Deeper examination of some functions of the `biometrics package`

`Examples`

18 de noviembre de 2025

This document expands on examples of different functions from the biometrics package (Salas-Eljatib et al., 2025), implemented in R (R Core Team, 2025), for internal use.

Índice

1. treestat

About this function

The `treestat` function aims to calculate a series of descriptive statistics at the tree level per sample unit (i.e., plot). A user can compute these statistics for one or more random variables, such as tree height or diameter, specified in the `y` option. In addition, the calculations can be segregated by a specified factor or categorical variabl, like tree species using, the `factvar` option. It is important to note that the statistics to be calculated are those widely used to describe the distribution of random variables, which are available in the `descstat()` function in the datana package (Salas-Eljatib & Campos, 2025).

Data for the examples

The data from a list of trees, i.e., the "tree listzou may already be familiar with, is available in the `eucaplot2` dataframe of the `biometrics` package and will be used for our examples.

```
# Dataframe to be used
df <- biometrics::eucaplot
head(df)
```

```
dbh health shape crown.class toth
1 28.0      1     1           2   27
2 35.5      1     1           1   31
3 34.0      1     2           1   31
4 37.5      1     2           1   35
5 26.5      1     2           2   27
6 29.0      1     1           2   26
```

```
# Some checking
datana::descstat(df[, c("dbh","toth")])
df$plot <- 1
```

```
dbh   toth
n         15.000 15.000
Minimum   24.000 26.000
Maximum   37.500 35.000
Mean      30.493 29.667
Median    29.500 30.000
Std. Dev.  3.800  3.039
CV(%)     12.463 10.245
```

Below are examples of the potential uses of the function in question.

1.1. For a single random variable

The total height, which is stored in the `toth` column, is used as the random variable of interest. A first example would then be

```
## Using the function
treestat(data = df, plot.id = "plot", y = "toth")
```

```
plot    t n.toth min.toth max.toth mean.toth median.toth sd.toth cv.toth
1    1 2025     15       26       35    29.667          30   3.039  10.245
```

1.2. For two random variables

If more than one random variable is needed, they are added in the y option, then define a vector with the names of the two variables. For instance, if we want to consider both the tree diameter and total height, we proceed as follows

```
# Now, for two random variables, instead of a single one
treestat(data = df, plot.id = "plot", y = c("dbh", "toth"))
```

```
plot    t n.dbh min.dbh max.dbh mean.dbh median.dbh sd.dbh cv.dbh n.toth min.toth
1    1 2025    15      24    37.5   30.493       29.5    3.8 12.463     15       26
  max.toth mean.toth median.toth sd.toth cv.toth
1       35    29.667          30   3.039  10.245
```

1.3. A complete descriptive statistics

As indicated above, the statistics to be calculated are those defined in the `descstat()` function of the datana package. However, the options of the `descstat()` function can be passed to the current `treestat()` function by specifying the respective option from the former. For example, the logical option `full` of the `descstat()` function, which is `FALSE` by default, changes when the opposite is specified.

```
## la opcion full de la funcion descstat()
datana::descstat(df[, c("dbh","toth")], full=TRUE)
```

```
dbh   toth
n               15.000 15.000
Minimum         24.000 26.000
Maximum         37.500 35.000
Mean            30.493 29.667
Median          29.500 30.000
Std. Dev.        3.800  3.039
CV(%)           12.463 10.245
25th Percentile 28.000 27.000
75th Percentile 33.000 31.000
Interq. range    5.000  4.000
Skewness         0.274  0.410
Kurtosis        -1.054 -1.192
```

An example of the above is specifying the `full` option of the `descstat()` function when applying the `treestat()` function to a random variable. For instance, for the variable `toth`, this is achieved by

```
## Using the function con opcion full
treestat(data = df, plot.id = "plot", y = "toth", full=TRUE)
```

```
plot    t n.toth min.toth max.toth mean.toth median.toth sd.toth cv.toth 25p.toth
1    1 2025     15       26       35    29.667          30   3.039  10.245       27
  75p.toth iqr.toth ske.toth kurt.toth
1       31        4     0.41    -1.192
```

Note the increase in the number of columns in the output, which represents a greater number of statistics calculated, compared to what was shown in §1.1.

1.4. About the use of the variable tree diameter

The diameter at breast height (d) is a variable measured on all trees within a sample unit. This variable can be added to the list of variables to which the function is applied by specifying the column name with the variable d in option `d` and setting the `want.add.d` option to `TRUE`, as shown below.

```
treestat(data = df, plot.id = "plot", y = "toth", d = "dbh", want.add.d = TRUE)
```

```
Message: The dbh was specified to be the column named: "dbh", which will be used accordantly.
  plot    t n.toth min.toth max.toth mean.toth median.toth sd.toth cv.toth n.dbh min.dbh
1    1 2025     15       26       35    29.667          30   3.039  10.245    15      24
  max.dbh mean.dbh median.dbh sd.dbh cv.dbh
1    37.5   30.493       29.5    3.8 12.463
```

Note the following:

- This output is equivalent to that given in §1.2, except that now the variable "dbhïs added at the end of the list of variables.
- The output displays a message indicating which data column was specified as the diameter at breast height.

1.5. Adding the tree level variable basal area

The basal area of a tree (g) is a variable calculated from the diameter d. Therefore, once the diameter is known, it is possible to calculate g. If you want to add the variable g to the list, you must specify that the `want.add.g` option is `TRUE`, as follows

```
treestat(data = df, plot.id = "plot", y = "toth", d = "dbh", want.add.g = TRUE)
```

```
  plot    t n.toth min.toth max.toth mean.toth median.toth sd.toth cv.toth n.g min.g
```

```
1    1 2025     15       26       35    29.667          30   3.039  10.245  15 0.045
  max.g mean.g median.g  sd.g  cv.g
1  0.11  0.074    0.068 0.019 25.07
```

Then the statistics are first calculated for the variable (or vector of variables) specified in the option `y`, to which the variable g is added (at the end).

It is worth mentioning that:

- The basal area can only be calculated in this function if the column containing

the diameter at breast height has been declared (in option `d`).

- By default, the language for messages and the output of the descstat() function is English.

However, if you want the results in Spanish, specify that the `eng` option is `FALSE`.

```
treestat(data = df, plot.id = "plot", y = "toth", d = "dbh", want.add.g = TRUE, eng = FALSE)
```

```
  plot    t n.toth min.toth max.toth mean.toth median.toth sd.toth cv.toth n.ab min.ab
1    1 2025     15       26       35    29.667          30   3.039  10.245   15  0.045
  max.ab mean.ab median.ab sd.ab cv.ab
1   0.11   0.074     0.068 0.019 25.07
```

1.6. Adding both tree diameter and basal area

Although the diameter variable is known, which was specified in option `d`, this variable was not added to the calculations in §1.5, since by default the option `want.add.d` is `FALSE`. Given the above, if the column for the diameter at chest height is specified, it can be added both to that variable and to the basal area, as two new variables (which are added to the object `y`) to the function in question, as follows

```
treestat(data = df, plot.id = "plot", y = "toth", d = "dbh",
         want.add.d = TRUE, want.add.g = TRUE)
```

```
Message: The dbh was specified to be the column named: "dbh", which will be used accordantly.
  plot    t n.toth min.toth max.toth mean.toth median.toth sd.toth cv.toth n.dbh min.dbh
1    1 2025     15       26       35    29.667          30   3.039  10.245    15      24
  max.dbh mean.dbh median.dbh sd.dbh cv.dbh n.g min.g max.g mean.g median.g  sd.g  cv.g
1    37.5   30.493       29.5    3.8 12.463  15 0.045  0.11  0.074    0.068 0.019 25.07
```

1.7. Segregating by a factor

All of the above can be performed segregated by a factor or categorical variable. To illustrate this, we will use the factor "crown class" (column `crown.class`), for a random variable, as in §1.1, as show in the following:

```r
# Do the same as before, but adding the computation by a factor
treestat(data = df, plot.id = "plot", y = "toth", factvar = "crown.class")
```

```
plot    t      factor level n.toth min.toth max.toth mean.toth median.toth sd.toth
2    1 2025 crown.class     1      8       30       35    32.000          31   2.070
3    1 2025 crown.class     2      7       26       29    27.000          27   1.000
1    1 2025        None   All     15       26       35    29.667          30   3.039
  cv.toth
2   6.469
3   3.704
1  10.245
```

In the previous output, note the following:

- The `factor` column contains the selected categorical variable, as a way to verify that the function is appropriately computing what the user wanted.
- The `level` column, contains each category of the factor being used.
- The row identified as **None** for the column `factor`, corresponds to the result of applying the function without considering a factor to segregate the computation. Therefore, these results are equivalent to those reported in §1.1.

1.8. Trees-list measured over time

Now, assume that the sample unit of the data used has been measured twice over time, both in **2020** and in **2025**.

```r
# Creando un set de datos con remediciones (ficticio)
df$time <- 2025
df$time[1:5] <- 2020
df
table(df$time)
```

```
dbh health shape crown.class toth plot time
1  28.0      1     1           2   27    1 2020
2  35.5      1     1           1   31    1 2020
3  34.0      1     2           1   31    1 2020
4  37.5      1     2           1   35    1 2020
5  26.5      1     2           2   27    1 2020
6  29.0      1     1           2   26    1 2025
7  24.0      1     1           2   27    1 2025
8  30.0      1     1           1   30    1 2025
9  29.5      1     2           2   27    1 2025
10 32.0      1     2           1   33    1 2025
```

```
11 29.4      1     1           2   26    1 2025
12 28.0      1     2           2   29    1 2025
13 31.5      1     2           1   35    1 2025
14 27.0      1     2           1   31    1 2025
15 35.5      2     2           1   30    1 2025

2020 2025
   5   10
```

Therefore, the computations to be carried out must consider not only the plot code (`plot` column) but also the year of measurement, which is stored in the `time` column.

```
## Using the function per measurement year
treestat(data = df, plot.id = "plot", y = "toth", t = "time")
```

```
plot time n.toth min.toth max.toth mean.toth median.toth sd.toth cv.toth
1    1 2020      5       27       35      30.2        31.0   3.347  11.082
2    1 2025     10       26       35      29.4        29.5   3.026  10.292
```

1.9. Segregated by a factor

As shown in §1.7, the function `treestat()` can be applied to trees-list measured over time. Nonetheless, the function can also handle, to carry the computation segregated by the levels of a factor. This implies to apply the function for each measurement-year and each level of a factor. Here we show how to achieve the above by measurement year (column `time`) and levels of the factor `crown.class`,

```
## Using the function per measurement year
treestat(data = df, plot.id = "plot", y = "toth", t = "time", factvar = "crown.class")
```

```
plot time      factor level n.toth min.toth max.toth mean.toth median.toth sd.toth
3    1 2020 crown.class     1      3       31       35    32.333        31.0   2.309
5    1 2020 crown.class     2      2       27       27    27.000        27.0   0.000
1    1 2020        None   All      5       27       35    30.200        31.0   3.347
4    1 2025 crown.class     1      5       30       35    31.800        31.0   2.168
6    1 2025 crown.class     2      5       26       29    27.000        27.0   1.225
2    1 2025        None   All     10       26       35    29.400        29.5   3.026
  cv.toth
3   7.142
5   0.000
1  11.082
4   6.817
6   4.536
2  10.292
```

1.10. Segregated by levels of one factor and several variables

As seen in §1.2, it is now also possible to consider adding more random variables to the above, as follows

```
# Do the same as before, but adding the computation by a factor
treestat(data = df, plot.id = "plot", y = c("dbh","toth"), t = "time",
         factvar = "crown.class", d = "dbh", want.add.g = TRUE)
```

```
plot time      factor level n.dbh min.dbh max.dbh mean.dbh median.dbh sd.dbh cv.dbh
3    1 2020 crown.class     1     3    34.0    37.5   35.667       35.50  1.756  4.923
4    1 2020 crown.class     2     2    26.5    28.0   27.250       27.25  1.061  3.892
1    1 2020        None   All     5    26.5    37.5   32.300       34.00  4.804 14.872
5    1 2025 crown.class     1     5    27.0    35.5   31.200       31.50  3.094  9.918
6    1 2025 crown.class     2     5    24.0    29.5   27.980       29.00  2.303  8.229
2    1 2025        None   All    10    24.0    35.5   29.590       29.45  3.081 10.412
  n.toth min.toth max.toth mean.toth median.toth sd.toth cv.toth n.g min.g max.g mean.g
3      3       31       35    32.333        31.0   2.309   7.142   3 0.091 0.110  0.100
4      2       27       27    27.000        27.0   0.000   0.000   2 0.055 0.062  0.058
1      5       27       35    30.200        31.0   3.347  11.082   5 0.055 0.110  0.083
5      5       30       35    31.800        31.0   2.168   6.817   5 0.057 0.099  0.077
6      5       26       29    27.000        27.0   1.225   4.536   5 0.045 0.068  0.062
2     10       26       35    29.400        29.5   3.026  10.292  10 0.045 0.099  0.069
  median.g  sd.g   cv.g
3    0.099 0.010  9.866
4    0.058 0.005  7.779
1    0.091 0.024 28.774
5    0.078 0.015 19.739
6    0.066 0.010 15.607
2    0.068 0.014 20.803
```

2. bankfit

About this function

The `bankfit()` function fits a list of models to a dataset, therefore, It is expected that this list will include more than one model. Although the bankfit function is part of the biometrics package (Salas-Eljatib et al., 2025), it requires functions from the datana package (Salas-Eljatib & Campos, 2025), so we need to install the later package as well. As expected, the user will need to provide the following main inputs: **list of functions** and **data**, which are explained in detail in the following subsections.

A. Models list

The list of models must have the following structure:

1. `expr` represents the relationship between the response variable y and the predictor variable x.
2. `pred.f` represents a function to compute the predicted value of the variable y based on the expresion (`expr`) of the fitted model. This only important for model on which the response variable of the statistical model implies a function of the response variable of interest, or as we call it "the biometrics response variable".
3. `type` indicates the fitting approach to be used for the model. The options here are: "lm.ºr "nls".
4. `params` provides the initial parameter values when the `type` is "nls".

```
my.modlist <- list(
  ## a linear model to be fitted using the lm() function
  mod1 = list(expr = log(y)~x,
              pred.f = function(y) exp(y),
              type = "lm"),
  ## a non-linear model to be fitted using the nls() function
  mod2 = list(expr = y~taperpoly(x, n = 2),
              params = list(b0 = 1, b1 = 0, b2 = 0.5),
              pred.f = function(y) sqrt(y),
              type = "nls")
)
```

B. Data

The data, as a dataframe object, must contain the variables as columns, specified in the models list object `mod.list` to be used for applying the `bankfit()` function. For instance, if we follow the example of the object with the models list `my.modlist` that we already created above, the data must have (at least) two columns, and they are `y` and `x`. If any of those columns are available, the `bankfit()` function will not fit the corresponding model that uses those variables.

Data for examples

We use tree-level data for 382 sample trees where volume and other dendrometric variables had been measured. The data are stored in the `treevolruca` dataframe of the `biometrics` package.

```
library(biometrics)
library(datana)
# Dataframe to be used
df <- biometrics::treevolruca
head(df)
```

```
tree      spp  dbh toth d6   totv
1    1    Roble 98.0 50.0 75 11.954
2    2 Olivillo 28.5 33.5 21  0.755
3    3 Olivillo 40.0 24.0 30  1.218
4    4 Olivillo 18.5 22.0 12  0.222
5    5   Laurel 33.0 29.0 25  0.989
6    6 Olivillo 57.0 45.0 50  4.203
```

```
# Some checking
datana::descstat(df[, c("dbh","toth","totv")])
```

```
dbh    toth    totv
n          382.000 382.000 382.000
Minimum     11.500   9.800   0.044
Maximum    143.400  53.500  18.595
Mean        48.752  27.115   2.899
Median      45.250  26.500   1.816
Std. Dev.   23.006   8.192   3.229
CV(%)       47.190  30.211 111.381
```

We will create new columns having the standard names use for tree-level variables based upon the IUFRO, as follows

```
df$d <- df$dbh # for the diameter at breast height
df$h <- df$toth # for total height
df$v <- df$totv # para el volumen total
```

A continuación se muestra una lista de modelos de volumen, los que se ajustarán de forma lineal con la función `lm()`.

```
modelstobefit <- list(mod1 = list(expr = v ~ I(d^2) + I(d^2 * h^2) + I(d6),
                                  pred.f = function(y, ...) y,
                                  type = "lm"),
                      mod2 = list(expr = I(d^2 / v) ~ I(1 / h),
                                  pred.f = function(y, d, ...) d^2 / y,
                                  type = "lm"),
                      mod3 = list(expr = I(log10(v)) ~ I(log10(d)) + I(log10(h)),
```

```
                    pred.f = function(y, ...) 10^y,
                    type = "lm"),
          mod4 = list(expr = I(log(v)) ~ I(log(d)) + I(log(h)),
                    pred.f = function(y, ...) exp(y),
                    type = "lm"),
          mod5 = list(expr = I(sqrt(v)) ~ I(log(d)) + I(log(h)),
                    pred.f = function(y, ...) y^2,
                    type = "lm")
          )
```

Note que es necesario cargar, antes de usar la función `bankfit()`, la librería `datana`, que desde acá se llama internamente a la función `modresults()`.

Para realizar el ajuste basta con llamar la función `bankfit`:

```
outfit <- bankfit(models = modelstobefit, data = df)
## vemos los contenidos de `ajuste'
lapply(outfit, names)
```

```
$mod1
[1] "mod"    "summ"   "data"   "pred.f" "type"

$mod2
[1] "mod"    "summ"   "data"   "pred.f" "type"

$mod3
[1] "mod"    "summ"   "data"   "pred.f" "type"

$mod4
[1] "mod"    "summ"   "data"   "pred.f" "type"

$mod5
[1] "mod"    "summ"   "data"   "pred.f" "type"
```

Los resultados del ajuste del modelo `mod1` se pueden verificar manipulando la salida como una lista

```
outfit$mod1$mod
```

```
Call:
lm(formula = models[[x]][["expr"]], data = data)

Coefficients:
 (Intercept)        I(d^2)  I(d^2 * h^2)         I(d6)
```

```
-1.342e+00      1.021e-04      3.168e-07      7.510e-02
```

Es posible guardar los resultados en un archivo `.rdata`, lo que es útil y necesario para **predecir** con otra función llamada `bankpred`.

```
## guarda sólo los modelos ajustados y su respectivo pred.f
bankfit(models = modelstobefit, data = df, file = "./resources/fitted.rdata")
## guarda modelos ajustados, pred.f, summary y los datos
bankfit(models = modelstobefit, data = df, file = "./resources/fitted-full.rdata",
        file.full = TRUE)
```

2.1.

3. bankpred

3.1. Sobre esta función

La función `bankpred` fue creada principalmente para los procesos de ajuste y validación de modelos, específicamente esta función toma una lista de modelos **ya ajustados** y (opcionalmente) una base de datos y realiza las respectivas predicciones por modelo.

El listado de modelos ajustados debe venir de la salida de la función `bankfit`, por ejemplo:

```
bankfit(models = lista_modelos, data = mis_datos, file = "ajustado.rdata")
```

Creará el archivo "`ajustado.rdata`" para ser usado posteriormente.

3.2. Datos para ejemplos

Se cargarán los modelos ajustados en el ejemplo de §2. Siempre es necesario indicar sobre qué datos se hará la predicción. En este caso usaremos los mismos datos de ajuste (`biometrics::treevolruca2`), indicándolos en la opción `data`:

```
df <- biometrics::treevolruca2
df$d <- df$dap
df$h <- df$atot
df$v <- df$vtot

pred1 <- bankpred(file = "./resources/fitted.rdata", data = df)
head(pred1)
```

```
y.hat     y.pred model
1 12.8773454 12.8773454  mod1
2  0.6064593  0.6064593  mod1
3  1.3660116  1.3660116  mod1
4 -0.3537542 -0.3537542  mod1
5  0.9364930  0.9364930  mod1
6  4.8287693  4.8287693  mod1
```

Es posible usar datos distintos a los del ajuste (*i.e* validar) mediante la opción `data`. A continuación se usará el **ajuste** hecho con los datos de `biometrics::treevolruca2` (§Data for examples) en la base de datos `datana::idahohd2`:

```
library(datana)
df <- idahohd2
df$d <- df$dap
df$h <- df$atot

pred2 <- bankpred(file = "./resources/fitted.rdata", data = df)
head(pred2)
```

```
y.hat    y.pred model
1           NA        NA  mod1
1100 1448.1708 1.0502905  mod2
2     988.7129 2.3303023  mod2
3    1064.0640 2.5412005  mod2
4    1436.4513 0.9022234  mod2
5    1340.3514 1.0773294  mod2
```

Se puede ver que las predicciones con datos distintos no son completas, en este caso el modelo 1 no se pudo predecir puesto que la variable respuesta $d6$ no existe en la nueva base de datos. En tales casos la predicción quedará con un valor asignado de `NA`.

Referencias

R Core Team. (2025). *R: A language and environment for statistical computing* [http://www.R-project.org]. R Foundation for Statistical Computing. Vienna, Austria.

Salas-Eljatib, C., & Campos, N. (2025). `datana`*: Data and functions to accompany Análisis de datos con R* [R package version 1.1.4]. https://doi.org/10.32614/CRAN.package.datana

Salas-Eljatib, C., Campos, N., & Marivil, M. (2025). `biometrics`*: Functions and datasets for forest biometrics and modelling* [R package version 1.0.2]. https://doi.org/10.32614/CRAN.package.biometrics