Juntando R y Python con rpy2

In [1]:
```
import pandas as pd
import numpy as np
import os
# recordar instalar con conda install -c r rpy2
os.environ['R_HOME'] = 'C:/ProgramData/Anaconda3/Lib/R'
```

In [2]:
```
from IPython.display import Image
```

In [3]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [4]:
```
import rpy2
```

In [5]:
```
import rpy2.robjects as ro #interactuamos directamente con r
import rpy2.robjects.numpy2ri #permite conversión entre un dato y otro, de numpy a rpy2
```

```
Unable to determine R library path: Command '('C:/ProgramData/Anaconda3/Lib/R\\bin\\Rscript',
'-e', 'cat(Sys.getenv("LD_LIBRARY_PATH"))')' returned non-zero exit status 1.
```

In [6]:
```
rpy2.robjects.numpy2ri.activate()
```

In [7]:
```
codigo_r = """
saludar <- function(cadena){
    return(paste("Hola,",cadena))
}
"""
```

In [8]:
```
ro.r(codigo_r)
```

Out[8]:
```
<rpy2.robjects.functions.SignatureTranslatedFunction object at 0x000001E39A1A2B40> [RTYPES.CL
OSXP]
R classes: ('function',)
```

In [9]:
```
saludar_py = ro.globalenv["saludar"]
```

In [10]:
```
saludar_py("Antonio Banderas")
```

Out[10]:
```
StrVector with 1 elements.

'Hola, Antonio Banderas'
```

In [11]:
```
res = saludar_py("Antonio Banderas")
res[0]
```

Out[11]:
```
'Hola, Antonio Banderas'
```

In [12]:
```
type(res)
```

Out[12]:
```
rpy2.robjects.vectors.StrVector
```

In [13]:
```
print(saludar_py.r_repr()) #desde python podemos acceder a R y bisceversa
```

```
function (cadena)
{
    return(paste("Hola,", cadena))
}
```

In [14]:
```
var_from_python = ro.FloatVector(np.arange(1,5,0.1))
```

In [15]:
```
var_from_python
```

Out[15]:
FloatVector with 40 elements.

1.000000 1.100000 1.200000 ... 4.700000 4.800000 4.900000

In [16]:
```
print(var_from_python.r_repr())
```

```
c(1, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2, 2.1, 2.2,
2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3, 3.1, 3.2, 3.3, 3.4, 3.5,
3.6, 3.7, 3.8, 3.9, 4, 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8,
4.9)
```

In [17]:
```
ro.globalenv["var_to_r"] = var_from_python # añade en r una variable var_to_r a partir de var
```

In [18]:
```
ro.r("var_to_r")
```

Out[18]:
```
array([1. , 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. , 2.1, 2.2,
       2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3. , 3.1, 3.2, 3.3, 3.4, 3.5,
       3.6, 3.7, 3.8, 3.9, 4. , 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8,
       4.9])
```

In [19]:
```
ro.r("sum(var_to_r)")
```

Out[19]:
```
array([118.])
```

In [20]:
```
ro.r("mean(var_to_r)")
```

Out[20]:
```
array([2.95])
```

In [21]:
```
ro.r("sd(var_to_r)")
```

Out[21]:
```
array([1.16904519])
```

In [22]:
```
np.sum(var_from_python)
```

Out[22]:
```
118.00000000000007
```

In [23]:
```
np.mean(var_from_python)
```

Out[23]:
```
2.950000000000002
```

In [24]:
```
ro.r("summary(var_to_r)")
```

Out[24]:
```
array([1.   , 1.975, 2.95 , 2.95 , 3.925, 4.9  ])
```

In [25]:
```
#ro.r("hist(var_to_r, breaks = 6)")
```

Trabajar de forma conjunta entre R y Python

In [26]:
```
from rpy2.robjects.packages import importr #importamos librerías directamente desde R
```

In [27]:
```
#ro.r("install.packages('extRemes')")
extremes = importr("extRemes") #library(extRemes)
```

In [28]:
```
fevd = extremes.fevd
```

In [29]:
```
print(fevd.__doc__)
```

```
Wrapper around an R function.

The docstring below is built from the R documentation.

description
-----------


 Fit a univariate extreme value distribution functions (e.g., GEV, GP, PP, Gumbel, or Exponen
tial) to data; possibly with covariates in the parameters.



fevd(
    x,
    data,
    threshold = rinterface.NULL,
    threshold_fun = ~,
    location_fun = ~,
    scale_fun = ~,
    shape_fun = ~,
    use_phi = False,
    type = c,
    method = c,
    initial = rinterface.NULL,
    span,
    units = rinterface.NULL,
    time_units = days,
    period_basis = year,
    na_action = <rpy2.rinterface.ListSexpVector object at 0x000001E39D5D5F40> [RTYPES.VECSX
P],
    optim_args = rinterface.NULL,
    priorFun = rinterface.NULL,
    priorParams = rinterface.NULL,
    proposalFun = rinterface.NULL,
    proposalParams = rinterface.NULL,
    iter = 9999.0,
    weights = 1.0,
    blocks = rinterface.NULL,
    verbose = False,
)
```

Args:
x : 'fevd': 'x' can be a numeric vector, the name of a column of 'data' or a formula giving the data to which the EVD is to be fit. In the case of the latter two, the 'data' argument must be specified, and must have appropriately named columns.

object : A list object of class "fevd" as returned by 'fevd'.

data : A data frame object with named columns giving the data to be fit, as well as any data necessary for modeling non-stationarity through the threshold and/or any of the parameters.

threshold : numeric (single or vector). If fitting a peak over threshold (POT) model (i.e., 'type' = "PP", "GP", "Exponential") this is the threshold over which (non-inclusive) data (or excesses) are used to estimate the parameters of the distribution function. If the length is greater than 1, then the length must be equal to either the length of 'x' (or number of rows of 'data') or to the number of unique arguments in 'threshold.fun'.

threshold.fun : formula describing a model for the thresholds using columns from 'data'. Any valid formula will work. 'data' must be supplied if this argument is anything other than ~ 1. Not for use with 'method' "Lmoments".

use.phi : logical; should the log of the scale parameter be used in the numerical optimization (for 'method' "MLE", "GMLE" and "Bayesian" only)? For the ML and GML estimation, this may make things more stable for some data.

type : 'fevd': character stating which EVD to fit. Default is to fit the generalized extreme value (GEV) distribution function (df).

method : 'fevd': character naming which type of estimation method to use. Default is to use maximum likelihood estimation (MLE).

initial : A list object with any named parameter component giving the initial value estimates for starting the numerical optimization (MLE/GMLE) or the MCMC iterations (Bayesian). In the case of MLE/GMLE, it is best to obtain a good intial guess, and in the Bayesian case, it is perhaps better to choose poor initial estimates. If NULL (default), then L-moments estimates and estimates based on Gumbel moments will be calculated, and whichever yields the lowest negative log-likelihood is used. In the case of 'type' "PP", an additional MLE/GMLE estimate is made for the generalized Pareto (GP) df, and parameters are converted to those of the Poisson Process (PP) model. Again, the initial estimates yielding the lowest negative log-likelihoo value are used for the initial guess.

span : single numeric giving the number of years (or other desired temporal unit) in the data set. Only used for POT models, and only important in the estimation for the PP model, but important for subsequent estimates of return levels for any POT model. If missing, it will be calculated using information from 'time.units'.

units : (optional) character giving the units of the data, which if given may be used subsequently (e.g., on plot axis labels, etc.).

time.units : character string that must be one of "hours", "minutes", "seconds", "days", "months", "years", "m/hour", "m/minute", "m/second", "m/day", "m/month", or "m/year"; where m is a number. If 'span' is missing, then this argument is used in determining the value of 'span'. It is also returned with the output and used subsequently for plot labelling, etc.

period.basis : character string giving the units for the period. Used only for plot labelling and naming output vectors from some of the method functions (e.g., for establishing what the period represents for the return period).

rperiods : numeric vector giving the return period(s) for which it is desired to calculate the corresponding return levels.

period : character string naming the units for the return period.

burn.in : The first 'burn.in' values are thrown out before calculating anything from the MCMC sample.

a : when plotting empirical probabilies and such, the function 'ppoints' is called, which has this argument 'a'.

d : numeric determining how to scale the rate parameter for the point process. If NULL, the function will attempt to scale based on the values of 'period.basis' and 'time.units', the first of which must be "year" and the second of which must be one of "days", "months", "years", "hours", "minutes" or "seconds". If none of these are the case, then 'd' should be specified, otherwise, it is not necessary.

na.action : function to be called to handle missing values. Generally, this should remain at the default (na.fail), and the user should take care to impute missing values in an appropriate manner as it may have serious consequences on the results.

optim.args : A list with named components matching exactly any arguments that the user wishes to specify to 'optim', which is used only for MLE and GMLE methods. By default, the "BFGS" method is used along with 'grlevd' for the gradient argument. Generally, the 'grlevd' function is used for the 'gr' option unless the user specifies otherwise, or the optimization method does not take gradient information.

priorFun : character naming a prior df to use for methods GMLE and Bayesian. The default for GMLE (not including Gumbel or Exponential types) is to use the one suggested by Martins and Stedinger (2000, 2001) on the shape parameter; a beta df on -0.5 to 0.5 with parameters 'p' and 'q'. Must take 'x' as its first argument for 'method' "GMLE". Optional arguments for the default function are 'p' and 'q' (see details section).

Note : if this argument is not NULL and 'method' is set to "MLE", it will be changed to "GMLE".

priorParams : named list containing any prior df parameters (where the list names are the same as the function argument names). Default for GMLE (assuming the default function is used) is to use 'q' = 6 and 'p' = 9. Note that in the Martins and Stedinger (2000, 2001) papers, they use a different EVD parametrization than is used here such that a positive shape parameter gives the upper bounded distribution instead of the heavy-tail one (as emloyed here). To be consistent with these papers, 'p' and 'q' are reversed inside the code so that they have the same interpretation as in the papers.

proposalFun : For Bayesian estimation only, this is a character naming a function used to generate proposal parameters at each iteration of the MCMC. If NULL (default), a random walk chain is used whereby if theta.i is the current value of the parameter, the proposed new parameter theta.star is given by theta.i + z, where z is drawn at random from a normal df.

proposalParams : A named list object describing any optional arguments to the 'proposalFun' function. All functions must take argument 'p', which must be a vector of the parameters, and 'ind', which is used to identify which parameter is to be proposed. The default 'proposalFun' function takes additional arguments 'mean' and 'sd', which must be vectors of length equal to the number of parameters in the model (default is to use zero for the mean of z for every parameter and 0.1 for its standard deviation).

iter : Used only for Bayesian estimation, this is the number of MCMC iterations to do.

weights : numeric of length 1 or n giving weights to be applied in the likelihood calculations (e.g., if there are data points to be weighted more/less heavily than others).

blocks : An optional list containing information required to fit point process models in a computationally-efficient manner by using only the exceedances and not the observations below the threshold(s). See details for further information.

FUN : character string naming a function to use to estimate the parameters from the MCMC sample. The function is applied to each column of the 'results' component of the returned 'fevd' object.

verbose : logical; should progress information be printed to the screen? If TRUE, for MLE/GMLE, the argument 'trace' will be set to 6 in the call to 'optim'.

prange : matrix whose columns are numeric vectors of length two for each parameter in the model giving the parameter range over which trace plots should be made. Default is to use either +/- 2 * std. err. of the parameter (first choice) or, if the standard error cannot be calculated, then +/- 2 * log2(abs(parameter)). Typically, these values seem to work very well for these plots.

... : Not used by most functions here. Optional arguments to 'plot' for the various 'plot' method functions.

details

See text books on extreme value analysis (EVA) for more on univariate EVA (e.g., Coles, 2001 and Reiss and Thomas, 2007 give fairly accessible introductions to the topic for most audiences; and Beirlant et al., 2004, de Haan and Ferreira, 2006, as well as Reiss and Thomas, 2007 give more complete theoretical treatments). The extreme value distributions (EVDs) have theoretical support for analyzing extreme values of a process. In particular, the generalized extreme value (GEV) df is appropriate for modeling block maxima (for large blocks, such as annual maxima), the generalized Pareto (GP) df models threshold excesses (i.e., x - u | x > u and u a high threshold).

The GEV df is given by

Pr(X <= x) = G(x) = exp[-(1 + shape*(x - location)/scale)^(-1/shape)]

for 1 + shape*(x - location) > 0 and scale > 0. If the shape parameter is zero, then the df is defined by continuity and simplies to

G(x) = exp(-exp((x - location)/scale)).

The GEV df is often called a family of distribution functions because it encompasses the three types of EVDs: Gumbel (shape = 0, light tail), Frechet (shape > 0, heavy tail) and the reverse Weibull (shape < 0, bounded upper tail at location - scale/shape). It was first found by R. von Mises (1936) and also independently noted later by meteorologist A. F. Jenkins (1955). It enjoys theretical support for modeling maxima taken over large blocks of a series of data.

The generalized Pareo df is given by (Pickands, 1975)

Pr(X <= x) = F(x) = 1 - [1 + shape*(x - threshold)/scale]^(-1/shape)

where 1 + shape*(x - threshold)/scale > 0, scale > 0, and x > threshold. If shape = 0, then the GP df is defined by continuity and becomes

F(x) = 1 - exp(-(x - threshold)/scale).

There is an approximate relationship between the GEV and GP distribution functions where the GP df is approximately the tail df for the GEV df. In particular, the scale parameter of the GP is a function of the threshold (denote it scale.u), and is equivalent to scale + shape*(threshold - location) where scale, shape and location are parameters from the equivalent GEV df. Similar to the GEV df, the shape parameter determines the tail behavior, where shape = 0 gives rise to the exponential df (light tail), shape > 0 the Pareto df (heavy tail) and shape < 0 the Beta df (bounded upper tail at location - scale.u/shape). Theoretical justification supports the use of the GP df family for modeling excesses over a high threshold (i.e., y = x - threshold). It is assumed here that x , q describe x (not y = x - threshold). Similarly, the random draws are y + threshold.

If interest is in minima or deficits under a low threshold, all of the above applies to the negative of the data (e.g., - max(-X_1,...,-X_n) = min(X_1, ..., X_n)) and fevd can be used so long as the user first negates the data, and subsequently realizes that the return levels (and location parameter) given will be the negative of the desired return levels (and location parameter), etc.

The study of extremes often involves a paucity of data, and for small sample sizes, L-moments may give better estimates than competing methods, but penalized MLE (cf. Coles and Dixon, 1999; Martins and Stedinger, 2000; 2001) may give better estimates than the L-moments for such samples. Martins and Stedinger (2000; 2001) use the terminology generalized MLE, which is also used here.

Non-stationary models:

The current code does not allow for non-stationary models with L-moments estimation.

For MLE/GMLE (see El Adlouni et al 2007 for using GMLE in fitting models whose parameters vary) and Bayesian estimation, linear models for the parameters may be fit using formulas, in which case the data argument must be supplied. Specifically, the models allowed for a set of covariates, y, are:

location(y) = mu0 + mu1 * f1(y) + mu2 * f2(y) + ...

scale(y) = sig0 + sig1 * g1(y) + sig2 * g2(y) + ...

log(scale(y)) = phi(y) = phi0 + phi1 * g1(y) + phi2 * g2(y) + ...

shape(y) = xi0 + xi1 * h1(y) + xi2 * h2(y) + ...

For non-stationary fitting it is recommended that the covariates within the generalized linear models are (at least approximately) centered and scaled (see examples below). It is generally ill-advised to include covariates in the shape parameter, but there are situations where it makes sense.

Non-stationary modeling is accomplished with fevd by using formulas via the arguments: threshold.fun , location.fun , scale.fun and shape.fun . See examples to see how to do this.

Initial Value Estimates:

In the case of MLE/GMLE, it can be very important to get good initial estimates (e.g., see the examples below). fevd attempts to find such estimates, but it is also possible for the user to supply their own initial estimates as a list object using the initial argument, whereby the components of the list are named according to which parameter(s) they are associated with. In particular, if the model is non-stationary, with covariates in the location (e.g., mu(t) = mu0 + mu1 * t), then initial may have a component named location that may contain either a single number (in which case, by default, the initial value for mu1 will be zero) or a vector of length two giving initial values for mu0 and mu1.

For Bayesian estimation, it is good practice to try several starting values at different points to make sure the initial values do not affect the outcome. However, if initial values are not passed in, the MLEs are used (which probably is not a good thing to do, but is more likely to yield good results).

For MLE/GMLE, two (in the case of PP, three) initial estimates are calculated along with their associated likelihood values. The initial estimates that yield the highest likelihood are used. These methods are:

1. L-moment estimates.

2. Let m = mean(xdat) and s = sqrt(6 * var(xdat)) / pi. Then, initial values assigend for the lcoation parameter when either initial is NULL or the location component of initial is NULL, are m - 0.57722 * s. When initial or the scale component of initial is NULL, the initial value for the scale parameter is taken to be s, and when initial or its shape component is NULL, the initial value for the shape parameter is taken to be 1e-8 (because these initial estimates are moment-based estimates for the Gumbel df, so the initial value is taken to be near zero).

3. In the case of PP, which is often the most difficult model to fit, MLEs are obtained for a GP model, and the resulting parameter estimates are converted to those of the approximately equivalent PP model.

In the case of a non-stationary model, if the default initial estimates are used, then the intercept term for each parameter is given the initial estimate, and all other parameters are set to zero initially. The exception is in the case of PP model fitting where the MLE from the GP fits are used, in which case, these parameter estimates may be non-zero.

The generalized MLE (GMLE) method:

This method places a penalty (or prior df) on the shape parameter to help ensure a better fit. The procedure is nearly identical to MLE, except the likelihood, L, is multiplied by the prior df, p(shape); and because the negative log-likelihood is used, the effect is that of subtracting this term. Currently, there is no supplied function by this package to calculate the gradient for the GMLE case, so in particular, the trace plot is not the trace of the actual negative log-likelihood (or gradient thereof) used in the estimation.

Bayesian Estimation:

It is possible to give your own prior and proposal distribution functions using the appropriate arguments listed above in the arguments section. At each iteration of the chain, the parameters are updated one at a time in random order. The default method uses a random walk chain for the proposal and normal distributions for the parameters.

Plotting output:

plot : The plot method function will take information from the fevd output and make any of various useful plots. The default, regardless of estimation method, is to produce a 2 by

2 panel of plots giving some common diagnostic plots. Possible types (determined by the typ e argument) include:

1. primary (default): yields the 2 by 2 panel of plots given by 3, 4, 6 and 7 below.

2. probprob : Model probabilities against empirical probabilities (obtained from the ppoin ts function). A good fit should yield a straight one-to-one line of points. In the case of a non-stationary model, the data are first transformed to either the Gumbel (block maxima mod els) or exponential (POT models) scale, and plotted against probabilities from these standard ized distribution functions. In the case of a PP model, the parameters are first converted t o those of the approximately equivalent GP df, and are plotted against the empirical data thr eshold excesses probabilities.

3. qq : Empirical quantiles against model quantiles. Again, a good fit will yield a straig ht one-to-one line of points. Generally, the qq-plot is preferred to the probability plot in 1 above. As in 2, for the non-stationary case, data are first transformed and plotted agains t quantiles from the standardized distributions. Also as in 2 above, in the case of the PP m odel, parameters are converted to those of the GP df and quantiles are from threshold excesse s of the data.

4. qq2 : Similar to 3, first data are simulated from the fitted model, and then the qq-plot between them (using the function qqplot from this self-same package) is made between them, which also yields confidence bands. Note that for a good fitting model, this should again yi eld a straight one-to-one line of points, but generally, it will not be as well-behaved as the plot in 3. The one-to-one line and a regression line fitting the quantiles is also show n. In the case of a non-stationary model, simulations are obtained by simulating from an app ropriate standardized EVD, re-ordered to follow the same ordering as the data to which the mo del was fit, and then back transformed using the covariates from data and the parameter est imates to put the simulated sample back on the original scale of the data. The PP model is h andled analogously as in 2 and 3 above.

5. and 6. Zplot : These are for PP model fits only and are based on Smith and Shively (199 5). The Z plot is a diagnostic for determining whether or not the random variable, Zk, defin ed as the (possibly non-homogeneous) Poisson intensity parameter(s) integrated from exceedanc e time k - 1 to exceedance time k (beginning the series with k = 1) is independent exponentia lly distributed with mean 1.

For the Z plot, it is necessary to scale the Poisson intensity parameter appropriately. For example, if the data are given on a daily time scale with an annual period basis, then this p arameter should be divided by, for example, 365.25. From the fitted fevd object, the funct ion will try to account for the correct scaling based on the two components period.basis an d time.units . The former currently must be year and the latter must be one of days , m onths , years , hours , minutes or seconds . If none of these are valid for your specif ic data (e.g., if an annual basis is not desired), then use the d argument to explicitly sp ecify the correct scaling.

7. hist : A histogram of the data is made, and the model density is shown with a blue dash ed line. In the case of non-stationary models, the data are first transformed to an appropri ate standardized EVD scale, and the model density line is for the self-same standardized EVD. Currently, this does not work for non-stationary POT models.

8. density : Same as 5, but the kernel density (using function density) for the data is p lotted instead of the histogram. In the case of the PP model, block maxima of the data are ca lculated and the density of these block maxima are compared to the PP in terms of the equival ent GEV df. If the model is non-stationary GEV, then the transformed data (to a stationary G umbel df) are used. If the model is a non-stationary POT model, then currently this option i s not available.

9. rl : Return level plot. This is done on the log-scale for the abscissa in order that th e type of EVD can be discerned from the shape (i.e., heavy tail distributions are concave, li ght tailed distributions are straight lines, and bounded upper-tailed distributions are conve x, asymptoting at the upper bound). 95 percent CIs are also shown (gray dashed lines). In t he case of non-stationary models, the data are plotted as a line, and the effective return levels (by default the 2-period (i.e., the median), 20-period and 100-period are used; period is usually annual) are also shown (see, e.g., Gilleland and Katz, 2011). In the case of the

PP model, the equivalent GEV df (stationary model) is assumed and data points are block maxima, where the blocks are determined from information passed in the call to fevd . In particular, the span argument (which, if not passed by the user, will have been determined by fevd using time.units along with the number of points per year (which is estimated from time.units) are used to find the blocks over which the maxima are taken. For the non-stationary case, the equivalent GP df is assumed and parameters are converted. This helps facilitate a more meaningful plot, e.g., in the presence of a non-constant threshold, but otherwise constant parameters.

10. trace : In each of cases (b) and (c) below, a 2 by the number of parameters panel of plots are created.

(a) L-moments: Not available for the L-moments estimation.

(b) For MLE/GMLE, the likelihood traces are shown for each parameter of the model, whereby all but one parameter is held fixed at the MLE/GMLE values, and the negative log-likelihood is graphed for varying values of the parameter of interest. Note that this differs greatly from the profile likelihood (see, e.g., profliker) where the likelihood is maximized over the remaining parameters. The gradient negative log-likelihoods are also shown for each parameter. These plots may be useful in diagnosing any fitting problems that might arise in practice. For ease of interpretation, the gradients are shown directly below the likleihoods for each parameter.

(c) For Bayesian estimation, the usual trace plots are shown with a gray vertical dashed line showing where the burn.in value lies; and a gray dashed horizontal line through the posterior mean. However, the posterior densities are also displayed for each parameter directly above the usual trace plots. It is not currently planned to allow for adding the prior dnsities to the posterior density graphs, as this can be easily implemented by the user, but is more difficult to do generally.

As with ci and distill , only plot need be called by the user. The appropriate choice of the other functions is automatically determined from the fevd fitted object.

Note that when blocks are provided to fevd , certain plots
that require the full set of observations (including non-exceedances)
cannot be produced.

Summaries and Printing:

summary and print method functions are available, and give different information depending on the estimation method used. However, in each case, the parameter estimates are printed to the screen. summary returns some useful output (similar to distill, but in a list format). The print method function need not be called as one can simply type the name of the fevd fitted object and return to execute the command (see examples below). The deviance information criterion (DIC) is calculated for the Bayesian estimation method as DIC = D(mean(theta)) + 2 * pd, where pd = mean(D(theta)) - D(mean(theta)), and D(theta) = -2 * log-likelihood evaluated at the parameter values given by theta. The means are taken over the posterior MCMC sample. The default estimation method for the parameter values from the MCMC sample is to take the mean of the sample (after removing the first burn.in samples). A good alternative is to set the FUN argument to postmode in order to obtain the posterior mode of the sample.

Using Blocks to Reduce Computation in PP Fitting:

If blocks is supplied, the user should
provide only the exceedances and not all of the data values. For
stationary models, the list should contain a component called
nBlocks indicating the number of observations within a block, where
blocks are defined in a manner analogous to that used in GEV
models. For nonstationary models, the list may contain one or more of
several components. For nonstationary models with covariates, the list
should contain a data component analogous to the data argument,
providing values for the blocks. If the threshold varies, the list
should contain a threshold component that is analogous to the
threshold argument. If some of the observations within any block are
missing (assuming missing at random or missing completely at random),

```
the list should contain a  proportionMissing  component that is a
vector with one value per block indicating the proportion of
observations missing for the block. To weight the blocks, the list can
contain a  weights  component with one value per block. Warning: to
properly analyze nonstationary models, any covariates, thresholds, and
weights must be constant within each block.
```

In [30]:
```
data = pd.read_csv("../../Data-Sets/datasets/time/time_series1.txt", sep= "\s+")
```

In [31]:
```
data
```

Out[31]:

	YYYYMMDD	HHMM	M(m/s)
0	19830101	0	7.9
1	19830101	100	8.2
2	19830101	200	8.5
3	19830101	300	9.0
4	19830101	400	9.9
...	...	...	...
275347	20140531	1900	6.9
275348	20140531	2000	6.5
275349	20140531	2100	6.2
275350	20140531	2200	5.9
275351	20140531	2300	5.6

275352 rows × 3 columns

In [32]:
```
#Juntando dos columnas de un dataset.
data = pd.read_csv("../../Data-Sets/datasets/time/time_series.txt", sep= "\s+", skiprows = 1,
```

In [33]:
```
data.head(5)
```

Out[33]:

	wind_speed
date_time	
1983-01-01 00:00:00	7.9
1983-01-01 01:00:00	8.2
1983-01-01 02:00:00	8.5
1983-01-01 03:00:00	9.0
1983-01-01 04:00:00	9.9

In [34]:
```
data.shape
```

Out[34]:
```
(275352, 1)
```

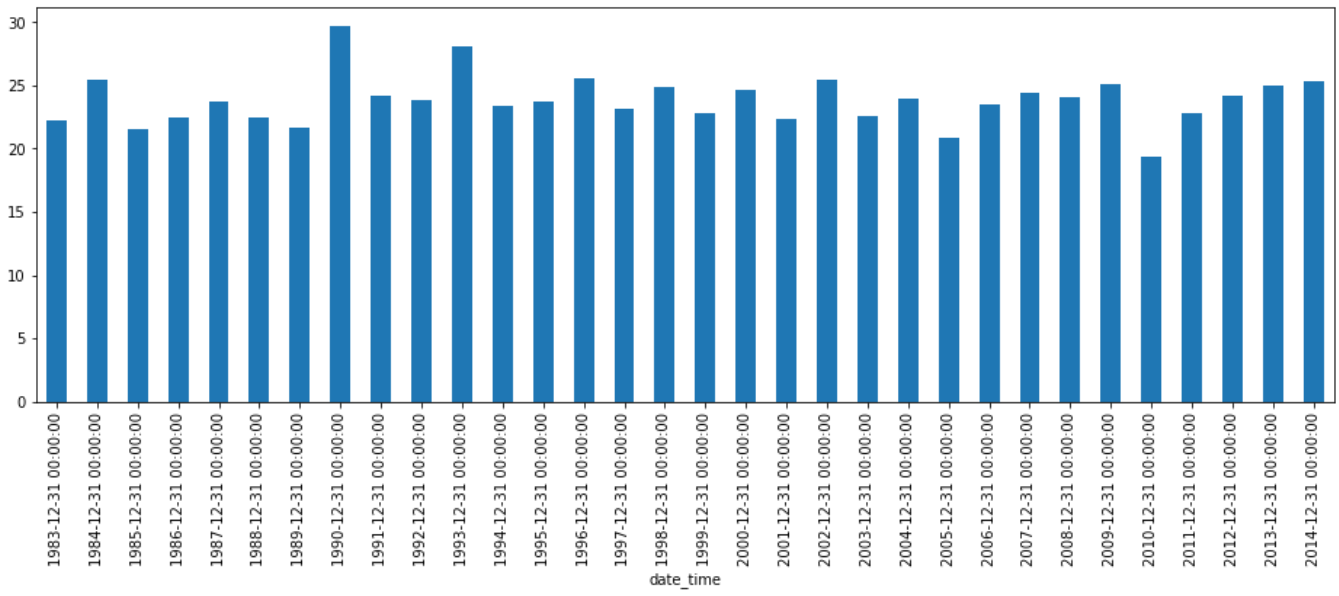
In [35]:
```
max_ws = data.wind_speed.groupby(pd.Grouper(freq="A")).max()
```

In [36]:
```
max_ws
```

Out[36]:
```
date_time
1983-12-31    22.2
1984-12-31    25.5
1985-12-31    21.5
1986-12-31    22.5
1987-12-31    23.7
1988-12-31    22.5
1989-12-31    21.7
1990-12-31    29.7
1991-12-31    24.2
1992-12-31    23.8
1993-12-31    28.1
1994-12-31    23.4
1995-12-31    23.7
1996-12-31    25.6
1997-12-31    23.2
1998-12-31    24.9
1999-12-31    22.8
2000-12-31    24.6
2001-12-31    22.3
2002-12-31    25.5
2003-12-31    22.6
2004-12-31    24.0
2005-12-31    20.8
2006-12-31    23.5
2007-12-31    24.4
2008-12-31    24.1
2009-12-31    25.1
2010-12-31    19.4
2011-12-31    22.8
2012-12-31    24.2
2013-12-31    25.0
2014-12-31    25.3
Freq: A-DEC, Name: wind_speed, dtype: float64
```

In [37]:
```
max_ws.plot(kind = "bar", figsize = (16,5))
```

Out[37]:
```
<AxesSubplot:xlabel='date_time'>
```



In [38]:
```
result  = fevd(max_ws.values, type = "GEV", method = "GMLE") #función de R
```

In [39]:
```
print(type(result))
```

```
<class 'rpy2.robjects.vectors.ListVector'>
```

In [40]:
```
result.r_repr
```

Out[40]:
```
<bound method RObjectMixin.r_repr of <rpy2.robjects.vectors.ListVector object at 0x000001E3A0
89A780> [RTYPES.VECSXP]
R classes: ('fevd',)
[LangSexpV..., StrSexpVe..., FloatSexp..., BoolSexpV..., ..., StrSexpVe..., StrSexpVe..., Lis
tSexpV..., ListSexpV...]
  call: <class 'rpy2.robjects.language.LangVector'>
  Rlang( (function (x, data, threshold = NULL, threshold.fun = ~1, location.fun = ~1,  )
  data.name: <class 'rpy2.robjects.vectors.StrVector'>
  <rpy2.robjects.vectors.StrVector object at 0x000001E3A0C038C0> [RTYPES.STRSXP]
R classes: ('character',)
['struct..., '23.8, ..., '22.6, ..., '25.3),..., '']
  weights: <class 'numpy.ndarray'>
  array([1.])
  missing.values: <class 'rpy2.robjects.vectors.BoolArray'>
  <rpy2.robjects.vectors.BoolArray object at 0x000001E3A5A51D40> [RTYPES.LGLSXP]
R classes: ('array',)
[       0,        0,        0, ...,        0,        0,        0,        0]
...
  x: <class 'rpy2.robjects.vectors.StrVector'>
  <rpy2.robjects.vectors.StrVector object at 0x000001E3A0C03B00> [RTYPES.STRSXP]
R classes: ('character',)
['na.fail']
  priorFun: <class 'rpy2.robjects.vectors.StrVector'>
  <rpy2.robjects.vectors.StrVector object at 0x000001E3A0C03600> [RTYPES.STRSXP]
R classes: ('character',)
['location', 'scale', 'shape']
<rpy2.robjects.vectors.ListVector object at 0x000001E3A089A780> [RTYPES.VECSXP]
R classes: ('fevd',)
[LangSexpV..., StrSexpVe..., FloatSexp..., BoolSexpV..., ..., StrSexpVe..., StrSexpVe..., Lis
tSexpV..., ListSexpV...]
<rpy2.robjects.vectors.ListVector object at 0x000001E3A089A780> [RTYPES.VECSXP]
R classes: ('fevd',)
[LangSexpV..., StrSexpVe..., FloatSexp..., BoolSexpV..., ..., StrSexpVe..., StrSexpVe..., Lis
tSexpV..., ListSexpV...]>
```

In [41]:
```
result.names
```

Out[41]:
StrVector with 20 elements.

'call' 'data.name' 'weights' ... 'parnames' 'results' 'initial....

In [42]:
```
print(result.names)
```

```
 [1] "call"           "data.name"      "weights"        "missing.values"
 [5] "in.data"        "x"              "priorFun"       "priorParams"
 [9] "method"         "type"           "period.basis"   "par.models"
[13] "const.loc"      "const.scale"    "const.shape"    "n"
[17] "na.action"      "parnames"       "results"        "initial.results"
```

In [43]:
```
res = result.rx("results")
print(res[0])
```

```
$par
```

```
  location      scale      shape
23.0639415  1.7576913 -0.1628816

$value
[1] 1e+16

$counts
function gradient
       1        1

$convergence
[1] 0

$message
NULL

$hessian
         location scale shape
location        0     0     0
scale           0     0     0
shape           0     0     0

$num.pars
$num.pars$location
[1] 1

$num.pars$scale
[1] 1

$num.pars$shape
[1] 1
```

In [44]:

```
print(res.r_repr)
```

```
<bound method RObjectMixin.r_repr of <rpy2.robjects.vectors.ListVector object at 0x000001E3A0
BFD580> [RTYPES.VECSXP]
R classes: ('list',)
[ListSexpVector]
  results: <class 'rpy2.rinterface.ListSexpVector'>
  <rpy2.rinterface.ListSexpVector object at 0x000001E3A0C0E780> [RTYPES.VECSXP]>
```

In [45]:

```
print(res[0].r_repr)
```

```
<bound method RObjectMixin.r_repr of <rpy2.robjects.vectors.ListVector object at 0x000001E3A0
C0EAC0> [RTYPES.VECSXP]
R classes: ('list',)
[Floa..., Floa..., IntS..., IntS..., NULL..., Floa..., List...]
  par: <class 'rpy2.rinterface.FloatSexpVector'>
  <rpy2.rinterface.FloatSexpVector object at 0x000001E3A0C0EB80> [RTYPES.REALSXP]
  value: <class 'rpy2.rinterface.FloatSexpVector'>
  <rpy2.rinterface.FloatSexpVector object at 0x000001E3A0C0EDC0> [RTYPES.REALSXP]
  counts: <class 'rpy2.rinterface.IntSexpVector'>
  <rpy2.rinterface.IntSexpVector object at 0x000001E3A0C0EB80> [RTYPES.INTSXP]
  convergence: <class 'rpy2.rinterface.IntSexpVector'>
  <rpy2.rinterface.IntSexpVector object at 0x000001E3A0C0EDC0> [RTYPES.INTSXP]
  message: <class 'rpy2.rinterface_lib.sexp.NULLType'>
  <rpy2.rinterface_lib.sexp.NULLType object at 0x000001E39A16F400> [RTYPES.NILSXP]
  hessian: <class 'rpy2.rinterface.FloatSexpVector'>
  <rpy2.rinterface.FloatSexpVector object at 0x000001E3A0C120C0> [RTYPES.REALSXP]
  num.pars: <class 'rpy2.rinterface.ListSexpVector'>
  <rpy2.rinterface.ListSexpVector object at 0x000001E3A0C12100> [RTYPES.VECSXP]>
```

In [46]:
```
loc, scale, shape = res[0].rx("par")[0]
```

In [47]:
```
loc, scale, shape
```

Out[47]:
```
(23.063941519915588, 1.7576912874286317, -0.1628816367714792)
```

Función mágica para R

In [48]:
```
%load_ext rpy2.ipython
```

```
C:\ProgramData\Anaconda3\lib\site-packages\rpy2\robjects\packages.py:366: UserWarning: The sy
mbol 'quartz' is not in this R namespace/package.
  warnings.warn(
```

In [49]:
```
help(rpy2.ipython.rmagic.RMagics.R)
```

```
Help on function R in module rpy2.ipython.rmagic:

R(self, line, cell=None, local_ns=None)
    ::

      %R [-i INPUT] [-o OUTPUT] [-n] [-w WIDTH] [-h HEIGHT] [-p POINTSIZE] [-b BG] [--noisola
tion] [-u {px,in,cm,mm}]
             [-r RES] [--type {cairo,cairo-png,Xlib,quartz}] [-c CONVERTER] [-d DISPLAY]
             [code ...]

    Execute code in R, optionally returning results to the Python runtime.

    In line mode, this will evaluate an expression and convert the returned
    value to a Python object.  The return value is determined by rpy2's
    behaviour of returning the result of evaluating the final expression.

    Multiple R expressions can be executed by joining them with
    semicolons::

        In [9]: %R X=c(1,4,5,7); sd(X); mean(X)
        Out[9]: array([ 4.25])

    In cell mode, this will run a block of R code. The resulting value
    is printed if it would be printed when evaluating the same code
    within a standard R REPL.

    Nothing is returned to python by default in cell mode::

        In [10]: %%R
           ....: Y = c(2,4,3,9)
           ....: summary(lm(Y~X))

        Call:
        lm(formula = Y ~ X)

        Residuals:
            1     2     3     4
         0.88 -0.24 -2.28  1.64

        Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
        (Intercept)   0.0800     2.3000   0.035    0.975
        X             1.0400     0.4822   2.157    0.164

        Residual standard error: 2.088 on 2 degrees of freedom
```

```
        Multiple R-squared: 0.6993,Adjusted R-squared: 0.549
        F-statistic: 4.651 on 1 and 2 DF,  p-value: 0.1638

    In the notebook, plots are published as the output of the cell::

        %R plot(X, Y)

    will create a scatter plot of X bs Y.

    If cell is not None and line has some R code, it is prepended to
    the R code in cell.

    Objects can be passed back and forth between rpy2 and python via the
    -i -o flags in line::

        In [14]: Z = np.array([1,4,5,10])

        In [15]: %R -i Z mean(Z)
        Out[15]: array([ 5.])

        In [16]: %R -o W W=Z*mean(Z)
        Out[16]: array([  5.,  20.,  25.,  50.])

        In [17]: W
        Out[17]: array([  5.,  20.,  25.,  50.])

    The return value is determined by these rules:

    * If the cell is not None (i.e., has contents), the magic returns None.

    * If the final line results in a NULL value when evaluated
      by rpy2, then None is returned.

    * No attempt is made to convert the final value to a structured array.
      Use %Rget to push a structured array.

    * If the -n flag is present, there is no return value.

    * A trailing ';' will also result in no return value as the last
      value in the line is an empty string.

    optional arguments:
      -i INPUT, --input INPUT
                            Names of input variable from `shell.user_ns` to be assigned to R
variables of the same names
                            after using the Converter self.converter. Multiple names can be p
assed separated only by
                            commas with no whitespace.
      -o OUTPUT, --output OUTPUT
                            Names of variables to be pushed from rpy2 to `shell.user_ns` afte
r executing cell body (rpy2's
                            internal facilities will apply ri2ro as appropriate). Multiple na
mes can be passed separated
                            only by commas with no whitespace.
      -n, --noreturn        Force the magic to not return anything.

    Plot:
      Arguments to plotting device

      -w WIDTH, --width WIDTH
                            Width of plotting device in R.
      -h HEIGHT, --height HEIGHT
                            Height of plotting device in R.
      -p POINTSIZE, --pointsize POINTSIZE
                            Pointsize of plotting device in R.
      -b BG, --bg BG        Background of plotting device in R.
```

```
    SVG:
      SVG specific arguments

      --noisolation         Disable SVG isolation in the Notebook. By default, SVGs are isola
ted to avoid namespace
                            collisions between figures. Disabling SVG isolation allows to ref
erence previous figures or
                            share CSS rules across a set of SVGs.

    PNG:
      PNG specific arguments

      -u <{px,in,cm,mm}>, --units <{px,in,cm,mm}>
                            Units of png plotting device sent as an argument to *png* in R. O
ne of ["px", "in", "cm",
                            "mm"].
      -r RES, --res RES     Resolution of png plotting device sent as an argument to *png* in
R. Defaults to 72 if *units*
                            is one of ["in", "cm", "mm"].
      --type <{cairo,cairo-png,Xlib,quartz}>
                            Type device used to generate the figure.
      -c CONVERTER, --converter CONVERTER
                            Name of local converter to use. A converter contains the rules to
convert objects back and
                            forth between Python and R. If not specified/None, the defaut con
verter for the magic's module
                            is used (that is rpy2's default converter + numpy converter + pan
das converter if all three
                            are available).
      -d DISPLAY, --display DISPLAY
                            Name of function to use to display the output of an R cell (the l
ast object or function call
                            that does not have a left-hand side assignment). That function wi
ll have the signature
                            `(robject, args)` where `robject` is the R objects that is an out
put of the cell, and `args` a
                            namespace with all parameters passed to the cell.
      code
```

In [50]:

```
%R X=c(1,4,5,7); sd(X); mean(X)
```

Out[50]:

```
array([4.25])
```

In [51]:

```
%%R
Y = c(2,4,3,9)
LM = lm(Y~X)
summary(LM)
```

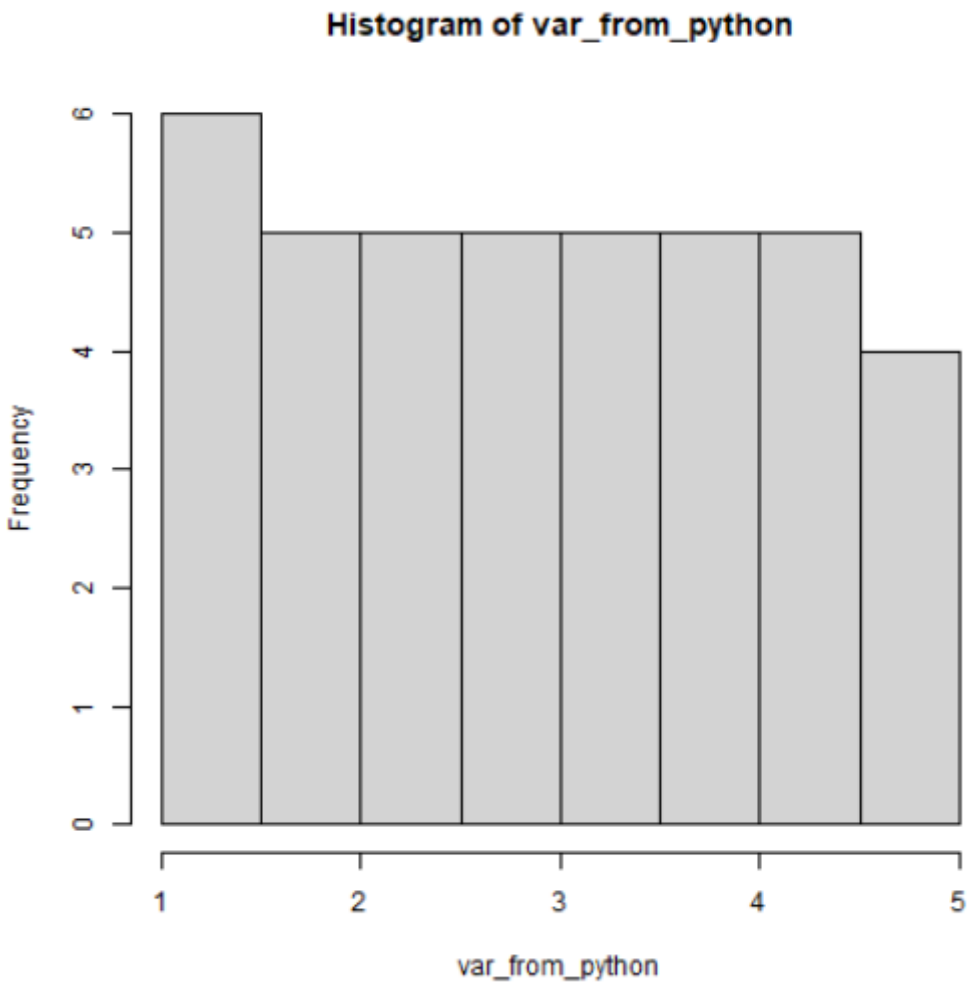
```
Call:
lm(formula = Y ~ X)

Residuals:
    1     2     3     4
 0.88 -0.24 -2.28  1.64

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.0800     2.3000   0.035    0.975
X             1.0400     0.4822   2.157    0.164

Residual standard error: 2.088 on 2 degrees of freedom
Multiple R-squared:  0.6993,    Adjusted R-squared:  0.549
```

```
F-statistic: 4.651 on 1 and 2 DF,  p-value: 0.1638
```

In [52]:
```
# %R -i result plot.fevd(result)
```

In [53]:
```
#%R -i var_from_python hist(var_from_python)
Image(filename="C:/Users/Kevin/Desktop/Kevin/1-Cursos/1-Machine Learning/CURSO UDEMY ML/17-Ju
```

Out[53]:

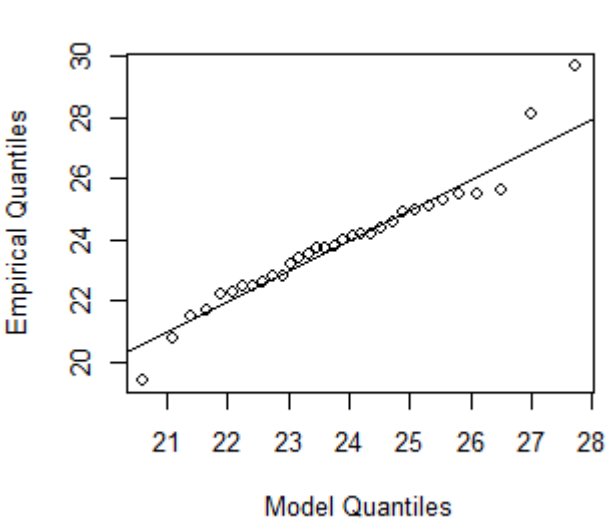


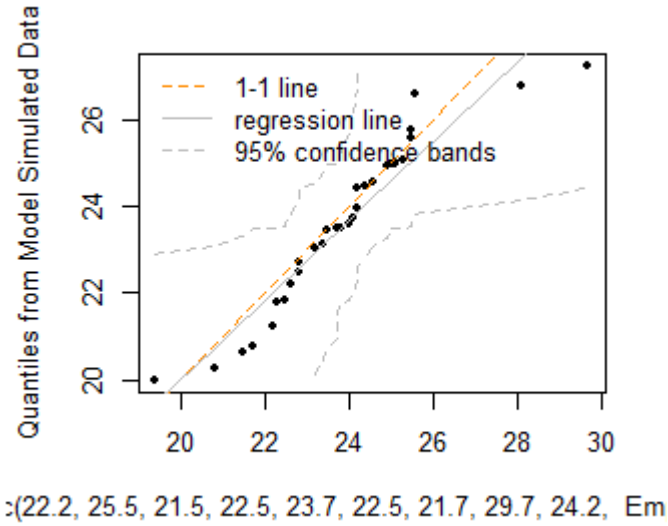
In [54]:
```
#%R -i result plot.fevd(result)
```

In [55]:
```
#ro.globalenv["result"] = result
#ro.r("plot.fevd(result)")
```

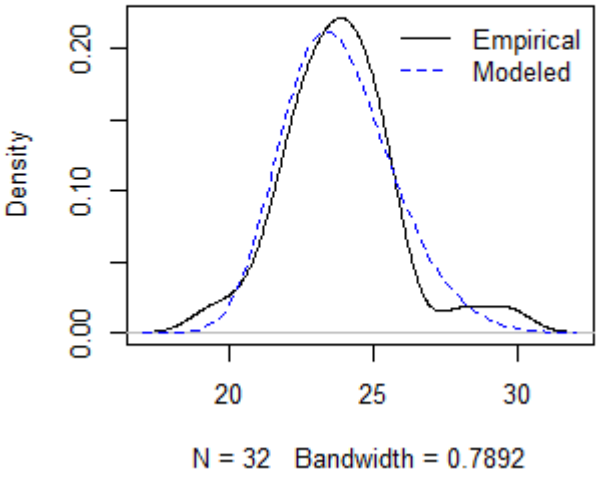
In [56]:
```
Image(filename="C:/Users/Kevin/Desktop/Kevin/1-Cursos/1-Machine Learning/CURSO UDEMY ML/17-Ju
```

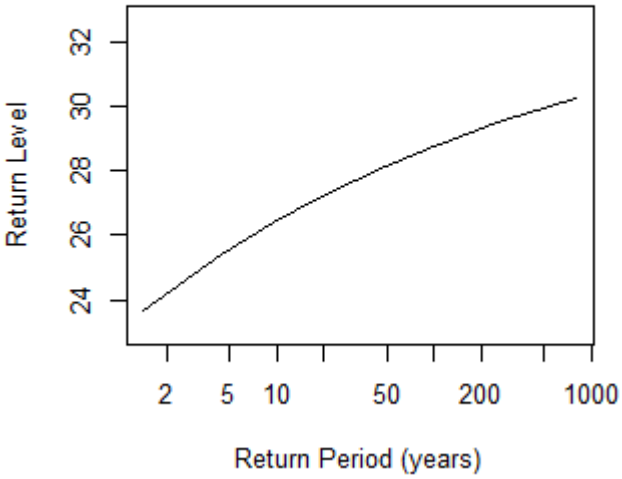
Out[56]:





:(22.2, 25.5, 21.5, 22.5, 23.7, 22.5, 21.7, 29.7, 24.2, Em





Un ejemplo coplejo de R, Python y Rmagic

In [65]:

```
metodos = ["MLE", "GMLE", "Bayesian", "Lmoments"]
tipo = ["GEV", "Gumbel"]
```

In [66]:

```
for t in tipo:
    for m in metodos:
        print("Tipo de Ajuste: ", t)
        print("Tipo de Método del Ajuste: ", m)
        result = fevd(max_ws.values, method = m, type = t)
        print(result.rx("results")[0])
        ro.globalenv["result"] = result
        ro.r("plot.fevd(result)")
        # %R -i result plot.fevd(result) crashea a la mierda
```

```
Tipo de Ajuste:  GEV
Tipo de Método del Ajuste:  MLE
$par
  location      scale      shape
23.0517078  1.8085853 -0.1497984

$value
[1] 66.22729
```

```
$counts
function gradient
      28        6

$convergence
[1] 0

$message
NULL

$hessian
           location      scale     shape
location  9.1825628 -0.3982934  11.22038
scale    -0.3982934 21.4422632  19.81854
shape    11.2203805 19.8185358 172.17411

$num.pars
$num.pars$location
[1] 1

$num.pars$scale
[1] 1

$num.pars$shape
[1] 1



Tipo de Ajuste:  GEV
Tipo de Método del Ajuste:  GMLE
$par
  location      scale      shape
23.0639415  1.7576913 -0.1628816

$value
[1] 1e+16

$counts
function gradient
       1        1

$convergence
[1] 0

$message
NULL

$hessian
         location scale shape
location        0     0     0
scale           0     0     0
shape           0     0     0

$num.pars
$num.pars$location
[1] 1

$num.pars$scale
[1] 1

$num.pars$shape
[1] 1



Tipo de Ajuste:  GEV
```

```
Tipo de Método del Ajuste:  Bayesian
[[ 2.30639415e+01  5.64001179e-01 -1.62881637e-01  0.00000000e+00]
 [ 2.31580069e+01  5.39151415e-01 -1.62881637e-01  2.00000000e+00]
 [ 2.31580069e+01  5.39151415e-01 -1.62881637e-01  0.00000000e+00]
 ...
 [ 2.23253900e+01  6.91351130e-01 -1.18715773e-01  2.00000000e+00]
 [ 2.23253900e+01  6.60909939e-01 -1.20705948e-02  2.00000000e+00]
 [ 2.23573297e+01  5.69648801e-01 -9.13097113e-02  3.00000000e+00]]
Tipo de Ajuste:  GEV
Tipo de Método del Ajuste:  Lmoments
[23.06394152  1.75769129 -0.16288164]
R[write to console]: Error in res[1, (nloc + nsc + 1):np] <- initial$shape :
  number of items to replace is not a multiple of replacement length

R[write to console]: In addition:
R[write to console]: Warning message:

R[write to console]: In (function (x, data, threshold = NULL, threshold.fun = ~1, location.fu
n = ~1,  :
R[write to console]:

Tipo de Ajuste:  Gumbel
Tipo de Método del Ajuste:  MLE
$par
 location     scale
22.905876  1.814452

$value
[1] 67.38487

$counts
function gradient
      14        5

$convergence
[1] 0

$message
NULL

$hessian
          location     scale
location  9.719762 -4.762621
scale    -4.762621 21.535378

$num.pars
$num.pars$location
[1] 1

$num.pars$scale
[1] 1

$num.pars$shape
NULL



Tipo de Ajuste:  Gumbel
Tipo de Método del Ajuste:  GMLE
$par
 location     scale
22.905876  1.814452

$value
[1] 67.38487
```

```
$counts
function gradient
      14        5

$convergence
[1] 0

$message
NULL

$hessian
          location     scale
location  9.719762 -4.762621
scale    -4.762621 21.535378

$num.pars
$num.pars$location
[1] 1

$num.pars$scale
[1] 1

$num.pars$shape
NULL



Tipo de Ajuste:  Gumbel
Tipo de Método del Ajuste:  Bayesian
R[write to console]:  fevd: Using method MLE.  No default for specified arguments.

---------------------------------------------------------------------------
RRuntimeError                             Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_14652/3003418773.py in <module>
      3         print("Tipo de Ajuste: ", t)
      4         print("Tipo de Método del Ajuste: ", m)
----> 5         result = fevd(max_ws.values, method = m, type = t)
      6         print(result.rx("results")[0])
      7         ro.globalenv["result"] = result

C:\ProgramData\Anaconda3\lib\site-packages\rpy2\robjects\functions.py in __call__(self, *arg
s, **kwargs)
    199                 v = kwargs.pop(k)
    200                 kwargs[r_k] = v
--> 201         return (super(SignatureTranslatedFunction, self)
    202                 .__call__(*args, **kwargs))
    203

C:\ProgramData\Anaconda3\lib\site-packages\rpy2\robjects\functions.py in __call__(self, *arg
s, **kwargs)
    122             else:
    123                 new_kwargs[k] = conversion.py2rpy(v)
--> 124         res = super(Function, self).__call__(*new_args, **new_kwargs)
    125         res = conversion.rpy2py(res)
    126         return res

C:\ProgramData\Anaconda3\lib\site-packages\rpy2\rinterface_lib\conversion.py in _(*args, **kw
args)
     43 def _cdata_res_to_rinterface(function):
     44     def _(*args, **kwargs):
---> 45         cdata = function(*args, **kwargs)
     46         # TODO: test cdata is of the expected CType
     47         return _cdata_to_rinterface(cdata)

C:\ProgramData\Anaconda3\lib\site-packages\rpy2\rinterface.py in __call__(self, *args, **kwar
gs)
```

```
    808             )
    809             if error_occured[0]:
--> 810                 raise embedded.RRuntimeError(_rinterface._geterrmessage())
    811         return res
    812 

RRuntimeError: Error in res[1, (nloc + nsc + 1):np] <- initial$shape :
  number of items to replace is not a multiple of replacement length
```

Se puede aprovechar el potencial de cálculo estadístico de R combinado con el potencial lógico, de carga y administración de datos de python para sacar el mejor provecho de ambos lenguajes. **R** no es un lenguaje de programación sino una **herramienta estadística** y **Python** si es un lenguaje de programación que posee varias herramientas estadísticas.

In [ ]: