Curso Handing

José Bustos

22/2/2023

Contents

Chapter 1 **2**

Sesión 1 . . . 2

Linear regression with incomplete data . . . 2

Actividad . . . 2

Recognizing missing data mechanisms . . . 3

t-test for MAR: data preparation . . . 3

t-test for MAR: interpretation . . . 4

Sesión 2 . . . 4

Aggregation plot . . . 4

Question . . . 5

Spine plot . . . 6

Mosaic plot . . . 7

Chapter 2 **8**

Sesión 1 . . . 8

Smelling the danger of mean imputation . . . 8

Mean-imputing the temperature . . . 9

Assessing imputation quality with margin plot . . . 10

Sesión 2 . . . 11

Vanilla hot-deck . . . 11

Hot-deck tricks & tips I: imputing within domains . . . 12

Hot-deck tricks & tips II: sorting by correlated variables . . . 14

Sesión 3 . . . 15

Choosing the number of neighbors . . . 15

kNN tricks & tips I: weighting donors . . . 17

Chapter 3 **19**

Sesión 1 19

Linear regression imputation 19

Initializing missing values & iterating over variables 20

Detecting convergence 21

Question 22

```
library(dplyr)
library(ggplot2)
biopics <- read.csv("~/edi_imp/handing/biopics.csv")
```

Chapter 1

Sesión 1

Linear regression with incomplete data

Missing data is a common problem and dealing with it appropriately is extremely important. Ignoring the missing data points or filling them incorrectly may cause the models to work in unexpected ways and cause the predictions and inferences to be biased.

In this chapter, you will be working with the `biopics` dataset. It contains information on a number of biographical movies, including their earnings, subject characteristics and some other variables. Some of the data points are, however, missing. The original data comes with the `fivethirtyeight` R package, but in this course, you will work with a slightly preprocessed version.

In this exercise, you will get to know the dataset and fit a linear regression model to explain a movie's earnings. Let's begin!

Actividad

Print the first 10 observations of the `biopics` data and get familiar with the variables.

```
# Print first 10 observations
head(biopics, 10)
```

```
##    country year earnings sub_num sub_type sub_race non_white sub_sex
## 1       UK 1971       NA       1 Criminal     <NA>         0    Male
## 2    US/UK 2013   56.700       1    Other  African         1    Male
## 3    US/UK 2010   18.300       1  Athlete     <NA>         0    Male
## 4   Canada 2014       NA       1    Other    White         0    Male
## 5       US 1998    0.537       1    Other     <NA>         0    Male
## 6       US 2008   81.200       1    Other    other         1    Male
## 7       UK 2002    1.130       1 Musician    White         0    Male
## 8       US 2013   95.000       1  Athlete  African         1    Male
## 9       US 1994   19.600       1  Athlete     <NA>         0    Male
## 10   US/UK 1987    1.080       2   Author     <NA>         0    Male
```

```r
# Get the number of missing values per variable
biopics %>%
    is.na() %>%
    colSums()
```

```
##   country      year  earnings   sub_num  sub_type  sub_race non_white   sub_sex
##         0         0       324         0         0       197         0         0
```

```r
# Fit linear regression to predict earnings
model_1 <- lm(earnings ~ country + year + sub_type,
              data = biopics)

# Fit linear regression to predict earnings
model_2 <- lm(earnings ~ country + year + sub_type + sub_race,
              data = biopics)
```

Recognizing missing data mechanisms

In this exercise, you will face six different scenarios in which some data are missing. Try assigning each of them to the most likely missing data mechanism. As a refresher, here are some general guidelines:

If the reason for missingness is purely random, it's MCAR. If the reason for missingness can be explained by another variable, it's MAR. If the reason for missingness depends on the missing value itself, it's MNAR.

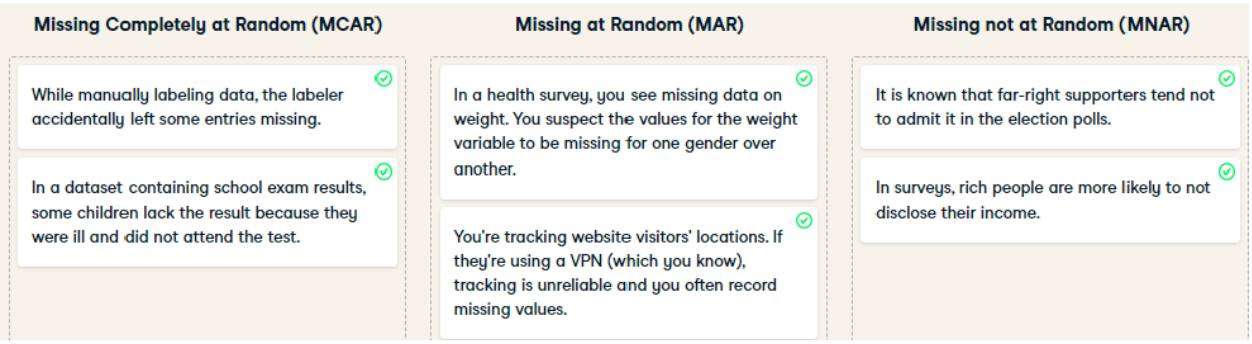


Figure 1: alt text

t-test for MAR: data preparation

Great work on classifying the missing data mechanisms in the last exercise! Of all three, MAR is arguably the most important one to detect, as many imputation methods assume the data are MAR. This exercise will, therefore, focus on testing for MAR.

You will be working with the familiar `biopics` data. The goal is to test whether the number of missing values in `earnings` differs per subject's gender. In this exercise, you will only prepare the data for the t-test. First, you will create a dummy variable indicating missingness in `earnings`. Then, you will split it per gender by first filtering the data to keep one of the genders, and then pulling the dummy variable. For filtering, it might be helpful to print `biopics`'s `head()` in the console and examine the gender variable.

```r
# Create a dummy variable for missing earnings
biopics <- biopics %>%
  mutate(missing_earnings = is.na(earnings))

# Pull the missing earnings dummy for males
missing_earnings_males <- biopics %>%
  filter(sub_sex == "Male") %>%
  pull(missing_earnings)

# Pull the missing earnings dummy for females
missing_earnings_females <- biopics %>%
  filter(sub_sex == "Female") %>%
  pull(missing_earnings)
```

t-test for MAR: interpretation

In the last exercise, you have prepared two vectors with the missing earnings values for each gender: missing_earnings_males and missing_earnings_females. Both of them are available in your workspace. Now you can perform the t-test to check if their means are significantly different from each other! Let's do some serious statistical testing!

```r
# Run the t-test
t.test(missing_earnings_males, missing_earnings_females)
```

```
##
##  Welch Two Sample t-test
##
## data:  missing_earnings_males and missing_earnings_females
## t = 1.1116, df = 294.39, p-value = 0.2672
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.03606549  0.12969214
## sample estimates:
## mean of x mean of y
## 0.4366438 0.3898305
```

Sesión 2

Aggregation plot

The aggregation plot provides the answer to the basic question one may ask about an incomplete dataset: in which combinations of variables the data are missing, and how often? It is very useful for gaining a high-level overview of the missingness patterns. For example, it makes it immediately visible if there is some combination of variables that are often missing together, which might suggest some relation between them.

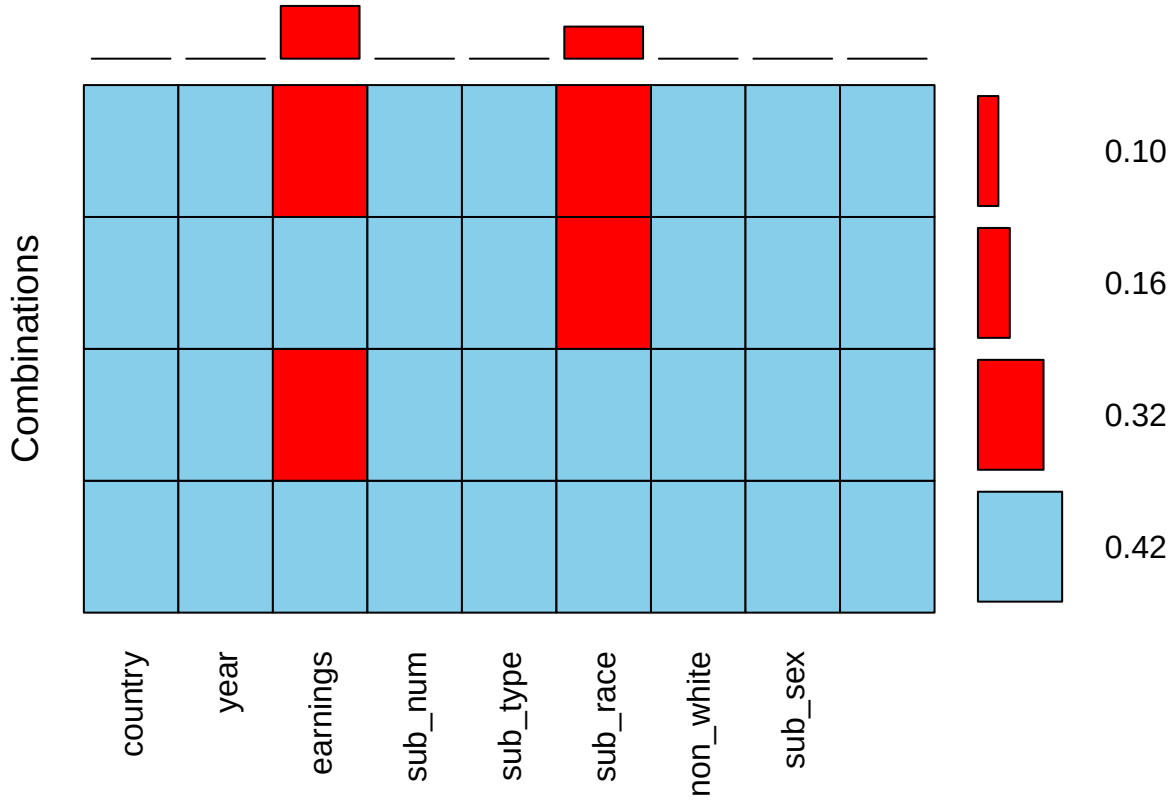
In this exercise, you will first draw the aggregation plot for the `biopics` data and then practice making conclusions based on it. Let's do some plotting!

```r
# Load the VIM package
library(VIM)
```

```
## 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
```

```
# Draw an aggregation plot of biopics
biopics %>%
    aggr(combined = TRUE, numbers = TRUE)
```



Question

Based on the aggregation plot you have just created, which of the following statements is false?

Possible Answers

10% of the observations have missing values in both `earnings` and `sub_race`.

There are more missing values in `sub_race` than in `earnings`.

42% of the observations have no missing entries.

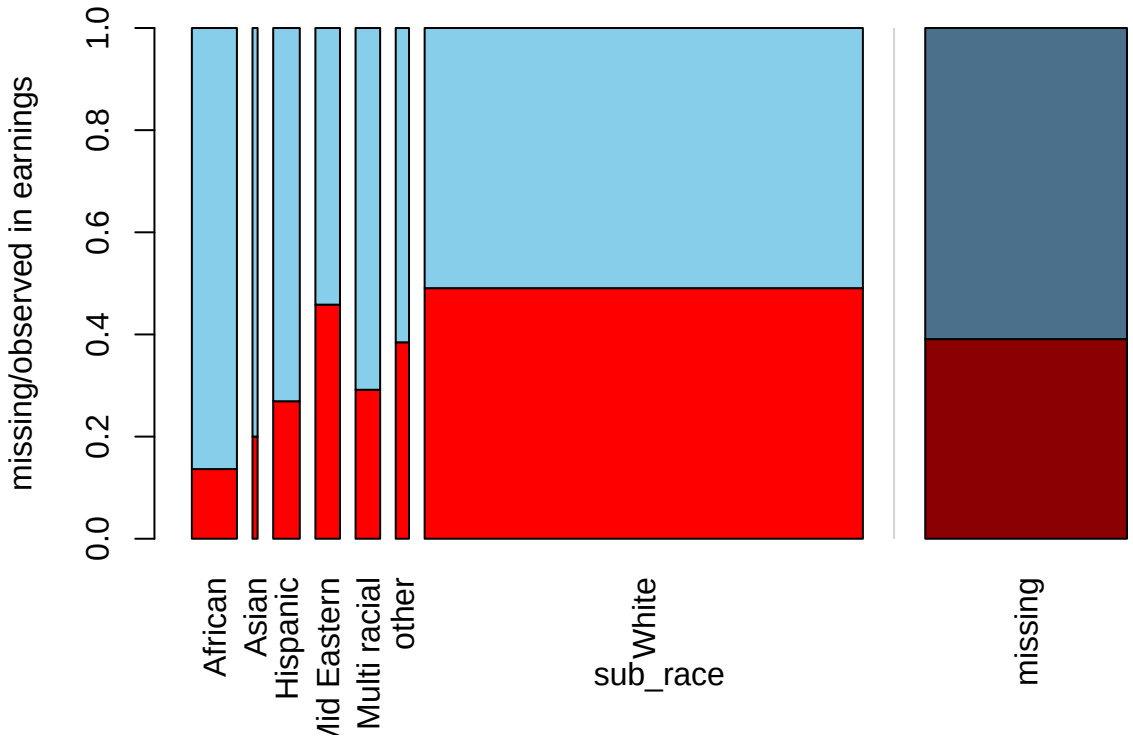
There are exactly two variables in the `biopics` data that have missing values.

Spine plot

The aggregation plot you have drawn in the previous exercise gave you some high-level overview of the missing data. If you are interested in the interaction between specific variables, a spine plot is the way to go. It allows you to study the percentage of missing values in one variable for different values of the other, which is conceptually very similar to the t-tests you have been running in the previous lesson.

In this exercise, you will draw a spine plot to investigate the percentage of missing data in `earnings` for different categories of `sub_race`. Is there more missing data on `earnings` for some specific races of the movie's main character? Let's find out! The `VIM` package has already been loaded for you.

```
# Draw a spine plot to analyse missing values in earnings by sub_race

biopics %>%
    dplyr::select(sub_race,earnings) %>%
    spineMiss()
```



Question

Based on the spine plot you have just created, which of the following statements is false?

Possible Answers

In the vast majority of movies, the main character is white.

When the main subject is African, we are the most likely to have complete earnings information.

As far as earnings and sub_race are concerned, the data seem to be MAR.

The race that appears most rarely in the data has around 40% of earnings missing. (incorrecta)

Mosaic plot

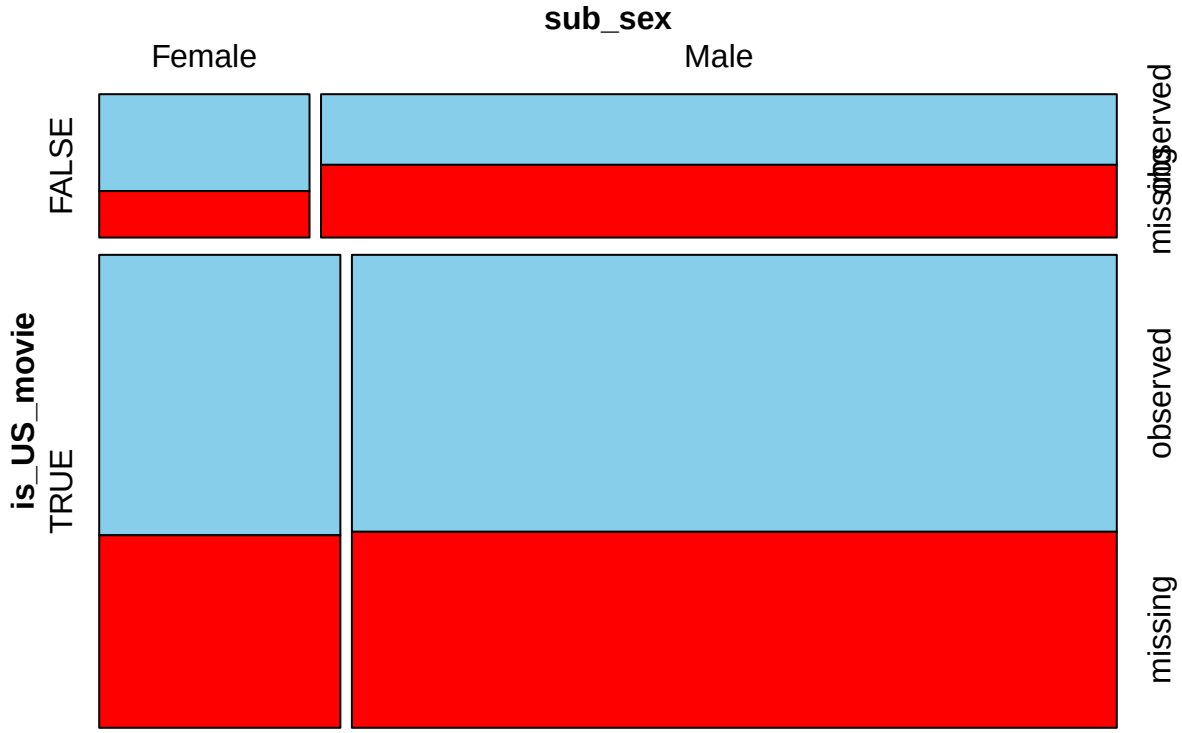
The spine plot you have created in the previous exercise allows you to study missing data patterns between two variables at a time. This idea is generalized to more variables in the form of a mosaic plot.

In this exercise, you will start by creating a dummy variable indicating whether the United States was involved in the production of each movie. To do this, you will use the `grepl()` function, which checks if the string passed as its first argument is present in the object passed as its second argument. Then, you will draw a mosaic plot to see if the subject's gender correlates with the amount of missing data on `earnings` for both US and non-US movies.

The `biopics` data as well as the `VIM` package are already loaded for you. Let's do some exploratory plotting!

Note that a propriety `display_image()` function has been created to return the output from the latest `VIM` package version. Make sure to expand the `HTML Viewer` section.

```
# Prepare data for plotting and draw a mosaic plot
biopics %>%
    # Create a dummy variable for US-produced movies
    mutate(is_US_movie = grepl("US", country)) %>%
    # Draw mosaic plot
    mosaicMiss(highlight = "earnings",
               plotvars = c("is_US_movie", "sub_sex"))
```




```
# Return plot from latest VIM package - expand the HTML viewer section
#display_image()
```

Chapter 2

Sesión 1

Smelling the danger of mean imputation

One of the most popular imputation methods is the mean imputation, in which missing values in a variable are replaced with the mean of the observed values in this variable. However, in many cases this simple approach is a poor choice. Sometimes a quick look at the data can already alert you to the dangers of mean-imputing.

In this chapter, you will be working with a subsample of the Tropical Atmosphere Ocean (`tao`) project data. The dataset consists of atmospheric measurements taken in two different time periods at five different locations. The data comes with the `VIM` package.

In this exercise you will familiarize yourself with the data and perform a simple analysis that will indicate what the consequences of mean imputation could be. Let's take a look at the `tao` data!

```
data(tao, package = "VIM")
names(tao)<-tolower(names(tao))
names(tao)<-sub("[.]", "_", names(tao))
names(tao)<-sub("[.]", "_", names(tao))
```

```
# Print first 10 observations
head(tao, 10)
```

```
##    year latitude longitude sea_surface_temp air_temp humidity uwind vwind
## 1  1997        0      -110            27.59    27.15     79.6  -6.4   5.4
## 2  1997        0      -110            27.55    27.02     75.8  -5.3   5.3
## 3  1997        0      -110            27.57    27.00     76.5  -5.1   4.5
## 4  1997        0      -110            27.62    26.93     76.2  -4.9   2.5
## 5  1997        0      -110            27.65    26.84     76.4  -3.5   4.1
## 6  1997        0      -110            27.83    26.94     76.7  -4.4   1.6
## 7  1997        0      -110            28.01    27.04     76.5  -2.0   3.5
## 8  1997        0      -110            28.04    27.11     78.3  -3.7   4.5
## 9  1997        0      -110            28.02    27.21     78.6  -4.2   5.0
## 10 1997        0      -110            28.05    27.25     76.9  -3.6   3.5
```

```
# Get the number of missing values per column
tao %>%
  is.na() %>%
  colSums()
```

```
##             year         latitude        longitude sea_surface_temp
##                0                0                0                3
##         air_temp         humidity            uwind            vwind
##               81               93                0                0
```

```
# Calculate the number of missing values in air_temp per year
tao %>%
  group_by(year) %>%
  summarize(num_miss = sum(is.na(air_temp)))
```

```
## # A tibble: 2 x 2
##    year num_miss
##   <int>    <int>
## 1  1993        4
## 2  1997       77
```

Mean-imputing the temperature

Mean imputation can be a risky business. If the variable you are mean-imputing is correlated with other variables, this correlation might be destroyed by the imputed values. You saw it looming in the previous exercise when you analyzed the air_temp variable.

To find out whether these concerns are valid, in this exercise you will perform mean imputation on `air_temp`, while also creating a binary indicator for where the values are imputed. It will come in handy in the next exercise, when you will be assessing your imputation's performance. Let's fill in those missing values!

```
tao_imp <- tao %>%
  # Create a binary indicator for missing values in air_temp
  mutate(air_temp_imp = ifelse(is.na(air_temp), TRUE, FALSE)) %>%
  # Impute air_temp with its mean
  mutate(air_temp = ifelse(is.na(air_temp), mean(air_temp, na.rm = TRUE), air_temp))
```

```r
# Print the first 10 rows of tao_imp
head(tao_imp, 10)
```

```
##    year latitude longitude sea_surface_temp air_temp humidity uwind vwind
## 1  1997        0      -110            27.59    27.15     79.6  -6.4   5.4
## 2  1997        0      -110            27.55    27.02     75.8  -5.3   5.3
## 3  1997        0      -110            27.57    27.00     76.5  -5.1   4.5
## 4  1997        0      -110            27.62    26.93     76.2  -4.9   2.5
## 5  1997        0      -110            27.65    26.84     76.4  -3.5   4.1
## 6  1997        0      -110            27.83    26.94     76.7  -4.4   1.6
## 7  1997        0      -110            28.01    27.04     76.5  -2.0   3.5
## 8  1997        0      -110            28.04    27.11     78.3  -3.7   4.5
## 9  1997        0      -110            28.02    27.21     78.6  -4.2   5.0
## 10 1997        0      -110            28.05    27.25     76.9  -3.6   3.5
##    air_temp_imp
## 1         FALSE
## 2         FALSE
## 3         FALSE
## 4         FALSE
## 5         FALSE
## 6         FALSE
## 7         FALSE
## 8         FALSE
## 9         FALSE
## 10        FALSE
```

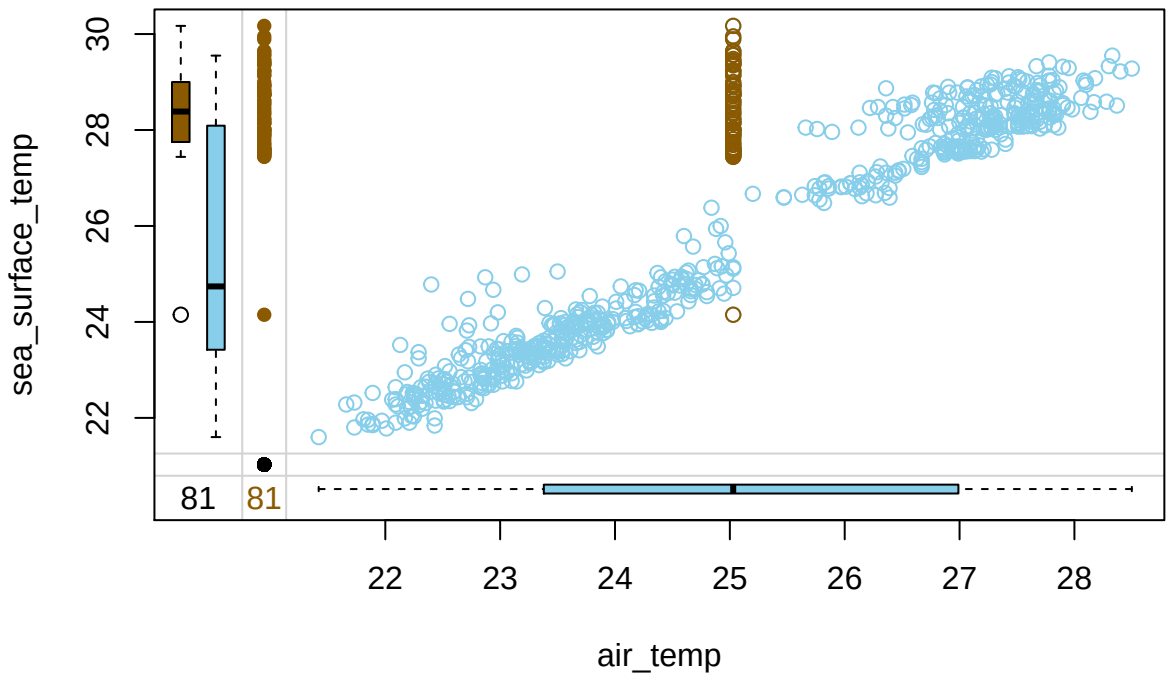
Assessing imputation quality with margin plot

In the last exercise, you have mean-imputed `air_temp` and added an indicator variable to denote which values were imputed, called `air_temp_imp`. Time to see how well this works.

Upon examining the `tao` data, you might have noticed that it also contains a variable called `sea_surface_temp`, which could reasonably be expected to be positively correlated with `air_temp`. If that's the case, you would expect these two temperatures to be both high or both low at the same time. Imputing mean air temperature when the sea temperature is high or low would break this relation.

To find out, in this exercise you will select the two temperature variables and the indicator variable and use them to draw a margin plot. Let's assess the mean imputation!

```r
# Draw a margin plot of air_temp vs sea_surface_temp
tao_imp %>%
  select(air_temp, sea_surface_temp, air_temp_imp) %>%
  marginplot(delimiter = "imp")
```



Sesión 2

Vanilla hot-deck

Hot-deck imputation is a simple method that replaces every missing value in a variable by the last observed value in this variable. It's very fast, as only one pass through the data is needed, but in its simplest form, hot-deck may sometimes break relations between the variables.

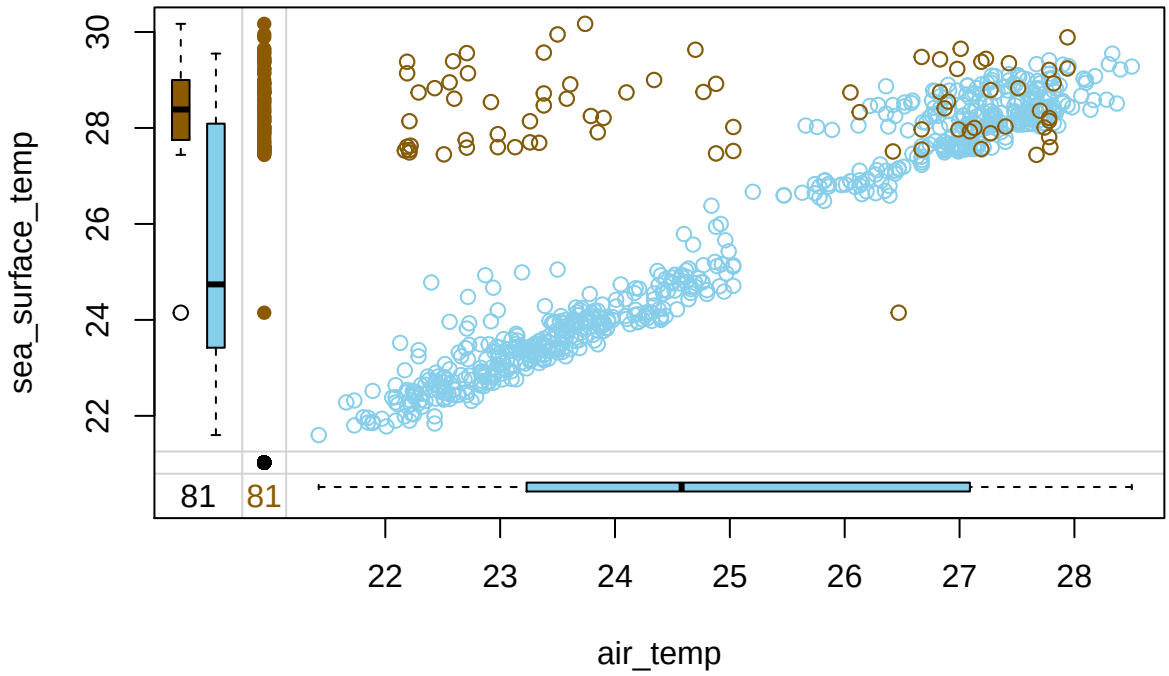
In this exercise, you will try it out on the tao dataset. You will hot-deck-impute missing values in the air temperature column `air_temp` and then draw a margin plot to analyze the relation between the imputed values with the sea surface temperature column `sea_surface_temp`. Let's see how it works!

```
# Load VIM package
library(VIM)

# Impute air_temp in tao with hot-deck imputation
tao_imp <- hotdeck(tao, variable = "air_temp")

# Check the number of missing values in each variable
tao_imp %>%
    is.na() %>%
    colSums()
```

```
##             year         latitude        longitude sea_surface_temp
##                0                0                0                3
```

```
##         air_temp         humidity            uwind            vwind
##                0               93                0                0
##     air_temp_imp
##                0
```

```
# Draw a margin plot of air_temp vs sea_surface_temp
tao_imp %>%
    select(air_temp, sea_surface_temp, air_temp_imp) %>%
    marginplot(delimiter = "imp")
```



Does the imputation look good? Notice the observations in the top left part of the plot with imputed `air_temp` and high `sea_surface_temp`. These observations must have been preceded by ones with low `air_temp` in the data frame, and so after hot-deck imputation, they ended up being outliers with low `air_temp` and high sea_surface_temp.

Hot-deck tricks & tips I: imputing within domains

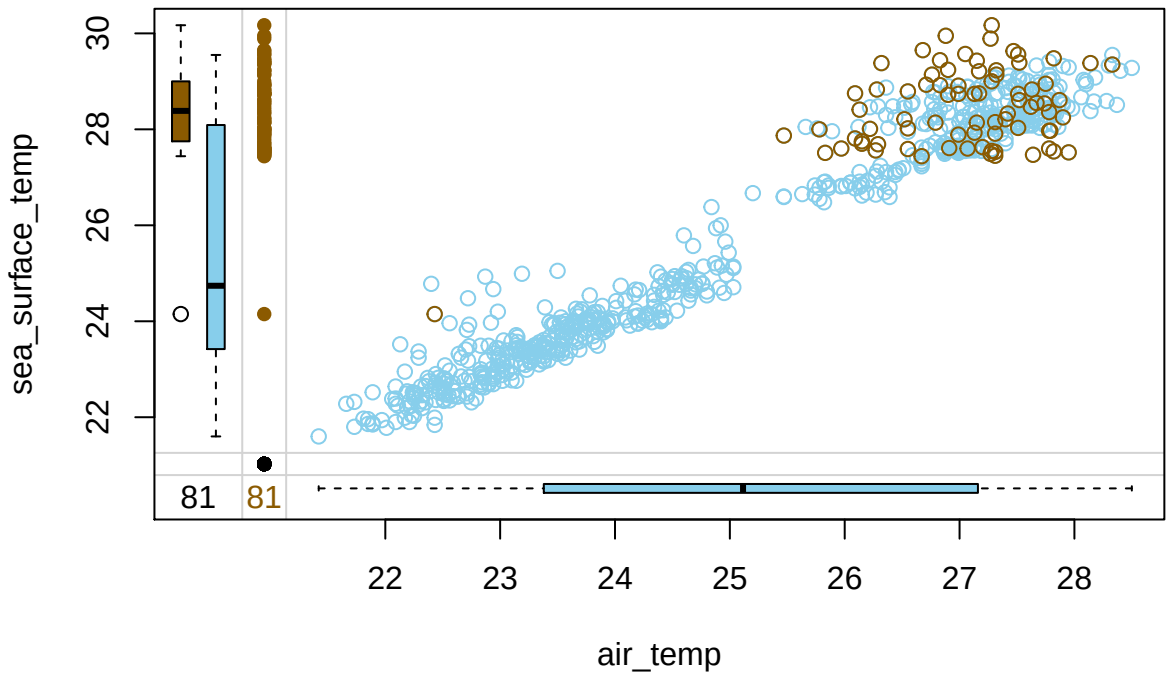
One trick that may help when hot-deck imputation breaks the relations between the variables is imputing within domains. What this means is that if the variable to be imputed is correlated with another, categorical variable, one can simply run hot-deck separately for each of its categories.

For instance, you might expect air temperature to depend on time, as we are seeing the average temperatures rising due to global warming. The time indicator you have available in the tao data is a categorical variable, year. Let's first check if the average air temperature is different in each of the two studied years and then run hot-deck within year domains. Finally, you will draw the margin plot again to assess the imputation performance.

```r
# Calculate mean air_temp per year
tao %>%
    group_by(year) %>%
    summarize(average_air_temp = mean(air_temp, na.rm = TRUE))
```

```
## # A tibble: 2 x 2
##    year average_air_temp
##   <int>            <dbl>
## 1  1993             23.4
## 2  1997             27.1
```

```r
# Hot-deck-impute air_temp in tao by year domain
tao_imp <- hotdeck(tao, variable = "air_temp", domain_var = "year")

# Draw a margin plot of air_temp vs sea_surface_temp
tao_imp %>%
    select(air_temp, sea_surface_temp, air_temp_imp) %>%
    marginplot(delimiter = "imp")
```



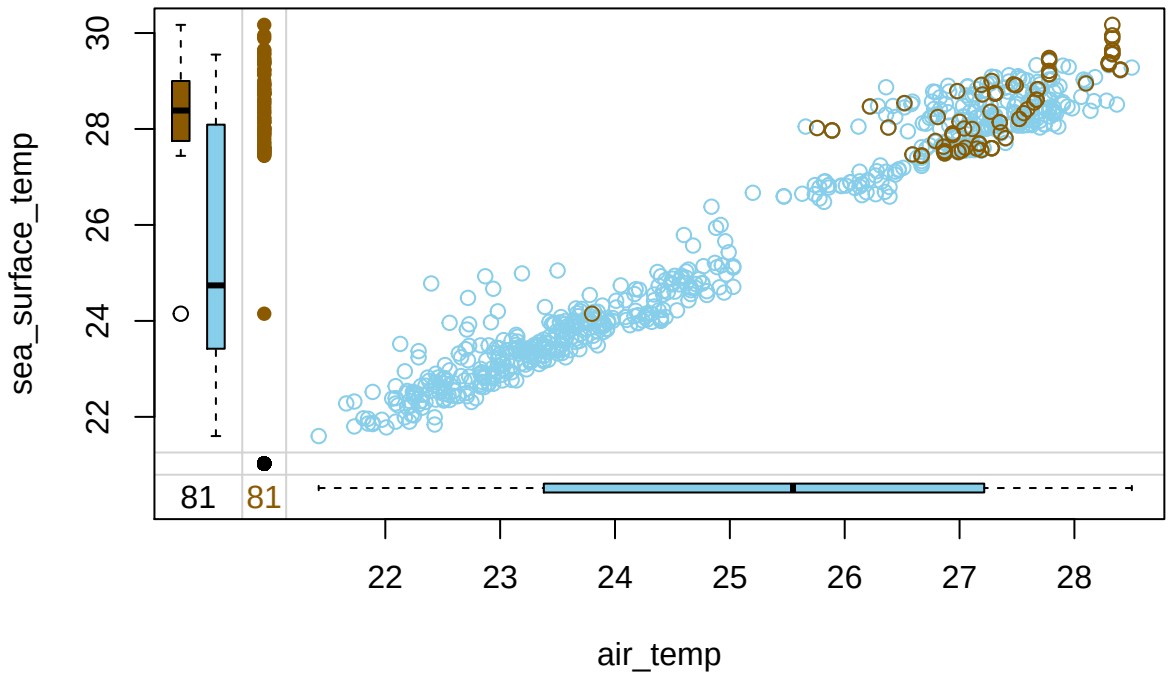
The results look much better this time. However, if you look at the top right corner of the plot, you will see that the variance in the imputed (orange) values is somewhat larger than among the observed (blue) values. Let's see if we can improve even further in the next exercise!

Hot-deck tricks & tips II: sorting by correlated variables

Another trick that can boost the performance of hot-deck imputation is sorting the data by variables correlated to the one we want to impute.

For instance, in all the margin plots you have been drawing recently, you have seen that air temperature is strongly correlated with sea surface temperature, which makes a lot of sense. You can exploit this knowledge to improve your hot-deck imputation. If you first order the data by sea_surface_temp, then every imputed air_temp value will come from a donor with a similar sea_surface_temp. Let's see how this will work!

```
# Hot-deck-impute air_temp in tao ordering by sea_surface_temp
tao_imp <- hotdeck(tao, variable = "air_temp", ord_var = "sea_surface_temp")

# Draw a margin plot of air_temp vs sea_surface_temp
tao_imp %>%
    select(air_temp, sea_surface_temp, air_temp_imp) %>%
    marginplot(delimiter = "imp")
```



This time the imputation seems not to impact the relation between air and sea temperatures: if not for the colors, you likely wouldn't know which ones are the imputed values. Hot-deck imputation, possibly enhanced with domain-imputing or sorting, is a fast and simple method that can serve you well in many situations. However, sometimes you may need a more complex approach.

Sesión 3

Choosing the number of neighbors

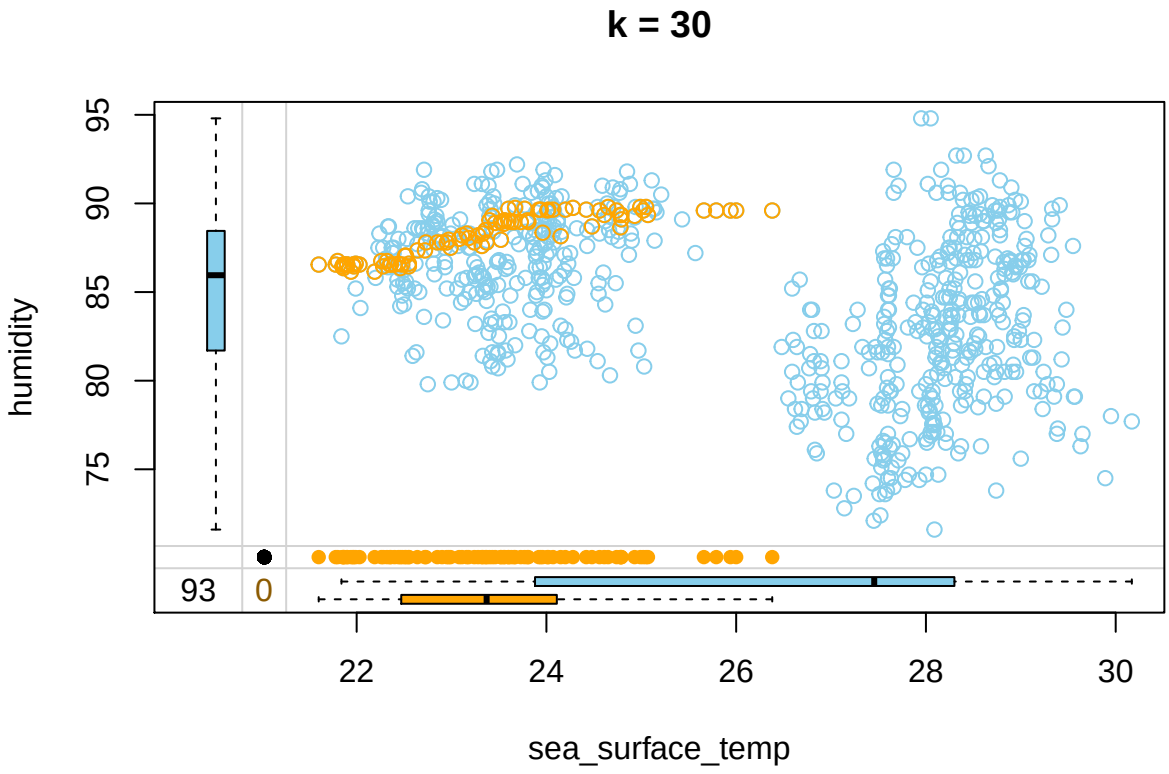
k-Nearest-Neighbors (or kNN) imputation fills the missing values in an observation based on the values coming from the k other observations that are most similar to it. The number of these similar observations, called neighbors, that are considered is a parameter that has to be chosen beforehand.

How to choose k? One way is to try different values and see how they impact the relations between the imputed and observed data.

Let's try imputing `humidity` in the `tao` data using three different values of k and see how the imputed values fit the relation between `humidity` and `sea_surface_temp`.

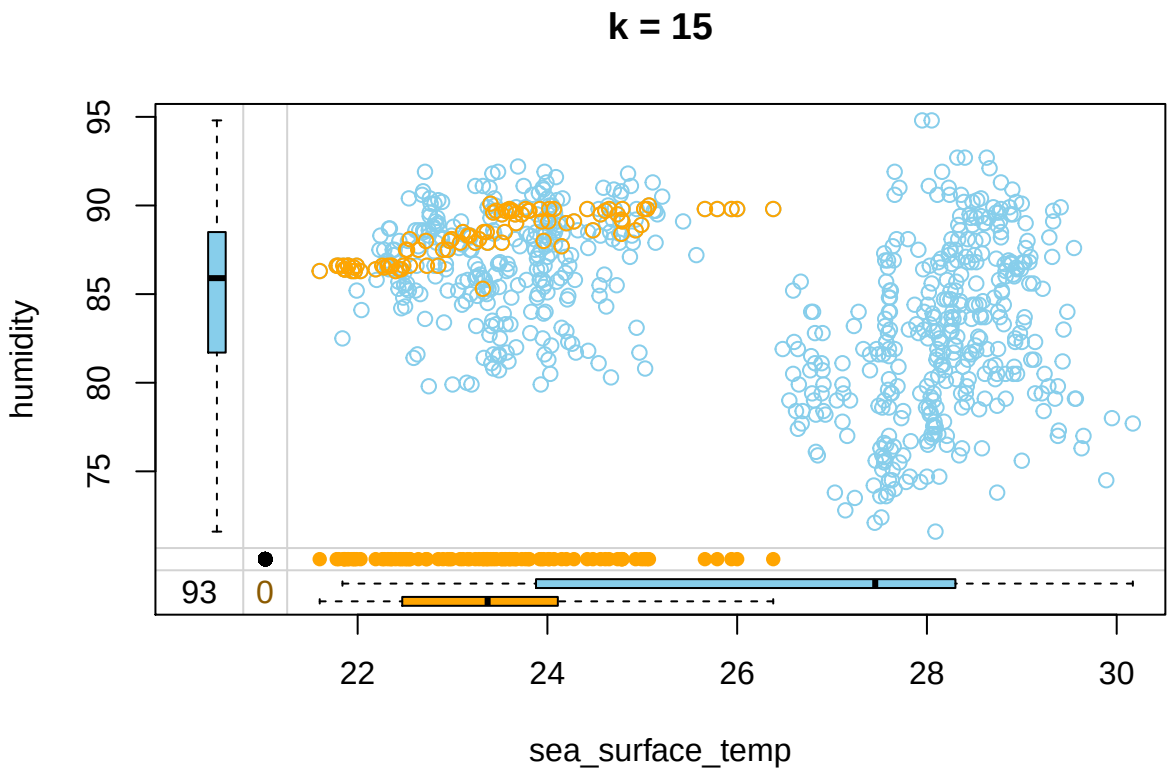
Impute `humidity` with kNN imputation using 30 neighbors and draw a `marginplot()` of `sea_surface_temp` vs `humidity`.

```
# Impute humidity using 30 neighbors
tao_imp <- kNN(tao, k = 30, variable = "humidity")

# Draw a margin plot of sea_surface_temp vs humidity
tao_imp %>%
    select(sea_surface_temp, humidity, humidity_imp) %>%
    marginplot(delimiter = "imp", main = "k = 30")
```



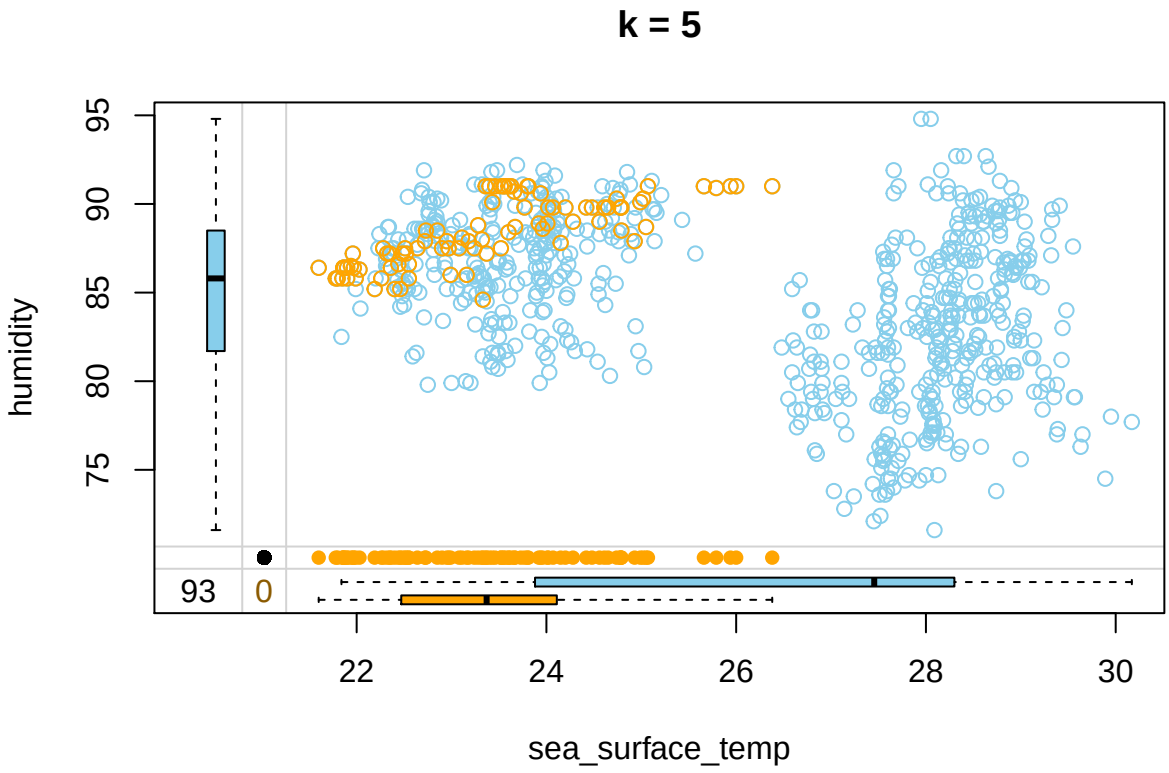
Impute `humidity` with kNN imputation using 15 neighbors and draw a `marginplot` of `sea_surface_temp` vs `humidity`.

```
# Impute humidity using 15 neighbors
tao_imp <- kNN(tao, k = 15, variable = "humidity")

# Draw a margin plot of sea_surface_temp vs humidity
tao_imp %>%
    select(sea_surface_temp, humidity, humidity_imp) %>%
    marginplot(delimiter = "imp", main = "k = 15")
```



Impute `humidity` with kNN imputation using 5 neighbors and draw a `marginplot` of `sea_surface_temp` vs `humidity`.

```
# Impute humidity using 5 neighbors
tao_imp <- kNN(tao, k = 5, variable = "humidity")

# Draw a margin plot of sea_surface_temp vs humidity
tao_imp %>%
    select(sea_surface_temp, humidity, humidity_imp) %>%
    marginplot(delimiter = "imp", main = "k = 5")
```

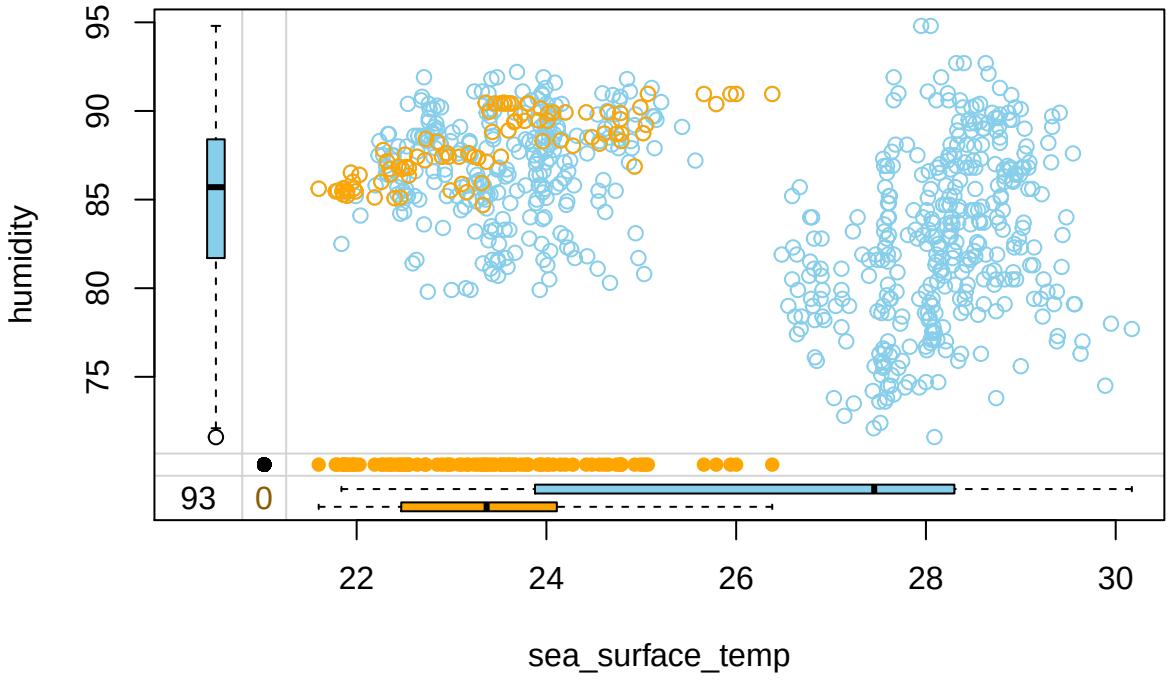


kNN tricks & tips I: weighting donors

A variation of `kNN` imputation that is frequently applied uses the so-called distance-weighted aggregation. What this means is that when we aggregate the values from the neighbors to obtain a replacement for a missing value, we do so using the weighted mean and the weights are inverted distances from each neighbor. As a result, closer neighbors have more impact on the imputed value.

In this exercise, you will apply the distance-weighted aggregation while imputing the `tao` data. This will only require passing two additional arguments to the `kNN()` function. Let's try it out!

```
# Load the VIM package
library(VIM)

# Impute humidity with kNN using distance-weighted mean
tao_imp <- kNN(tao,
               k = 5,
               variable = "humidity",
               numFun = weighted.mean,
               weightDist = TRUE)

tao_imp %>%
    select(sea_surface_temp, humidity, humidity_imp) %>%
    marginplot(delimiter = "imp")
```



kNN tricks & tips II: sorting variables

As the k-Nearest Neighbors algorithm loops over the variables in the data to impute them, it computes distances between observations using other variables, some of which have already been imputed in the previous steps. This means that if the variables located earlier in the data have a lot of missing values, then the subsequent distance calculation is based on a lot of imputed values. This introduces noise to the distance calculation.

For this reason, it is a good practice to sort the variables increasingly by the number of missing values before performing kNN imputation. This way, each distance calculation is based on as much observed data and as little imputed data as possible.

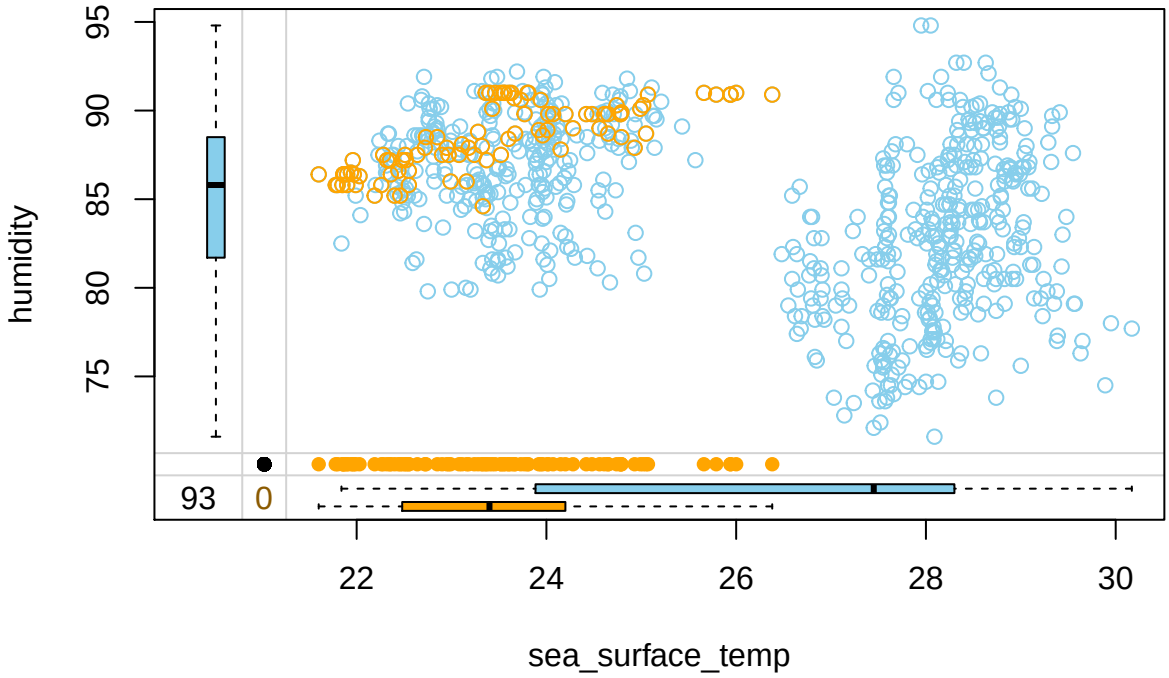
Let's try this out on the `tao` data!

```
# Get tao variable names sorted by number of NAs
vars_by_NAs <- tao %>%
  is.na() %>%
  colSums() %>%
  sort(decreasing = FALSE) %>%
  names()

# Sort tao variables and feed it to kNN imputation
tao_imp <- tao %>%
  select(vars_by_NAs) %>%
  kNN(k= 5)
```

```
## Note: Using an external vector in selections is ambiguous.
## i Use ‘all_of(vars_by_NAs)‘ instead of ‘vars_by_NAs‘ to silence this message.
```

```
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.
```

```
tao_imp %>%
    select(sea_surface_temp, humidity, humidity_imp) %>%
    marginplot(delimiter = "imp")
```



The kNN you have just coded should be more accurate and robust against faulty imputations, so remember to sort your variables first before performing kNN imputation! This brings us to the end of this chapter. Keep it up! See you in Chapter 3, where you will learn to use statistical and machine learning models to impute missing values!

Chapter 3

Sesión 1

Linear regression imputation

Sometimes, you can use domain knowledge, previous research or simply your common sense to describe the relations between the variables in your data. In such cases, model-based imputation is a great solution, as it allows you to impute each variable according to a statistical model that you can specify yourself, taking into account any assumptions you might have about how the variables impact each other.

For continuous variables, a popular model choice is linear regression. It doesn't restrict you to linear relations though! You can always include a square or a logarithm of a variable in the predictors. In this exercise, you

will work with the `simputation` package to run a single linear regression imputation on the `tao` data and analyze the results. Let's give it a try!

```
# Load the simputation package
library(simputation)

# Impute air_temp and humidity with linear regression
formula <- air_temp + humidity ~ year + latitude + sea_surface_temp
tao_imp <- impute_lm(tao, formula)
```

```
# Load the simputation package
library(simputation)

# Impute air_temp and humidity with linear regression
formula <- air_temp + humidity ~ year + latitude + sea_surface_temp
tao_imp <- impute_lm(tao, formula)

# Check the number of missing values per column
tao_imp %>%
  is.na() %>%
  colSums()
```

```
##             year         latitude        longitude sea_surface_temp
##                0                0                0                3
##         air_temp         humidity            uwind            vwind
##                3                2                0                0
```

```
# Print rows of tao_imp in which air_temp or humidity are still missing
tao_imp %>%
  filter(is.na(air_temp) | is.na(humidity))
```

```
##   year latitude longitude sea_surface_temp air_temp humidity uwind vwind
## 1 1993        0       -95               NA       NA       NA  -5.6   3.1
## 2 1993        0       -95               NA       NA       NA  -6.3   0.5
## 3 1993       -2       -95               NA       NA     89.9  -3.4   2.4
```

Linear regression fails when at least one of the predictors is missing. In this case, it was sea_surface_temp. In the next exercise, you will fix it by initializing the missing values before running `impute_lm()`!

Initializing missing values & iterating over variables

As you have just seen, running `impute_lm()` might not fill-in all the missing values. To ensure you impute all of them, you should initialize the missing values with a simple method, such as the hot-deck imputation you learned about in the previous chapter, which simply feeds forward the last observed value.

Moreover, a single imputation is usually not enough. It is based on the basic initialized values and could be biased. A proper approach is to iterate over the variables, imputing them one at a time in the locations where they were originally missing.

In this exercise, you will first initialize the missing values with hot-deck imputation and then loop five times over `air_temp` and `humidity` from the `tao` data to impute them with linear regression. Let's get to it!

```r
# Initialize missing values with hot-deck
tao_imp <- hotdeck(tao)

# Create boolean masks for where air_temp and humidity are missing
missing_air_temp <- tao_imp$air_temp_imp
missing_humidity <- tao_imp$humidity_imp

for (i in 1:5) {
  # Set air_temp to NA in places where it was originally missing and re-impute it
  tao_imp$air_temp[missing_air_temp] <- NA
  tao_imp <- impute_lm(tao_imp, air_temp ~ year + latitude + sea_surface_temp + humidity)
  # Set humidity to NA in places where it was originally missing and re-impute it
  tao_imp$humidity[missing_humidity] <- NA
  tao_imp <- impute_lm(tao_imp, humidity ~ year + latitude + sea_surface_temp + air_temp)
}
```

That's a professional approach to model-based imputation you have just coded! But how do we know that 5 is the proper number of iterations to run? Let's look at the convergence in the next exercise!

Detecting convergence

Great job iterating over the variables in the last exercise! But how many iterations are needed? When the imputed values don't change with the new iteration, we can stop.

You will now extend your code to compute the differences between the imputed variables in subsequent iterations. To do this, you will use the Mean Absolute Percentage Change function, defined for you as follows:

```r
mapc <- function(a, b) {   mean(abs(b - a) / a, na.rm = TRUE) }
```

`mapc()` outputs a single number that tells you how much b differs from a. You will use it to check how much the imputed variables change across iterations. Based on this, you will decide how many of them are needed!

The boolean masks `missing_air_temp` and `missing_humidity` are available for you, as is the hotdeck-initialized `tao_imp` data.

```r
mapc<- function(a, b) {
  mean(abs(b - a) / a, na.rm = TRUE)
}
```

```r
diff_air_temp <- c()
diff_humidity <- c()

for (i in 1:5) {
  # Assign the outcome of the previous iteration (or initialization) to prev_iter
  prev_iter <- tao_imp
  # Impute air_temp and humidity at originally missing locations
  tao_imp$air_temp[missing_air_temp] <- NA
  tao_imp <- impute_lm(tao_imp, air_temp ~ year + latitude + sea_surface_temp + humidity)
  tao_imp$humidity[missing_humidity] <- NA
  tao_imp <- impute_lm(tao_imp, humidity ~ year + latitude + sea_surface_temp + air_temp)
  # Calculate MAPC for air_temp and humidity and append them to previous iteration's MAPCs
  diff_air_temp <- c(diff_air_temp, mapc(prev_iter$air_temp, tao_imp$air_temp))
  diff_humidity <- c(diff_humidity, mapc(prev_iter$humidity, tao_imp$humidity))
}
```

Question

Based on the differences stored in `diff_air_temp` and `diff_humidity`, what is the sufficient number of iterations to run?

To answer this question, you can print the two vectors in the console and analyze the numbers, or plot them using the function provided for you: just run `plot_diffs(diff_air_temp, diff_humidity)` in the console.

```
plot_diffs <- function(a, b) {
  data.frame("mapc" = c(a, b),
             "Variable" = c(rep("air_temp", length(a)),
                            rep("humidity", length(b))),
             "Iterations" = c(1:length(a), 1:length(b))) %>%
    ggplot(aes(Iterations, mapc, color = Variable)) +
    geom_line(size = 1.5) +
    ylab("Mean absolute percentage change") +
    ggtitle("Changes in imputed variables' values across iterations") +
    theme(legend.position = "bottom")
}
```

```
plot_diffs(diff_air_temp, diff_humidity)
```