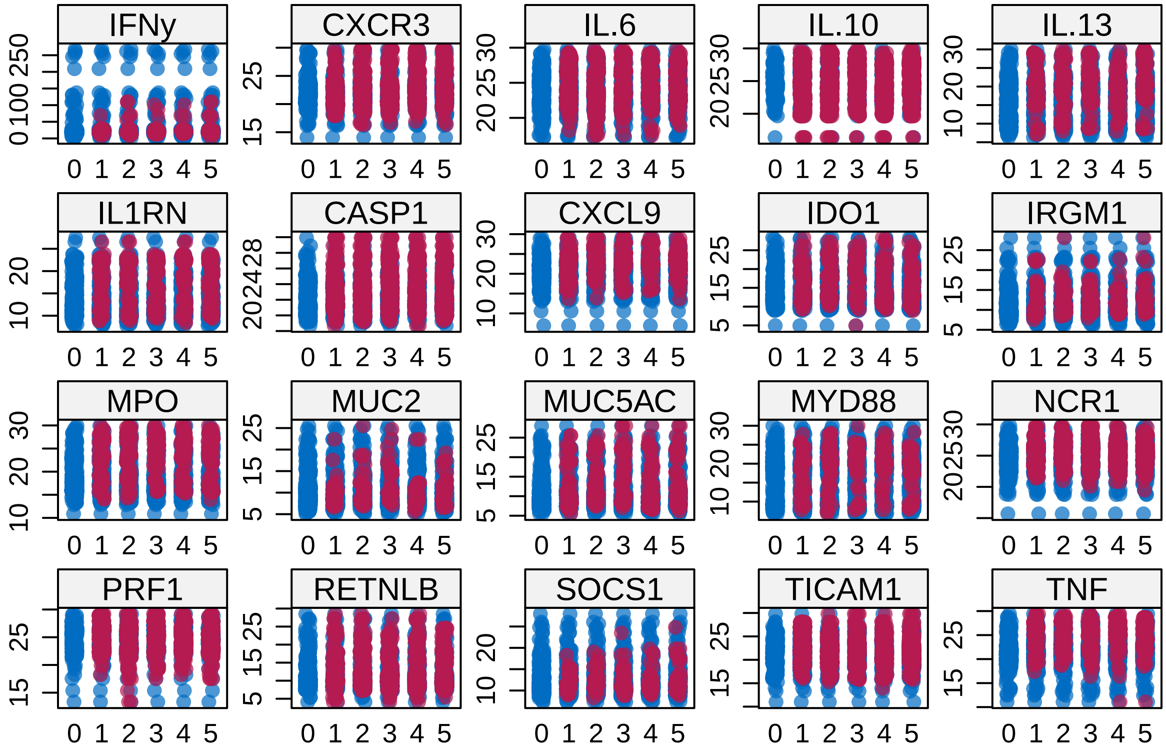
```
densityplot(igf)
```



The density of the imputed data for each imputed dataset is showed in magenta while the density of the observed data is showed in blue. Again, under our previous assumptions we expect the distributions to be similar.

Another useful visual take on the distributions can be obtained using the stripplot() function that shows the distributions of the variables as individual points

```
stripplot(igf, pch = 20, cex = 1.2)
```



Applying the model for predicting weight loss to our imputed data set

Start by making the predictions for the field data.

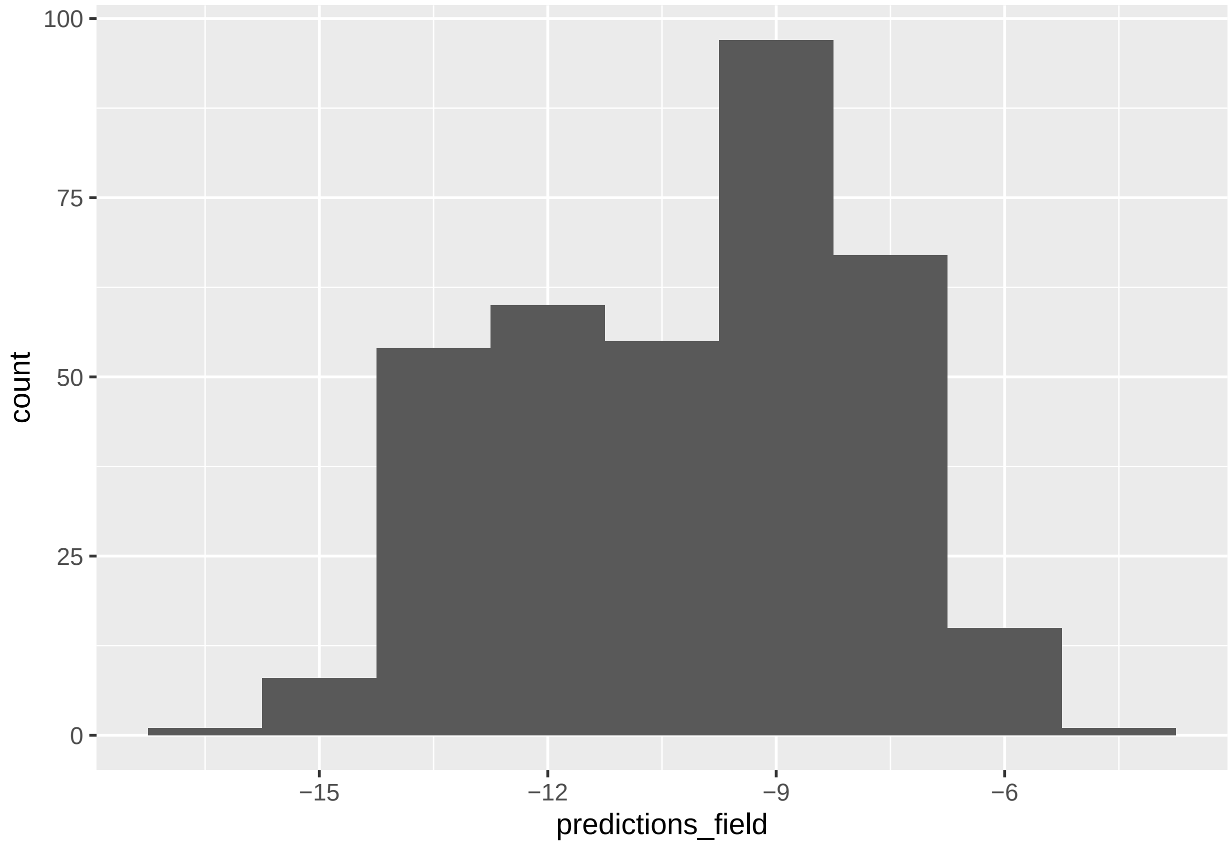
```r
# Start by selecting the columns that appear in both the training data set and the
# field data set
completeField <- completeField %>%
  dplyr::select(intersect(colnames(completeField), colnames(train.data)))

set.seed(540)

#The predict() function in R is used to predict the values based on the input data.
predictions_field <- predict(model, completeField)

# assign test.data to a new object, so that we can make changes
result_field <- completeField

#add the new variable of predictions to the result object
result_field <- cbind(result_field, predictions_field)

#add the results to a data frame containing test data and the prediction
f <- f %>%
  dplyr::select(-intersect(colnames(result_field), colnames(f)))

result_field <- cbind(f, result_field)
```

It is time to apply the package of Alice Balard et al. on our predictions!

Let's see if we indeed have differences across the hybrid index with our predicted weight loss.

Check the distribution

```
result_field %>% ggplot(aes(x = predictions_field)) +
  geom_histogram(binwidth = 1.5)
```

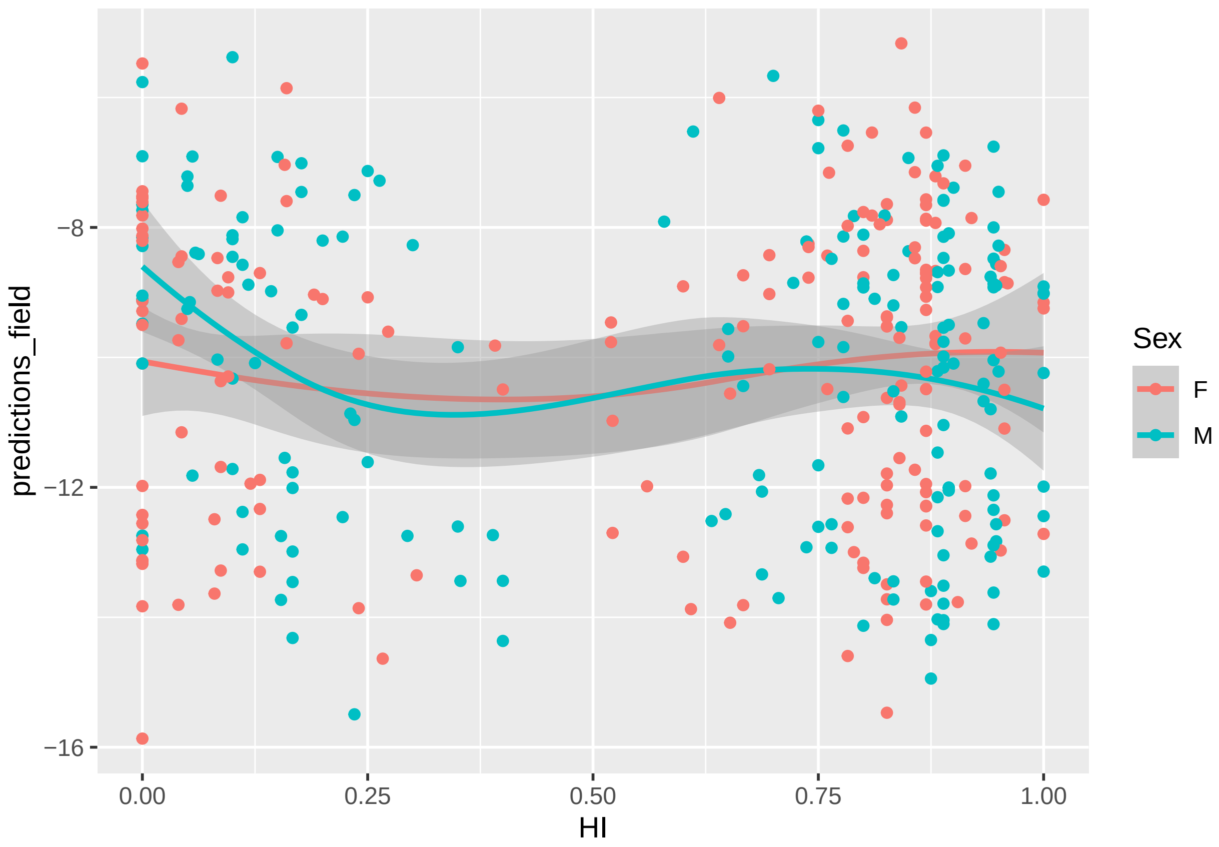


```
result_field %>%
    ggplot(aes(x = HI , y = predictions_field , color = Sex)) +
    geom_smooth() +
    geom_point()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

```
## Warning: Removed 1 rows containing non-finite values (stat_smooth).
```

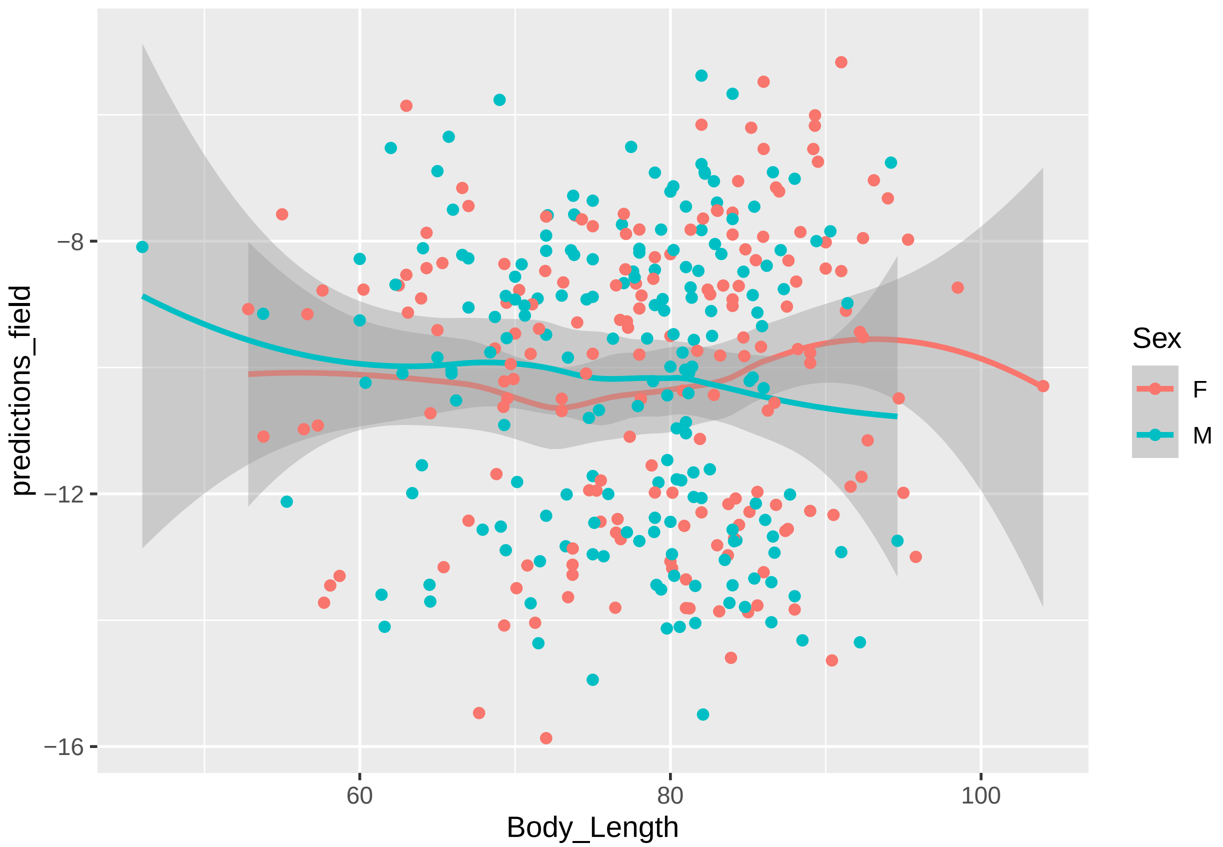
```
## Warning: Removed 1 rows containing missing values (geom_point).
```



```
result_field %>%
    ggplot(aes(x = Body_Length , y = predictions_field , color = Sex)) +
    geom_smooth() +
    geom_point()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

```
## Warning: Removed 1 rows containing non-finite values (stat_smooth).
## Removed 1 rows containing missing values (geom_point).
```



Nice to see that they are normally distributed.

Fitting distributions??

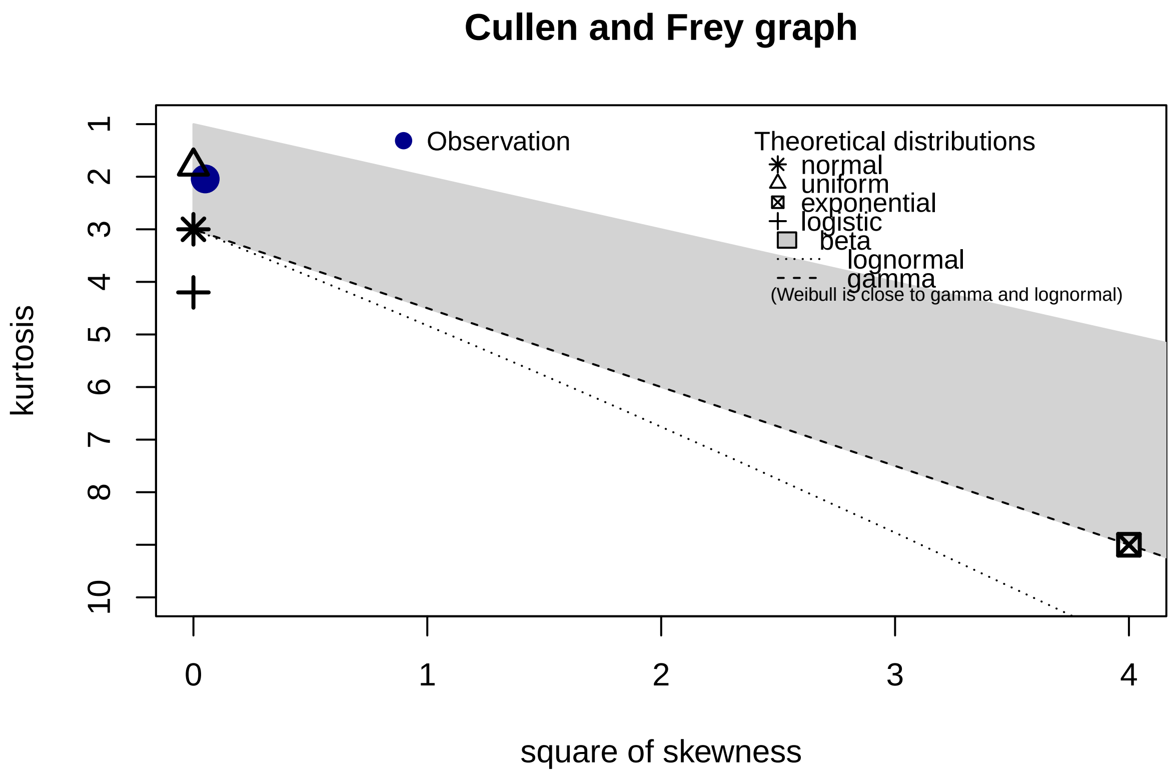
Ratios / Percentages are not normally distributed. Weibull is a good distributions.

Alice used weibull for the qpcr data. (paper)

```
library(fitdistrplus)
```

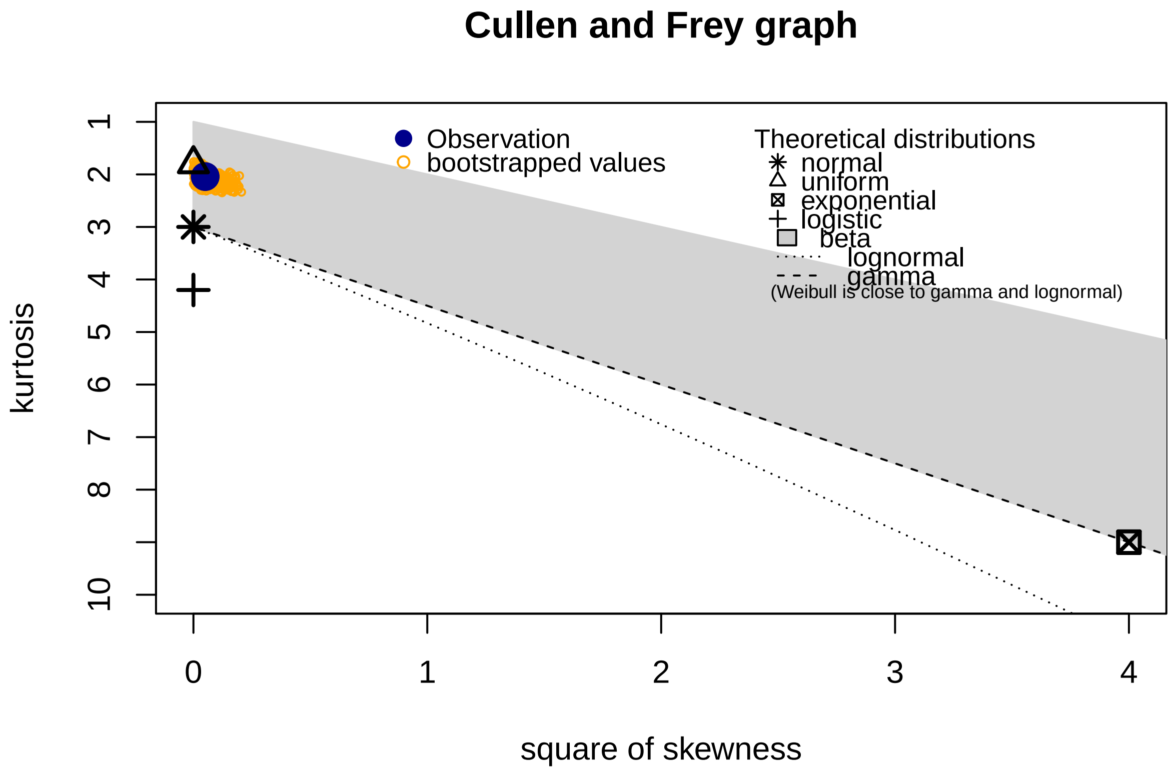
```
## Loading required package: MASS

##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select

## Loading required package: survival

##
## Attaching package: 'survival'

## The following object is masked from 'package:caret':
##
##     cluster
```

```
library(logspline)

result_field <- result_field %>%
dplyr::mutate(WL = predictions_field * (-1))

x <- result_field$WL

descdist(data = x, discrete = FALSE)
```



```
## summary statistics
## ------
## min:  5.167267   max:  15.86692
## median:  9.722013
## mean:  10.15507
## estimated sd:  2.332844
## estimated skewness:  0.2244729
## estimated kurtosis:  2.041805
```

```
descdist(data = x, discrete = FALSE, #data is continuous
         boot = 1000)
```



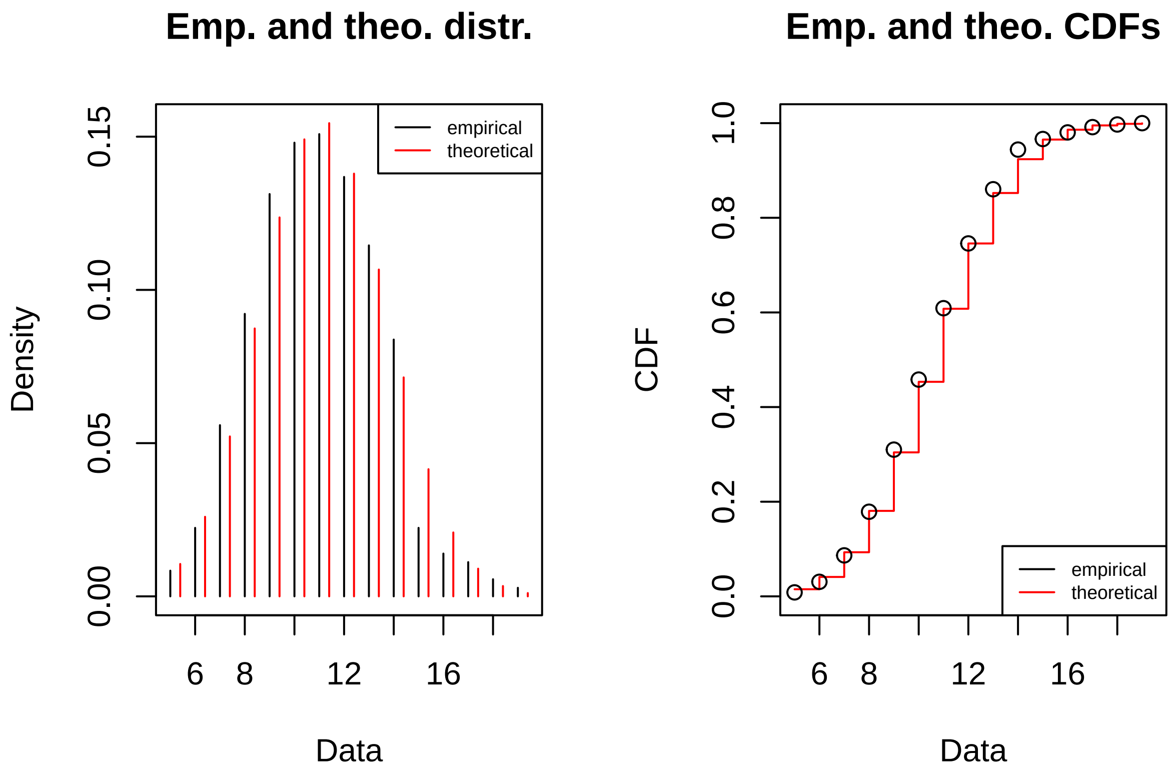
```
## summary statistics
## ------
## min:  5.167267   max:  15.86692
## median:  9.722013
## mean:  10.15507
## estimated sd:  2.332844
## estimated skewness:  0.2244729
## estimated kurtosis:  2.041805
```

Test for binomial distribution

```
set.seed(10)
n = 25
size = 27
prob = .4
data = rbinom(x, size = size, prob = prob)
fit = fitdist(data = data, dist="binom",
                   fix.arg=list(size = size),
                   start=list(prob = 0.1))

summary(fit)
```

```
## Fitting of the distribution ' binom ' by maximum likelihood
## Parameters :
##       estimate  Std. Error
## prob 0.4012002 0.004985353
## Fixed parameters:
##      value
## size    27
## Loglikelihood:  -832.1786   AIC:  1666.357   BIC:  1670.238
```

```
plot(fit)
```



```
normal_ <- fitdist(x, "norm")
weibull_ <- fitdist(x, "weibull")
gamma_ <- fitdist(x, "gamma")
```

```
library(fitdistrplus) # evaluate distribution

# Define function to be used to test, get the log lik and aic
tryDistrib <- function(x, distrib){
  # deals with fitdistr error:
  fit <- tryCatch(MASS::fitdistr(x, distrib), error=function(err) "fit failed")
  return(list(fit = fit,
              loglik = tryCatch(fit$loglik, error=function(err) "no loglik computed"),
              AIC = tryCatch(fit$aic, error=function(err) "no aic computed")))
}
```

```
findGoodDist <- function(x, distribs, distribs2){
  l =lapply(distribs, function(i) tryDistrib(x, i))
  names(l) <- distribs
  print(l)
  listDistr <- lapply(distribs2, function(i){
    if (i %in% "t"){
      fitdistrplus::fitdist(x, i, start = list(df =2))
    } else {
      fitdistrplus::fitdist(x,i)
    }}
  )
  par(mfrow=c(2,2))
  denscomp(listDistr, legendtext=distribs2)
  cdfcomp(listDistr, legendtext=distribs2)
  qqcomp(listDistr, legendtext=distribs2)
  ppcomp(listDistr, legendtext=distribs2)
  par(mfrow=c(1,1))
}
```

```
tryDistrib(x, "normal")
```

```
## $fit
##       mean            sd
##   10.15506798    2.32958395
##  ( 0.12312234) ( 0.08706064)
##
## $loglik
## [1] -810.7369
##
## $AIC
## NULL
```

```
tryDistrib(x, "binomial")
```

```
## $fit
## [1] "fit failed"
##
## $loglik
## [1] "no loglik computed"
##
## $AIC
## [1] "no aic computed"
```

```
tryDistrib(x, "student")
```

```
## $fit
## [1] "fit failed"
##
## $loglik
## [1] "no loglik computed"
##
## $AIC
## [1] "no aic computed"
```

```r
tryDistrib(x, "weibull")
```

```
## Warning in densfun(x, parm[1], parm[2], ...): NaNs produced

## Warning in densfun(x, parm[1], parm[2], ...): NaNs produced
```

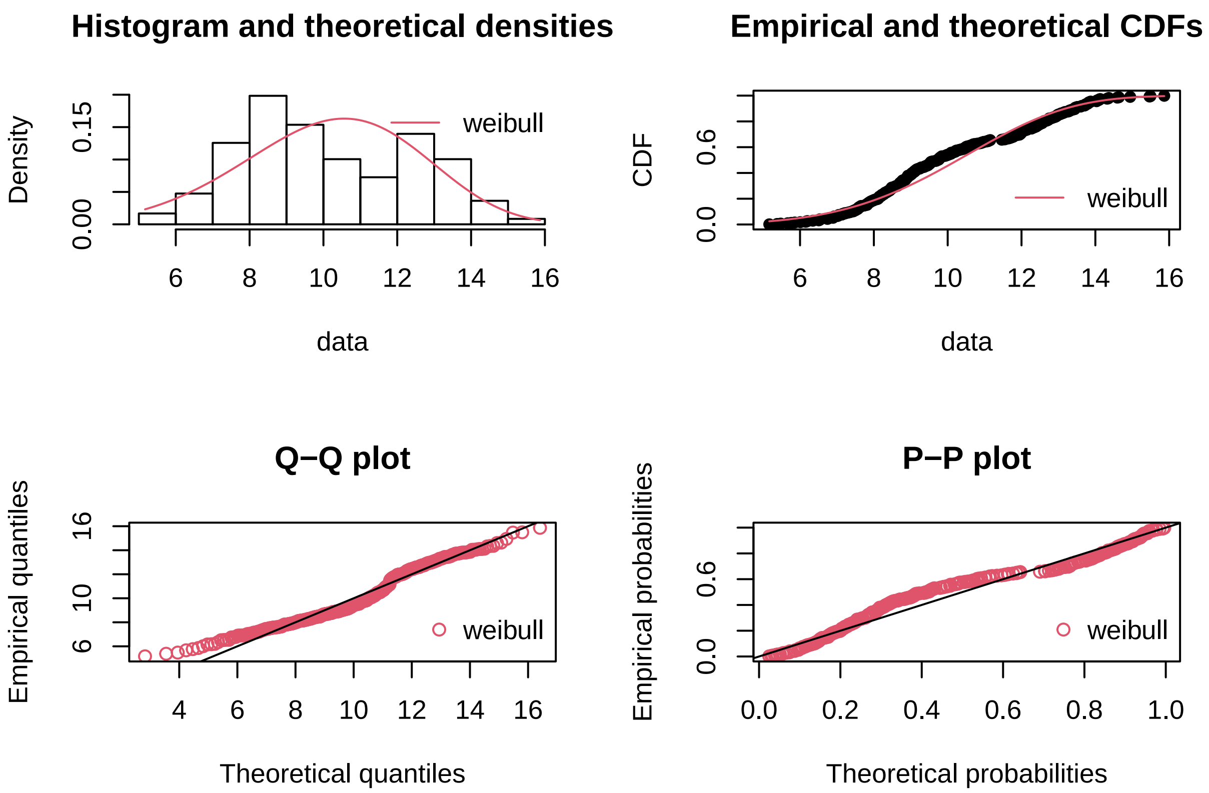
```
## $fit
##      shape        scale
##    4.8050354   11.0891250
##   ( 0.1964323) ( 0.1290913)
##
## $loglik
## [1] -813.8475
##
## $AIC
## NULL
```

```r
tryDistrib(x, "weibullshifted")
```

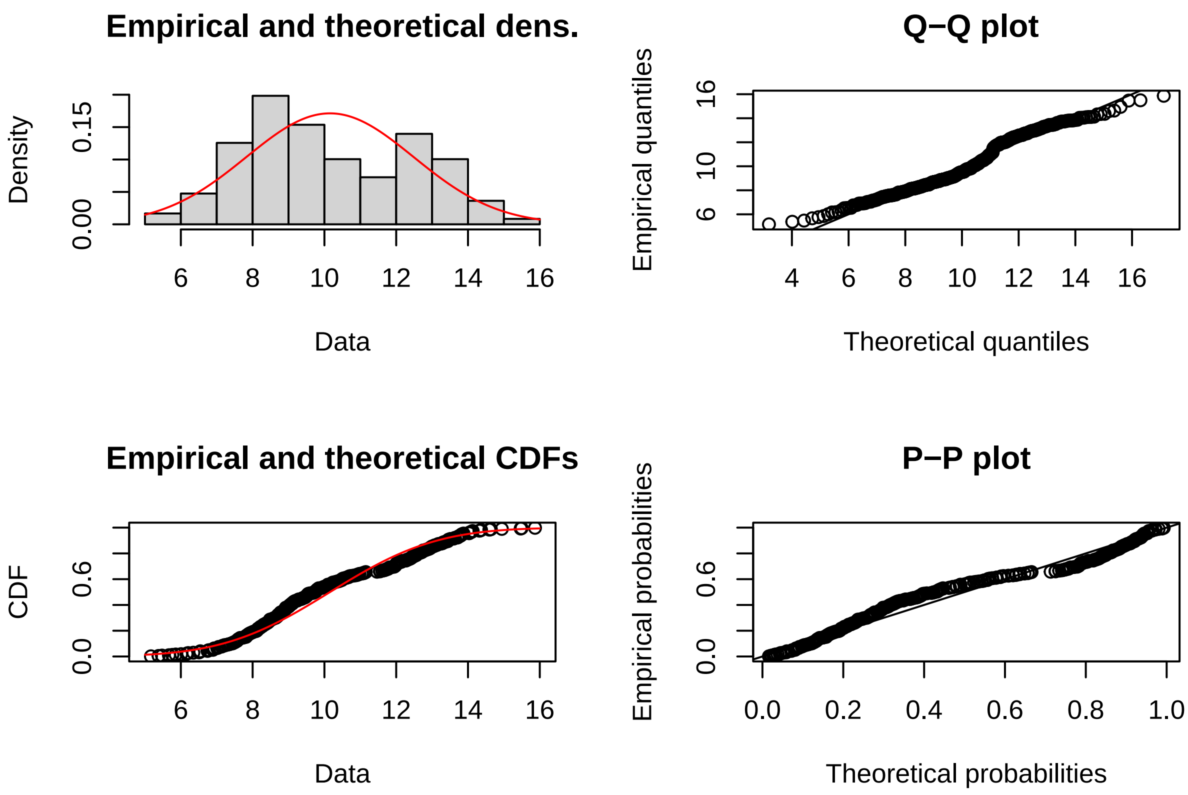
```
## $fit
## [1] "fit failed"
##
## $loglik
## [1] "no loglik computed"
##
## $AIC
## [1] "no aic computed"
```

```r
findGoodDist(x, "normal", "weibull")
```

```
## $normal
## $normal$fit
##       mean           sd
##   10.15506798    2.32958395
##  ( 0.12312234) ( 0.08706064)
##
## $normal$loglik
## [1] -810.7369
##
## $normal$AIC
## NULL
```


Histogram and theoretical densities
Density
0.00
0.15
data
6
8
10
12
14
16
weibull
Empirical and theoretical CDFs
CDF
0.0
0.6
data
6
8
10
12
14
16
weibull
Q−Q plot
Empirical quantiles
6
10
16
Theoretical quantiles
4
6
8
10
12
14
16
weibull
P−P plot
Empirical probabilities
0.0
0.6
Theoretical probabilities
0.0
0.2
0.4
0.6
0.8
1.0
weibull

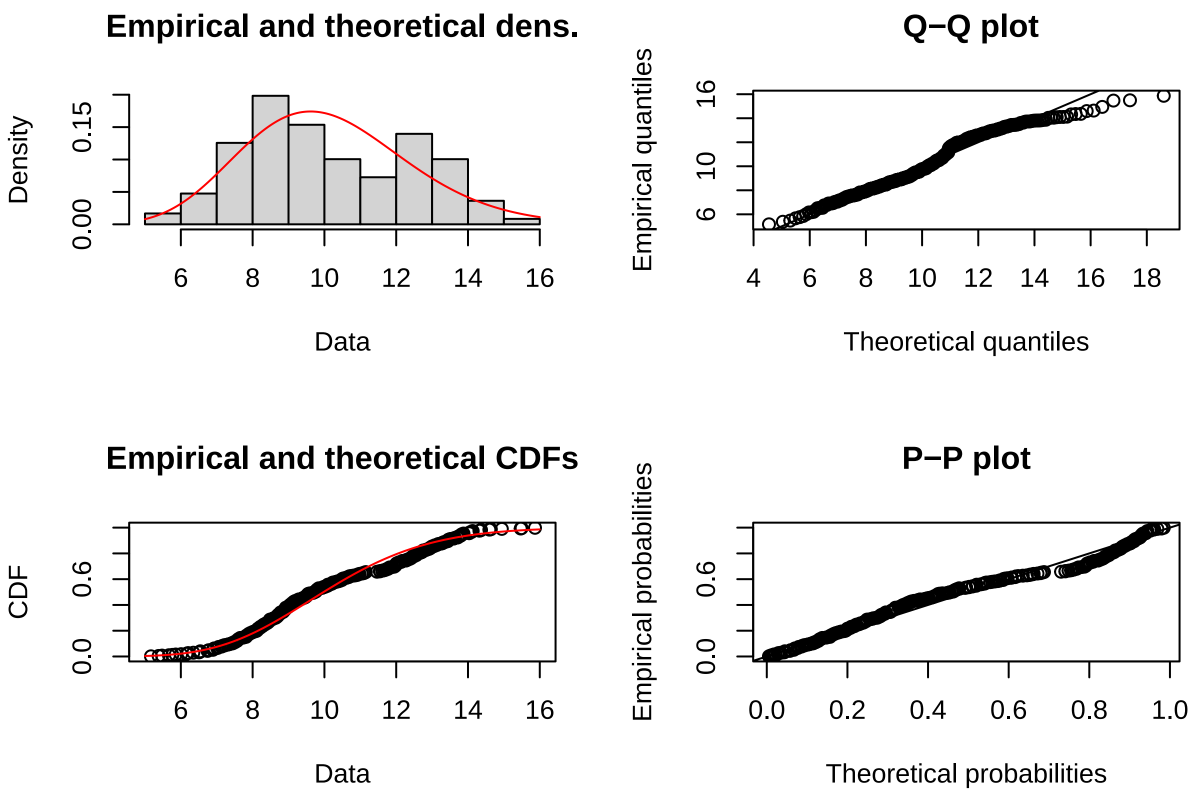
```
plot(normal_)
```



```
summary(normal_)
```

```
## Fitting of the distribution ' norm ' by maximum likelihood
## Parameters :
##       estimate Std. Error
## mean 10.155068 0.12312234
## sd    2.329584 0.08706057
## Loglikelihood:  -810.7369   AIC:  1625.474   BIC:  1633.235
## Correlation matrix:
##      mean sd
## mean    1  0
## sd      0  1
```

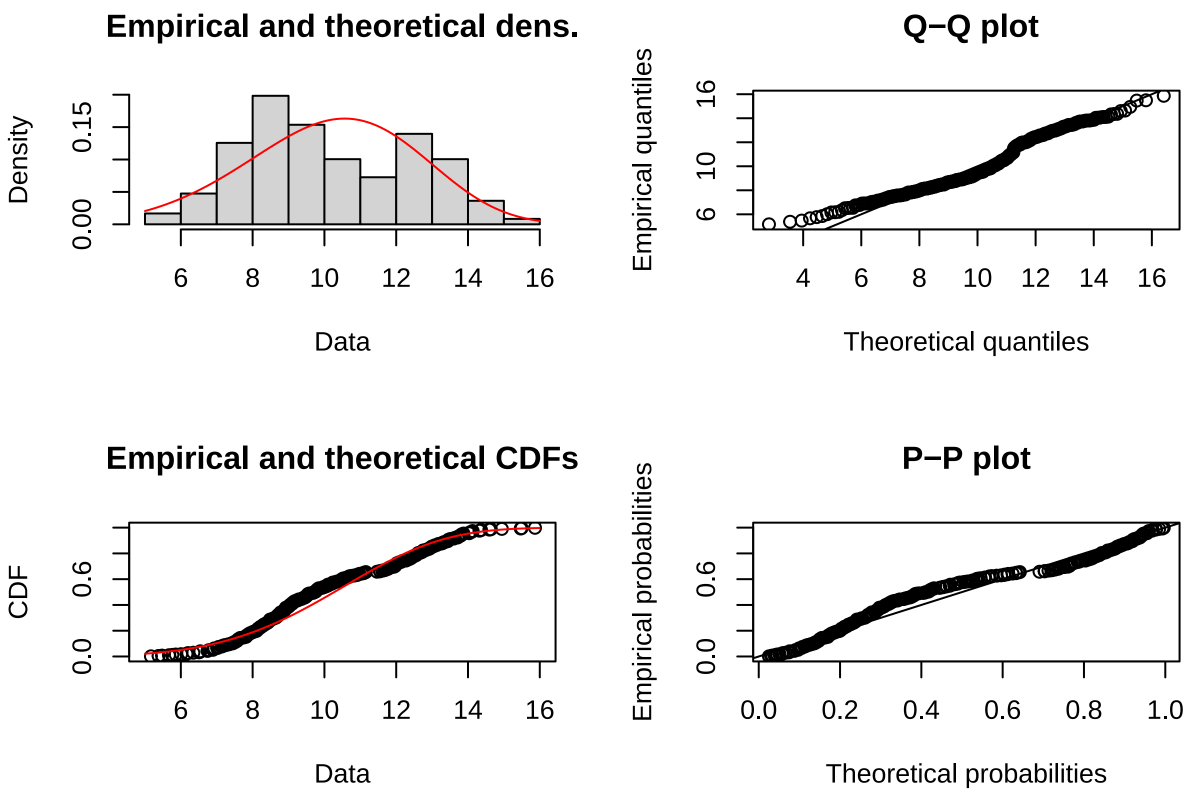
```
plot(gamma_)
```



```
summary(gamma_)
```

```
## Fitting of the distribution ' gamma ' by maximum likelihood
## Parameters :
##        estimate Std. Error
## shape 18.777962  1.3912338
## rate   1.849093  0.1388406
## Loglikelihood:  -806.4038   AIC:  1616.808   BIC:  1624.569
## Correlation matrix:
##           shape      rate
## shape 1.0000000 0.9867194
## rate  0.9867194 1.0000000
```

```
plot(weibull_)
```



```
summary(weibull_)
```

```
## Fitting of the distribution ' weibull ' by maximum likelihood
## Parameters :
##        estimate Std. Error
## shape  4.804773  0.1964228
## scale 11.089299  0.1291004
## Loglikelihood:  -813.8475   AIC:  1631.695   BIC:  1639.456
## Correlation matrix:
##           shape     scale
## shape 1.0000000 0.3273439
## scale 0.3273439 1.0000000
```

### Is alpha significant for each hypothesis?

H0: the expected load for the subspecies and between 2 groups is the same

H1: the mean load across 2 groups is the same, but can differ across subspecies

H2: the mean load across subspecies is the same, but can differ between the 2 groups

H3: the mean load can differ both across subspecies and between 2 groups

```
result_field$Sex <- as.factor(result_field$Sex)

result_field <- result_field %>%
    drop_na(HI)
```

```r
parasiteLoad::getParamBounds("weibull", data = result_field, response = "WL")
```

```
##      L1start         L1LB         L1UB      L2start         L2LB         L2UB
## 10.153594373  0.000000001 15.866922159 10.153594373  0.000000001 15.866922159
##   alphaStart      alphaLB      alphaUB myshapeStart    myshapeLB    myshapeUB
##  0.000000000 -5.000000000  5.000000000  1.000000000  0.000000001  5.000000000
```

```r
speparam <- c(L1start = 10,
                     L1LB = 1e-9,
                     L1UB = 20,
                     L2start = 10,
                     L2LB = 1e-9,
                     L2UB = 20,
                     alphaStart = 0, alphaLB = -5, alphaUB = 5,
                     myshapeStart = 1, myshapeLB = 1e-9, myshapeUB = 5)

##All
fitWL_Sex <- parasiteLoad::analyse(data = result_field,
                        response = "WL",
                        model = "weibull",
                        group = "Sex")
```

```
## [1] "Analysing data for response: WL"
## [1] "Fit for the response: WL"
## [1] "Fitting for all"
## [1] "Fitting model basic without alpha"
## [1] "Did converge"
## [1] "Fitting model basic with alpha"
## [1] "Did converge"
## [1] "Fitting model advanced without alpha"
## [1] "Did converge"
## [1] "Fitting model advanced with alpha"
## [1] "Did converge"
## [1] "Fitting for groupA : F"
## [1] "Fitting model basic without alpha"
## [1] "Did converge"
## [1] "Fitting model basic with alpha"
## [1] "Did converge"
## [1] "Fitting model advanced without alpha"
## [1] "Did converge"
## [1] "Fitting model advanced with alpha"
## [1] "Did converge"
## [1] "Fitting for groupB : M"
## [1] "Fitting model basic without alpha"
## [1] "Did converge"
## [1] "Fitting model basic with alpha"
## [1] "Did converge"
## [1] "Fitting model advanced without alpha"
## [1] "Did converge"
## [1] "Fitting model advanced with alpha"
## [1] "Did converge"
## [1] "Testing H0 no alpha vs alpha"
##    dLL dDF    pvalue
## 1 1.26   1 0.1121067
## [1] "Testing H1 no alpha vs alpha"
##    dLL dDF    pvalue
## 1 1.21   1 0.1203272
## [1] "Testing H2 groupA no alpha vs alpha"
##    dLL dDF    pvalue
## 1 0.06   1 0.7284464
## [1] "Testing H2 groupB no alpha vs alpha"
##    dLL dDF     pvalue
## 1 1.98   1 0.04635762
## [1] "Testing H3 groupA no alpha vs alpha"
##    dLL dDF    pvalue
## 1 0.33   1 0.4160676
## [1] "Testing H3 groupB no alpha vs alpha"
##    dLL dDF     pvalue
## 1 2.61   1 0.02237828
## [1] "Testing H1 vs H0"
##    dLL dDF    pvalue
## 1 0.01   1 0.8913727
## [1] "Testing H2 vs H0"
##    dLL dDF    pvalue
## 1 0.86   3 0.6343848
## [1] "Testing H3 vs H1"
##    dLL dDF    pvalue
## 1 3.56   4 0.1294703
## [1] "Testing H3 vs H2"
##    dLL dDF     pvalue
## 1 2.72   2 0.06616066
```

```
fitWL_Sex
```

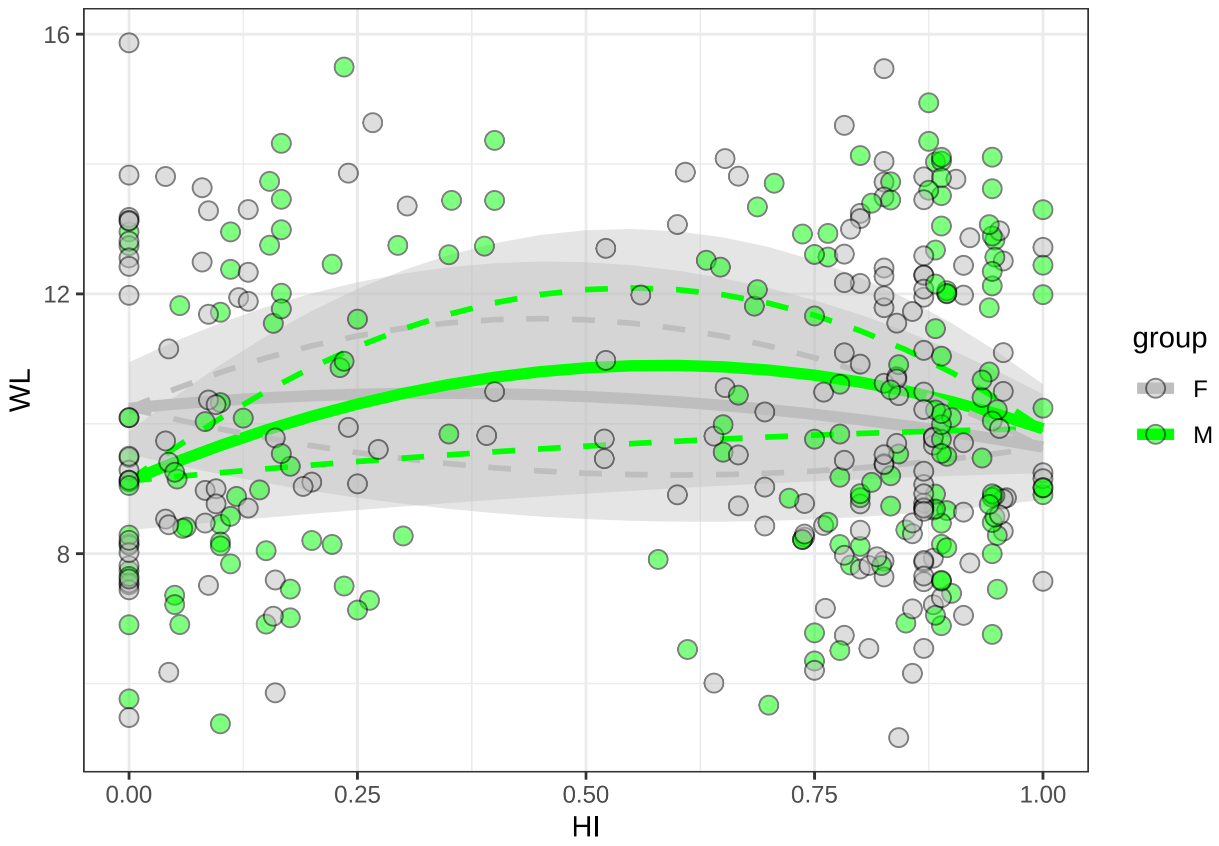
```
## $H0
##
## Call:
## bbmle::mle2(minuslogl = response ~ dweibull(shape = myshape,
##     scale = MeanLoad(L1, L1, alpha, HI)/gamma(1 + (1/myshape))),
##     start = start, method = config$method, optimizer = config$optimizer,
##     data = data, lower = c(L1 = paramBounds[["L1LB"]], alpha = paramBounds[["alphaLB"]],
##         myshape = paramBounds[["myshapeLB"]]), upper = c(L1 = paramBounds[["L1UB"]],
##         alpha = paramBounds[["alphaUB"]], myshape = paramBounds[["myshapeUB"]]),
##     control = config$control)
##
## Coefficients:
##         L1      alpha    myshape
##  9.8668586 -0.1284202  4.8126675
##
## Log-likelihood: -810.77
## Best method: bobyqa
##
## $H1
##
## Call:
## bbmle::mle2(minuslogl = response ~ dweibull(shape = myshape,
##     scale = MeanLoad(L1, L2, alpha, HI)/gamma(1 + (1/myshape))),
##     start = start, method = config$method, optimizer = config$optimizer,
##     data = data, lower = c(L1 = paramBounds[["L1LB"]], L2 = paramBounds[["L2LB"]],
##         alpha = paramBounds[["alphaLB"]], myshape = paramBounds[["myshapeLB"]]),
##     upper = c(L1 = paramBounds[["L1UB"]], L2 = paramBounds[["L2UB"]],
##         alpha = paramBounds[["alphaUB"]], myshape = paramBounds[["myshapeUB"]]),
##     control = config$control)
##
## Coefficients:
##         L1         L2      alpha    myshape
##  9.8461464  9.8879260 -0.1266397  4.8120492
##
## Log-likelihood: -810.76
## Best method: bobyqa
##
## $H2
## $H2$groupA
##
## Call:
## bbmle::mle2(minuslogl = response ~ dweibull(shape = myshape,
##     scale = MeanLoad(L1, L1, alpha, HI)/gamma(1 + (1/myshape))),
##     start = start, method = config$method, optimizer = config$optimizer,
##     data = data, lower = c(L1 = paramBounds[["L1LB"]], alpha = paramBounds[["alphaLB"]],
##         myshape = paramBounds[["myshapeLB"]]), upper = c(L1 = paramBounds[["L1UB"]],
##         alpha = paramBounds[["alphaUB"]], myshape = paramBounds[["myshapeUB"]]),
##     control = config$control)
##
## Coefficients:
##          L1       alpha     myshape
## 10.02463142 -0.03702305  4.78582955
##
## Log-likelihood: -403.82
## Best method: bobyqa
##
## $H2$groupB
##
## Call:
## bbmle::mle2(minuslogl = response ~ dweibull(shape = myshape,
##     scale = MeanLoad(L1, L1, alpha, HI)/gamma(1 + (1/myshape))),
##     start = start, method = config$method, optimizer = config$optimizer,
##     data = data, lower = c(L1 = paramBounds[["L1LB"]], alpha = paramBounds[["alphaLB"]],
##         myshape = paramBounds[["myshapeLB"]]), upper = c(L1 = paramBounds[["L1UB"]],
##         alpha = paramBounds[["alphaUB"]], myshape = paramBounds[["myshapeUB"]]),
##     control = config$control)
##
## Coefficients:
##         L1      alpha    myshape
##  9.6609835 -0.2479875  4.8717470
##
## Log-likelihood: -406.09
## Best method: bobyqa
##
##
## $H3
## $H3$groupA
##
## Call:
## bbmle::mle2(minuslogl = response ~ dweibull(shape = myshape,
##     scale = MeanLoad(L1, L2, alpha, HI)/gamma(1 + (1/myshape))),
##     start = start, method = config$method, optimizer = config$optimizer,
##     data = data, lower = c(L1 = paramBounds[["L1LB"]], L2 = paramBounds[["L2LB"]],
##         alpha = paramBounds[["alphaLB"]], myshape = paramBounds[["myshapeLB"]]),
##     upper = c(L1 = paramBounds[["L1UB"]], L2 = paramBounds[["L2UB"]],
##         alpha = paramBounds[["alphaUB"]], myshape = paramBounds[["myshapeUB"]]),
##     control = config$control)
##
## Coefficients:
##          L1          L2       alpha     myshape
## 10.23938519  9.64263827 -0.09630871  4.81591546
##
## Log-likelihood: -403
## Best method: bobyqa
##
## $H3$groupB
##
## Call:
## bbmle::mle2(minuslogl = response ~ dweibull(shape = myshape,
##     scale = MeanLoad(L1, L2, alpha, HI)/gamma(1 + (1/myshape))),
##     start = start, method = config$method, optimizer = config$optimizer,
##     data = data, lower = c(L1 = paramBounds[["L1LB"]], L2 = paramBounds[["L2LB"]],
##         alpha = paramBounds[["alphaLB"]], myshape = paramBounds[["myshapeLB"]]),
##     upper = c(L1 = paramBounds[["L1UB"]], L2 = paramBounds[["L2UB"]],
##         alpha = paramBounds[["alphaUB"]], myshape = paramBounds[["myshapeUB"]]),
##     control = config$control)
##
## Coefficients:
##         L1         L2      alpha    myshape
##  9.1198785  9.9236196 -0.2813268  4.9194178
##
## Log-likelihood: -404.2
## Best method: bobyqa
```

```
plot_WL_Sex<- bananaPlot(mod = fitWL_Sex$H3,
             data = result_field,
             response = "WL",
             group = "Sex") +
  scale_fill_manual(values = c("grey", "green")) +
  scale_color_manual(values = c("grey", "green")) +
  theme_bw()
```

```
## Scale for 'fill' is already present. Adding another scale for 'fill', which
## will replace the existing scale.
```

```
## Scale for 'colour' is already present. Adding another scale for 'colour',
## which will replace the existing scale.
```

```
plot_WL_Sex
```



Summary stats for the field

Can we test the hybrid index, WL and the infection ?

```
result_field %>%
    dplyr::group_by(MC.Eimeria) %>%
    summarize(length(Mouse_ID))
```

```
## # A tibble: 3 x 2
##   MC.Eimeria ‘length(Mouse_ID)‘
##   <lgl>                   <int>
## 1 FALSE                     111
## 2 TRUE                       92
## 3 NA                        154
```

```
result_field %>%
    dplyr::group_by(eimeriaSpecies) %>%
    summarize(length(Mouse_ID))
```

```
## # A tibble: 4 x 2
##   eimeriaSpecies ‘length(Mouse_ID)‘
##   <fct>                        <int>
## 1 E_falciformis                    8
## 2 E_ferrisi                       10
## 3 Negative                        31
## 4 <NA>                           308
```

Reproducing for melting curve

```
result_field_mc <- result_field %>%
    drop_na("MC.Eimeria")

parasiteLoad::getParamBounds("weibull", data = result_field_mc, response = "WL")
```

```
##      L1start         L1LB         L1UB      L2start         L2LB         L2UB
## 10.181285727  0.000000001 15.493176007 10.181285727  0.000000001 15.493176007
##   alphaStart       alphaLB      alphaUB myshapeStart    myshapeLB    myshapeUB
##  0.000000000 -5.000000000  5.000000000  1.000000000  0.000000001  5.000000000
```

```
speparam <- c(L1start = 10,
                     L1LB = 1e-9,
                     L1UB = 20,
                     L2start = 10,
                     L2LB = 1e-9,
                     L2UB = 20,
                     alphaStart = 0, alphaLB = -5, alphaUB = 5,
                     myshapeStart = 1, myshapeLB = 1e-9, myshapeUB = 5)


result_field_mc <- result_field_mc %>%
    dplyr::mutate(Eimeria = case_when(
        MC.Eimeria == "TRUE" ~ "positive",
        MC.Eimeria == "FALSE" ~ "negative",
        TRUE ~ ""
    ))

result_field_mc$Eimeria <- as.factor(result_field_mc$Eimeria)


##All
fitWL_Eimeria <- parasiteLoad::analyse(data = result_field_mc,
                        response = "WL",
                        model = "weibull",
                        group = "Eimeria")
```

```
## [1] "Analysing data for response: WL"
## [1] "Fit for the response: WL"
## [1] "Fitting for all"
## [1] "Fitting model basic without alpha"
## [1] "Did converge"
## [1] "Fitting model basic with alpha"
## [1] "Did converge"
## [1] "Fitting model advanced without alpha"
## [1] "Did converge"
## [1] "Fitting model advanced with alpha"
## [1] "Did converge"
## [1] "Fitting for groupA : negative"
## [1] "Fitting model basic without alpha"
## [1] "Did converge"
## [1] "Fitting model basic with alpha"
## [1] "Did converge"
## [1] "Fitting model advanced without alpha"
## [1] "Did converge"
## [1] "Fitting model advanced with alpha"
## [1] "Did converge"
## [1] "Fitting for groupB : positive"
## [1] "Fitting model basic without alpha"
```

```
## Warning in bbmle::mle2(response ~ dweibull(shape = myshape, scale =
## MeanLoad(L1, : some parameters are on the boundary: variance-covariance
## calculations based on Hessian may be unreliable
```

```
## [1] "Did converge"
## [1] "Fitting model basic with alpha"
```

```
## Warning in bbmle::mle2(response ~ dweibull(shape = myshape, scale =
## MeanLoad(L1, : some parameters are on the boundary: variance-covariance
## calculations based on Hessian may be unreliable
```

```
## [1] "Did converge"
## [1] "Fitting model advanced without alpha"
```

```
## Warning in bbmle::mle2(response ~ dweibull(shape = myshape, scale =
## MeanLoad(L1, : some parameters are on the boundary: variance-covariance
## calculations based on Hessian may be unreliable
```

```
## [1] "Did converge"
## [1] "Fitting model advanced with alpha"
```

```
## Warning in bbmle::mle2(response ~ dweibull(shape = myshape, scale =
## MeanLoad(L1, : some parameters are on the boundary: variance-covariance
## calculations based on Hessian may be unreliable
```

```
## [1] "Did converge"
## [1] "Testing H0 no alpha vs alpha"
##    dLL dDF     pvalue
## 1 1.71   1 0.06453548
## [1] "Testing H1 no alpha vs alpha"
##    dLL dDF    pvalue
## 1 1.69   1 0.0661018
## [1] "Testing H2 groupA no alpha vs alpha"
##    dLL dDF     pvalue
## 1 1.47   1 0.08612194
## [1] "Testing H2 groupB no alpha vs alpha"
##    dLL dDF    pvalue
## 1 0.11   1 0.6442281
## [1] "Testing H3 groupA no alpha vs alpha"
##    dLL dDF     pvalue
## 1 1.48   1 0.08548109
## [1] "Testing H3 groupB no alpha vs alpha"
##    dLL dDF   pvalue
## 1 0.08   1 0.681768
## [1] "Testing H1 vs H0"
##    dLL dDF    pvalue
## 1 0.02   1 0.8365323
## [1] "Testing H2 vs H0"
##    dLL dDF    pvalue
## 1 1.84   3 0.2980282
## [1] "Testing H3 vs H1"
##    dLL dDF pvalue
## 1 1.91   4   0.43
## [1] "Testing H3 vs H2"
##    dLL dDF    pvalue
## 1 0.09   2 0.9102882
```

```
fitWL_Eimeria
```

```
## $H0
##
## Call:
## bbmle::mle2(minuslogl = response ~ dweibull(shape = myshape,
##     scale = MeanLoad(L1, L1, alpha, HI)/gamma(1 + (1/myshape))),
##     start = start, method = config$method, optimizer = config$optimizer,
##     data = data, lower = c(L1 = paramBounds[["L1LB"]], alpha = paramBounds[["alphaLB"]],
##         myshape = paramBounds[["myshapeLB"]]), upper = c(L1 = paramBounds[["L1UB"]],
##         alpha = paramBounds[["alphaUB"]], myshape = paramBounds[["myshapeUB"]]),
##     control = config$control)
##
## Coefficients:
##         L1      alpha    myshape
##  9.6979193 -0.2274891  4.8290351
##
## Log-likelihood: -461.36
## Best method: bobyqa
##
## $H1
##
## Call:
## bbmle::mle2(minuslogl = response ~ dweibull(shape = myshape,
##     scale = MeanLoad(L1, L2, alpha, HI)/gamma(1 + (1/myshape))),
##     start = start, method = config$method, optimizer = config$optimizer,
##     data = data, lower = c(L1 = paramBounds[["L1LB"]], L2 = paramBounds[["L2LB"]],
##         alpha = paramBounds[["alphaLB"]], myshape = paramBounds[["myshapeLB"]]),
##     upper = c(L1 = paramBounds[["L1UB"]], L2 = paramBounds[["L2UB"]],
##         alpha = paramBounds[["alphaUB"]], myshape = paramBounds[["myshapeUB"]]),
##     control = config$control)
##
## Coefficients:
```

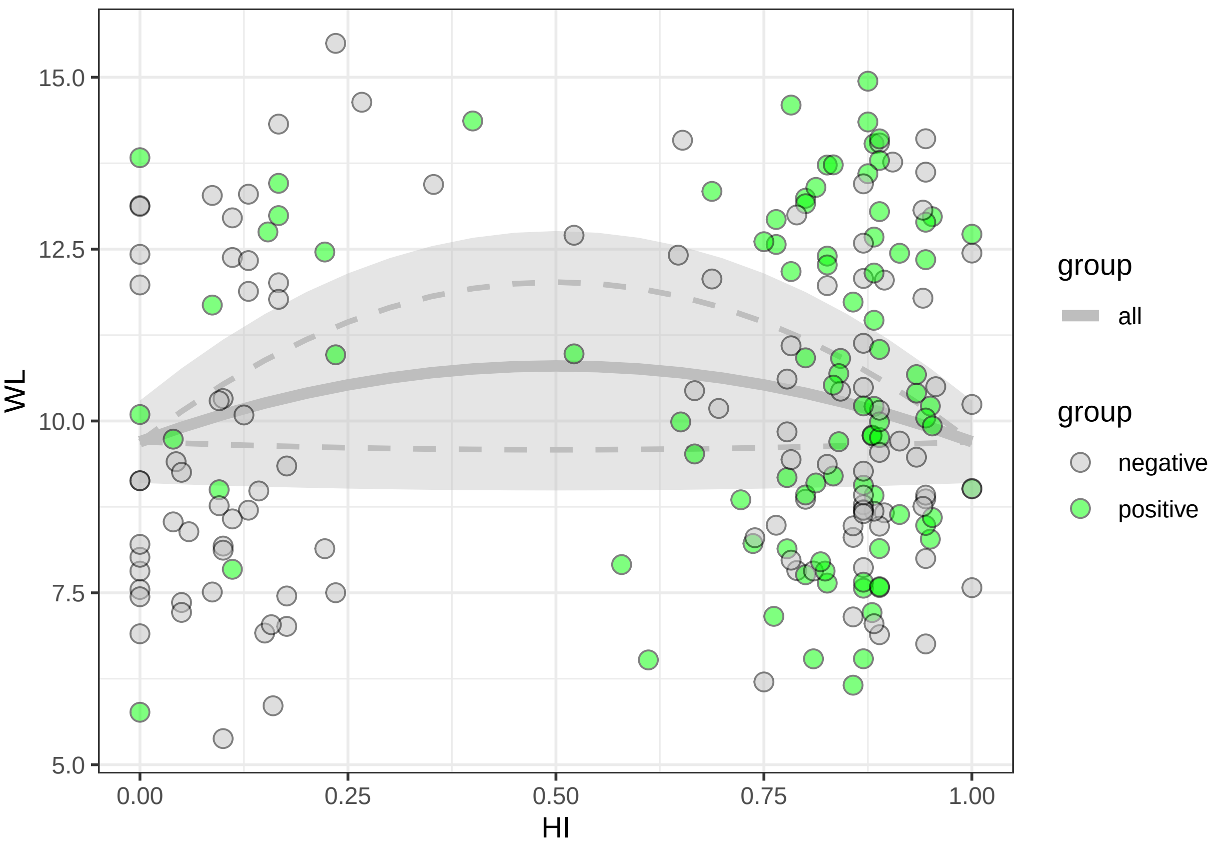
```
##         L1         L2      alpha    myshape
##  9.6496847  9.7324970 -0.2256719  4.8281459
##
## Log-likelihood: -461.34
## Best method: bobyqa
##
## $H2
## $H2$groupA
##
## Call:
## bbmle::mle2(minuslogl = response ~ dweibull(shape = myshape,
##     scale = MeanLoad(L1, L1, alpha, HI)/gamma(1 + (1/myshape))),
##     start = start, method = config$method, optimizer = config$optimizer,
##     data = data, lower = c(L1 = paramBounds[["L1LB"]], alpha = paramBounds[["alphaLB"]],
##         myshape = paramBounds[["myshapeLB"]]), upper = c(L1 = paramBounds[["L1UB"]],
##         alpha = paramBounds[["alphaUB"]], myshape = paramBounds[["myshapeUB"]]),
##     control = config$control)
##
## Coefficients:
##         L1      alpha    myshape
##  9.3543379 -0.2715782  4.6722155
##
## Log-likelihood: -250.73
## Best method: bobyqa
##
## $H2$groupB
##
## Call:
## bbmle::mle2(minuslogl = response ~ dweibull(shape = myshape,
##     scale = MeanLoad(L1, L1, alpha, HI)/gamma(1 + (1/myshape))),
##     start = start, method = config$method, optimizer = config$optimizer,
##     data = data, lower = c(L1 = paramBounds[["L1LB"]], alpha = paramBounds[["alphaLB"]],
##         myshape = paramBounds[["myshapeLB"]]), upper = c(L1 = paramBounds[["L1UB"]],
##         alpha = paramBounds[["alphaUB"]], myshape = paramBounds[["myshapeUB"]]),
##     control = config$control)
##
## Coefficients:
##          L1       alpha     myshape
## 10.30191439 -0.08948953  5.00000000
##
## Log-likelihood: -208.79
## Best method: bobyqa
##
##
## $H3
## $H3$groupA
##
## Call:
## bbmle::mle2(minuslogl = response ~ dweibull(shape = myshape,
##     scale = MeanLoad(L1, L2, alpha, HI)/gamma(1 + (1/myshape))),
##     start = start, method = config$method, optimizer = config$optimizer,
##     data = data, lower = c(L1 = paramBounds[["L1LB"]], L2 = paramBounds[["L2LB"]],
##         alpha = paramBounds[["alphaLB"]], myshape = paramBounds[["myshapeLB"]]),
##     upper = c(L1 = paramBounds[["L1UB"]], L2 = paramBounds[["L2UB"]],
##         alpha = paramBounds[["alphaUB"]], myshape = paramBounds[["myshapeUB"]]),
##     control = config$control)
##
## Coefficients:
##         L1         L2      alpha    myshape
##  9.3806551  9.3256920 -0.2731113  4.6736797
##
## Log-likelihood: -250.73
## Best method: bobyqa
##
## $H3$groupB
##
## Call:
## bbmle::mle2(minuslogl = response ~ dweibull(shape = myshape,
##     scale = MeanLoad(L1, L2, alpha, HI)/gamma(1 + (1/myshape))),
##     start = start, method = config$method, optimizer = config$optimizer,
##     data = data, lower = c(L1 = paramBounds[["L1LB"]], L2 = paramBounds[["L2LB"]],
##         alpha = paramBounds[["alphaLB"]], myshape = paramBounds[["myshapeLB"]]),
##     upper = c(L1 = paramBounds[["L1UB"]], L2 = paramBounds[["L2UB"]],
##         alpha = paramBounds[["alphaUB"]], myshape = paramBounds[["myshapeUB"]]),
##     control = config$control)
##
## Coefficients:
##          L1          L2       alpha     myshape
## 10.57317690 10.23286696 -0.08115155  5.00000000
##
## Log-likelihood: -208.7
## Best method: bobyqa
```

```
plot_WL_Eimeria <- bananaPlot(mod = fitWL_Eimeria$H0,
             data = result_field_mc,
             response = "WL",
             group = "Eimeria") +
  scale_fill_manual(values = c("grey", "green")) +
  scale_color_manual(values = c("grey", "green")) +
  theme_bw()
```

```
## Scale for 'fill' is already present. Adding another scale for 'fill', which
## will replace the existing scale.
```

```
## Scale for 'colour' is already present. Adding another scale for 'colour',
## which will replace the existing scale.
```

```
plot_WL_Eimeria
```


15.0
12.5
10.0
7.5
5.0
WL
0.00
0.25
0.50
0.75
1.00
HI
group
all
group
negative
positive