Performing Multiple Imputation in R

Lena Kristin Bache-Mathiesen

Contents

Introduction

This document is intended as a guide for imputing missing training load observations using multiple imputation with predicted mean matching. Recommendations in Flexible Imputation of Missing Data by Stef van Buuren were followed in the missing data study, and will provide the basis for this guide. For å general guide on performing mulitple imputation in R, see: https://stefvanbuuren.name/fimd/workflow.html

Data preparation

First, load required packages. Multivariate Imputation by Chained Equations (MICE) is for multiple imputation. `tidyverse` is for miscellaneous functions for reading data and working with list objects.

```
library(mice)
library(tidyverse)
```

Next step is to load your data. Here we load the anonymised football data that are available as supplementary to the study. We add fake injuries to the data with a predetermined relationship with training load. We add missing to the datasets at random so we can impute them in the next step. If you have your own dataset, you can simply load the data and skip ahead to the next chapter.

```
d_rpe_full = read_delim("norwegian_premier_league_football_rpe_anon.csv", delim = ";")
d_td_full = read_delim("norwegian_premier_league_football_td_anon.csv", delim = ";") %>%
  rename(gps_td = total_distance_daily, gps_v4 = v4_distance_daily,
         gps_v5 = v5_distance_daily, gps_pl = player_load_daily) %>% select(-gps_day)

# we remove all inherent missing for sake of the example
d_rpe_nomissing = na.omit(d_rpe_full)
d_td_nomissing = na.omit(d_td_full)
```

We add some fake injuries with a fake relationship with training load.

```r
# logistic function
log_reg = function(tl_coef){
  res = 1 / (1 + exp(-tl_coef))
  res
}

# linear logistic regression function
# where the level of sRPE effects injury probability
inj_probability_srpe = function(srpe){
  y = log_reg(-2 + 0.003*srpe)
  y
}

# create fake injuries
d_srpe = d_rpe_nomissing %>%
  mutate(srpe = duration*rpe,
         inj_prop = inj_probability_srpe(srpe),
         injury = rbinom(length(inj_prop), 1, prob = inj_prop))

# remove variables we in theory wouldn't know about
# in a real life situation
d_srpe = d_srpe %>% dplyr::select(-inj_prop, -srpe, -starts_with("missing"))

# repeat for total distance
inj_probability_td = function(gps_td){
  y = log_reg(-2 + 0.0003*gps_td)
  y
}

d_td = d_td_nomissing %>%
  mutate(inj_prop = inj_probability_td(gps_td),
         injury = rbinom(length(inj_prop), 1, prob = inj_prop))

keyvars = c("p_id", "training_date", "mc_day", "week_nr")
d_td = d_td %>% select(all_of(keyvars), injury, starts_with("gps"), srpe)
```

Add missing to the training load variables.

```r
# adding missing for our example
# under missing completely at random
add_mcar_rpe = function(d, missing_prop){
  n_values = nrow(d)
  d = d %>% rownames_to_column()
  random_spots_rpe = sample(1:n_values, round(missing_prop*n_values))
  random_spots_min = sample(1:n_values, round(missing_prop*n_values))
  d = d %>% mutate(rpe = ifelse(rowname %in% random_spots_rpe, NA, rpe),
                   duration = ifelse(rowname %in% random_spots_min, NA, duration)) %>%
      dplyr::select(-rowname)
  d
}

# adding 25% missing randomly distributed between duration and rpe
d_rpe_missing = add_mcar_rpe(d_srpe, 0.25)
```

```r
add_mcar_td = function(d, missing_prop){
  n_values = nrow(d)
  d = d %>% rownames_to_column()
  random_spots_td = sample(1:n_values, round(missing_prop*n_values))
  d = d %>% mutate(gps_td = ifelse(rowname %in% random_spots_td, NA, gps_td)) %>%
      dplyr::select(-rowname)
  d
}

# adding 25% missing in total distance
d_td_missing = add_mcar_td(d_td, 0.25)
```

Imputing session RPE

sRPE is a derived variable. We tested different ways to impute sRPE and found out that the best method (the one with least bias) was imputing duration and rpe before calculating sRPE (Impute, then transform).

Imputation using multiple imputation and predicted mean matching code below. The default method in `mice` is PMM, and the default number of datasets is 5. For PMM, the default number of donors is also 5. However, you can change the number of datasets with the argument `m`. `print = FALSE` ensures we don't have to see each iteration printed in the console, but you can set it to `TRUE` if you would like to be assured that the command is actually running. Sometimes, MI can take a while, commonly several minutes for large datasets.

```r
mids.pmm = mice(d_rpe_missing, print = FALSE, seed = 1234, m = 5)
```

Then, we need to tell R to calculate the sRPE in our imputed objects, which is done here:

```r
imp.pmm = mice::complete(mids.pmm, "long", include = TRUE)
imp.pmm$srpe = with(imp.pmm, rpe*duration)
mids.itt.pmm = as.mids(imp.pmm)
```

We can then fit our logistic regression model, with injury as the response variable and sRPE the independent variable, using this mids object. The `pool()` command automatically pools the fitted estimates into one summary using Rubin's rules.

```r
fit.pmm =  with(mids.itt.pmm, glm(injury ~ srpe, family = binomial))
fit.pmm.pooled = pool(fit.pmm)
summary(fit.pmm.pooled, "all", conf.int = TRUE)
```

```
##          term m     estimate    std.error statistic       df p.value
## 1 (Intercept) 5 -2.005506611 0.0770036616 -26.04430 88.50906       0
## 2        srpe 5  0.002969957 0.0001838164  16.15719 63.76668       0
##          2.5 %       97.5 %       riv    lambda       fmi         ubar
## 1 -2.158522941 -1.852490281 0.2659012 0.2100489 0.2273139 4.684065e-03
## 2  0.002602716  0.003337199 0.3301355 0.2481969 0.2707172 2.540228e-08
##              b            t dfcom
## 1 1.037915e-03 5.929564e-03  4723
## 2 6.988496e-09 3.378848e-08  4723
```

If we don't want to use the `mice` package to fit our imputed datasets, we can make a list out of the imputed data and perform our model-fitting on each dataset in the list manually.

We could fit any kind of model using the regular `glm()` command from base R. Here we provide an example using a mixed model with a random intercept per player, with the `glmer()` command from the `lme4` package. The `lme4` package is not entirely compatible with the `mice` package, so we'd have to do this if we wanted to run a mixed model.

Note that the `map()` function provided by the `tidyverse` package ensures that the model is run on each dataset in the list.

```
list.pmm = mice::complete(mids.itt.pmm, "all")
# the first dataset in the list is the original, unimputed data.
# we remove this first.
list.pmm = list.pmm[-1]
# load the mixed model package and run our mixed logistic regression model
library(lme4)
fit_mixed = list.pmm %>% map(glmer, formula = injury ~ srpe + (1 | p_id), family = "binomial")
```

If you want to look at the imputed datasets in a regular dataframe, we can convert our list with the following code using the `tidyverse` package.

The `imap` command adds an index to each dataset so we can tell which row belongs to which replicated dataset in the final data.frame.

After running the code below, `d.pmm` can be used as any ordinary dataframe object.

```
list.pmm = list.pmm %>% imap(., ~mutate(., dataset_n = .y))
d.pmm = list.pmm %>%  bind_rows()
```

Imputing GPS measures (Total distance)

Imputing total distance or any other GPS measure isn't very different from imputing sRPE. The main difference is that we can skip a step, since GPS measures are not derived variables.

Imputation using multiple imputation and predicted mean matching code below. The default method in `mice` is PMM, and the default number of datasets is 5. For PMM, the default number of donors is also 5. However, you can change the number of datasets with the argument `m`. `print = FALSE` ensures we don't have to see each iteration printed in the console, but you can set it to `TRUE` if you would like to be assured that the command is actually running. Sometimes, MI can take a while, commonly several minutes for large datasets.

```
mids.pmm.gps = mice(d_td_missing, print = FALSE, seed = 1234, m = 5)
```

We can then fit our logistic regression model, with injury as the response variable and sRPE the independent variable, using this mids object. The `pool()` command automatically pools the fitted estimates into one summary using Rubin's rules.

```
fit.pmm.gps =  with(mids.pmm.gps, glm(injury ~ gps_td, family = binomial))
fit.pmm.gps.pooled = pool(fit.pmm.gps)
summary(fit.pmm.gps.pooled, "all", conf.int = TRUE)
```

```
##          term m      estimate    std.error statistic       df p.value
## 1 (Intercept) 5 -1.9109539124 0.0991470145 -19.27394 2197.712       0
## 2      gps_td 5  0.0002852687 0.0000179115  15.92657 2347.092       0
##           2.5 %        97.5 %         riv      lambda        fmi         ubar
## 1 -2.1053855700 -1.7165222547 0.014335259 0.014132663 0.01502862 9.691205e-03
## 2  0.0002501447  0.0003203927 0.009651365 0.009559107 0.01040200 3.177551e-10
##              b            t dfcom
## 1 1.157716e-04 9.830130e-03  2506
## 2 2.555642e-12 3.208219e-10  2506
```

If you have missing in multiple variables, there is no need to impute then one by one in individual commands. By default, all variables in the dataset will be used in the imputation model. Imputed observations will provide information for imputation of missing observations that have not been imputed yet.

Here is an example where we add intermittent missing in both total distance and in the high-speed running distance, then impute the data.

```
add_mcar_hsr = function(d, missing_prop){
  n_values = nrow(d)
  d = d %>% rownames_to_column()
  random_spots_td = sample(1:n_values, round(missing_prop*n_values))
  d = d %>% mutate(gps_v4 = ifelse(rowname %in% random_spots_td, NA, gps_v4)) %>%
      dplyr::select(-rowname)
  d
}

# create missing
d_td_missing_multvar = add_mcar_hsr(d_td_missing, 0.25)

# impute both total distance and high-speed running distance in the same dataset
mids.pmm.gps.multvar = mice(d_td_missing_multvar, print = FALSE, seed = 1234, m = 5)
```

We can see which methods were used to impute the data by running

```
mids.pmm.gps.multvar$method
```

```
##          p_id training_date        mc_day       week_nr        injury
##            ""            ""            ""            ""            ""
##       gps_td        gps_v4        gps_v5        gps_pl          srpe
##         "pmm"         "pmm"            ""            ""            ""
```

In our results, it shows that both total distance and gps_v4 was imputed with PMM, the default.

The predictor matrix shows which variables were used in the imputation model. A 1 indicates the variable is used to impute another, a 0 that it was not.

```
mids.pmm.gps.multvar$predictorMatrix
```

```
##               p_id training_date mc_day week_nr injury gps_td gps_v4 gps_v5
## p_id             0             1      0       0      1      1      1      1
## training_date    1             0      0       0      1      1      1      1
## mc_day           1             1      0       0      1      1      1      1
## week_nr          1             1      0       0      1      1      1      1
## injury           1             1      0       0      0      1      1      1
## gps_td           1             1      0       0      1      0      1      1
## gps_v4           1             1      0       0      1      1      0      1
## gps_v5           1             1      0       0      1      1      1      0
## gps_pl           1             1      0       0      1      1      1      1
## srpe             1             1      0       0      1      1      1      1
##               gps_pl srpe
## p_id               1    1
## training_date      1    1
## mc_day             1    1
## week_nr            1    1
## injury             1    1
## gps_td             1    1
## gps_v4             1    1
## gps_v5             1    1
## gps_pl             0    1
## srpe               1    0
```

The predictor matrix is useful if we know certain variables should not be in the imputation model. See the

section Pitfalls and other details for more information of variables to watch out for. Below is an example where we remove the sRPE from the imputation model before we impute the data.

```
predmatrix_td = make.predictorMatrix(d_td_missing_multvar)
predmatrix_td["gps_td", "srpe"] = 0
mids.pmm.gps.nosrpe = mice(d_td_missing_multvar, pred = predmatrix_td, print = FALSE, seed = 1234)
```

Pitfalls and other details

Imputation model specification

In general, misspecification of the imputation model can affect the estimates. Predicted mean matching (PMM) is fairly robust to misspecification compared to other methods (Morris, White, Royston 2014, https://doi.org/10.1186/1471-2288-14-75). However, it can still cause bias if the imputation model is poor. For one, three or fewer predictors have shown to decrease the performance of PMM. This may also hold true for too weak predictors. It is therefore advised to use as much information as available.

"Conditioning on all other data is often reasonable for small to medium datasets, containing up to, say, 20–30 variables, without derived variables, interactions effects and other complexities." https://stefvanbuuren.name/fimd/sec-modelform.html#sec:predictors

The following section has some advice for the case of tremendous number of predictors available: https://stefvanbuuren.name/fimd/sec-toomany.html . When in doubt, IC methods (AIC, BIC) can be used to determine the imputation model (Noghrehchi et al 2021, https://onlinelibrary.wiley.com/doi/full/10.1002/sim.8915).

The `mice()` function from the `mice` package checks for linear dependencies during the imputation iterations, and temporarily removes predictors from the imputation models where needed. This includes variables that are identified as constant or which have collinearity issues.

The default imputation model in the `MICE` package is a linear regression model. This may be inadequate in the presence of strong non-linear relationships between the predictors and the response (training load) in the imputation model (Van Buuren 2018). This can be handled by specifying Restricted Cubic Splines or Fractional Polynomials in the imputation model. Another option is to forgo PMM for more advanced methods, such as Machine Learning using a Random Forest algorithm (Benson 2021 implements Multiple Impuitation with Machine Learning https://doi.org/10.52082/jssm.2021.188).

Note that if the missing data is considered to be under the assumption of missing at random, meaning the probability of missing is dependent on other variables in the data, the variables for which the missing is dependent upon should not be in the imputation model for PMM. The degree to which this affects the estimates depends on how strongly the variable predicts missing (see 3.4.4 Pitfalls in this section: https://stefvanbuuren.name/fimd/sec-pmm.html). This is especially important for session RPE which was found to be missing at random in a study on missing RPE (Benson et al. 2021, https://www.jssm.org/researchjssm-20-188.xml.xml).

Do not fear including the study response variable (here, injury) in the imputation model. Moons et al. 2006 (https://www.sciencedirect.com/science/article/pii/S0895435606000606) shows that it improves the imputation model, and Sterne et al. 2009 highlights not including the response variable as a typical pitfall in epidemiological studies (https://www.bmj.com/content/338/bmj.b2393/). The response variable itself, however, should not be imputed unconsciously (see Peters et al. 2012 https://www.sciencedirect.com/science/article/pii/S0895435611003878).

Using injury as a predictor for imputing training load, but not imputing injury can be specified as follows:

```
method_srpe = make.method(d_rpe_missing)
method_srpe["injury"] = ""
mids.pmm.noinjury = mice(d_rpe_missing, method = method_srpe, print = FALSE, seed = 1234)
```

How much missing data is too much for multiple imputation?

The proportion of missing data should, in general, not be used to determine whether to use multiple imputation or not. So long as the imputation model is specified correctly, multiple imputation with predicted mean matching can perform even under 80% and 90% missing data (Madley-Dowd et al. 2019, https://www.sciencedirect.com/science/article/pii/S0895435618308710). Consider the case of having 100 000 rows of data, and 90% are missing. In this scenario, there are still 10 000 rows of observations that can inform an imputation model. Given that the remaining 10% of observations are representative of the population, and are of a reasonable sample size, multiple imputation can safely be performed.

However, missing training load observations may be under the assumption of Missing Not at Random (MNAR). Such missing data patterns arise when the probability of missing is either 1) dependent on unobserved variables or 2) dependent on the data that is missing. For instance, if athletes neglect to report RPE on intense training days, the data are MNAR, as the probability of missing is dependent on the RPE values themselves - the higher RPE, the higher chance of missing. This may cause selection bias. Some cases of MNAR may mimic MAR enough to justify performing multiple imputation regardless (Schouten and Vink 2018, https://doi.org/10.1177/0049124118799376), but we advise caution against performing analyses on data suspected to be MNAR.

Predicted mean matching is a method that selects values at random from a handful of so-called "donor" rows. If the sample size is too small, it risks picking the same donor repeatedly, essentially creating countless duplicates. The number of non-missing observations is therefore more important to consider than the percentage missing when determining whether to impute or not.

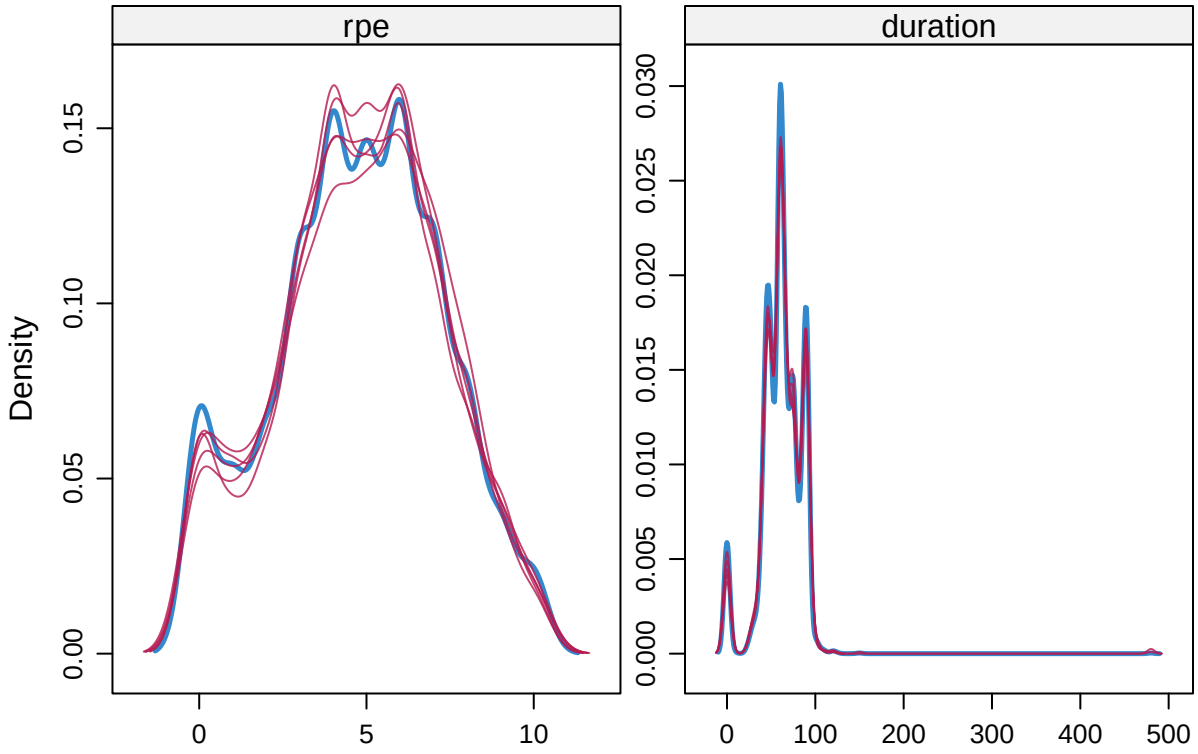
Validating the imputation

After imputing the data, it can useful to determine whether it was successful with visual inspections. This can be used to identify misspecification errors or determine in which direction the imputed values are biased.

The `mice` package comes with a very handy `densityplot()` function that compares the distribution of the real data vs. that of the imputed data. Red are the imputed datasets, blue are the original data.

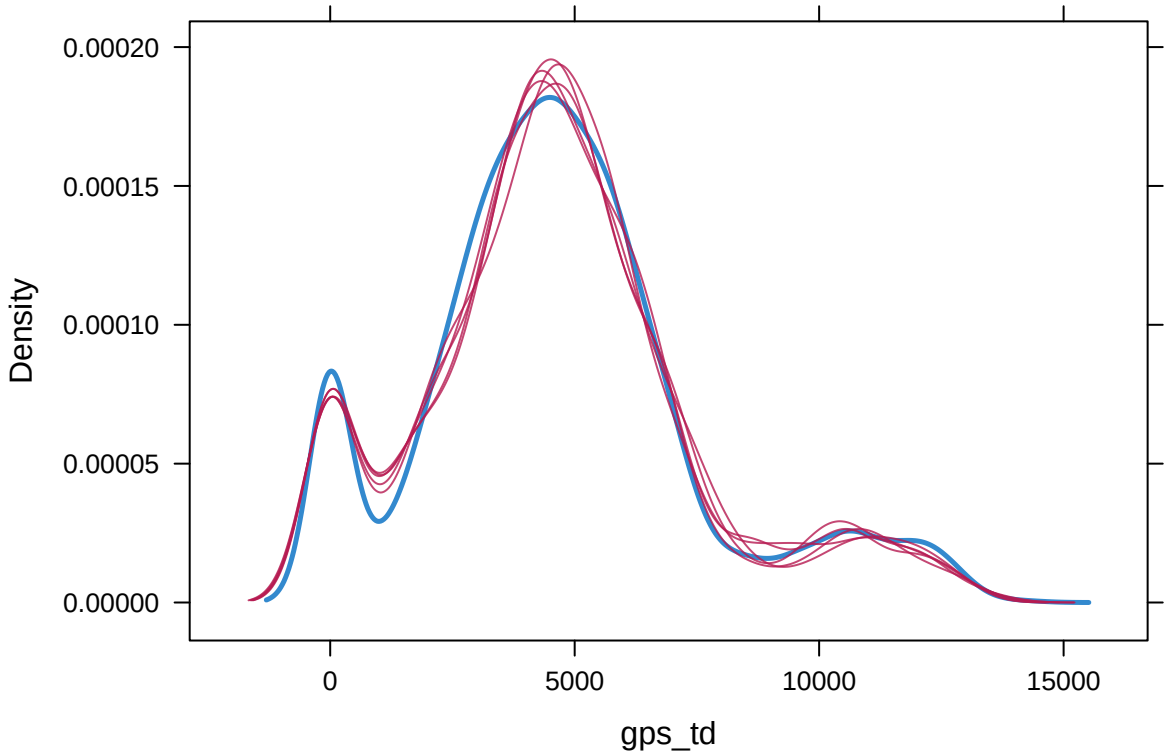
For session RPE:

```
densityplot(x=mids.itt.pmm, data = ~rpe + duration)
```



For total distance:

```
densityplot(x=mids.pmm.gps, data = ~gps_td)
```


0.00020
0.00015
0.00010
0.00005
0.00000
Density
0
5000
10000
15000
gps_td