empirical 0.2.0

Probability Distributions as Models of Data

Abby Spurdle

December 3, 2018

Computes continuous (not step) empirical (and nonparametric) probability density, cumulative distribution and quantile functions. Supports univariate, multivariate and conditional probability distributions, some kernel smoothing features and weighted data (possibly useful mixed with fuzzy clustering). Can compute multivariate and conditional probabilities. Also, can compute conditional medians, quantiles and modes.

Pre-Intro

This package uses objects, which are also functions, which are also models, which are also probability distributions.

Some functions (constructors) return other functions (models), which can be evaluated. The resulting functions have attributes (which along with their arguments) determine their return values.

This is intermediate between a standard object oriented approach and a functional approach. And I believe that this is the best approach for implementing probability distributions, especially nonparametric probability distributions.

Introduction

This package computes what I refer to as empirical models (or empirical probability distributions).

Currently, this includes:

- Continuous empirical probability density functions (EPDFs).
- Continuous empirical cumulative distribution functions (ECDFs).
- Continuous empirical quantile functions (EQFs).

It supports univariate, multivariate and conditional probability distributions.

Currently, univariate models are computed differently to multivariate models:

- Univariate models use what I refer to as a quantile approach and compute a series of vertices (similar to a standard ECDF) and then use a cubic hermite spline to interpolate between them. By default, univariate models smooth the data, using a lowess style smoother, which I refer to as derandomization.
- Multivariate (and conditional) models use kernel smoothing, which I refer to as restacking.

EPDFs can be used to compute modes and visualize the shape of probability distributions. ECDFs can be used to compute probabilities and to a lesser extent, visualize the shape too. And EQFs can be used to compute medians and quantiles.

Also, we can compute conditional probabilities, medians, quantiles and modes from conditional probability distributions, which is similar to regression.

Models may be weighted (possibly useful mixed with fuzzy clustering).

The philosophy behind empirical models, is to model data directly, with as few assumptions as possible but with some support for robustness.

Important Notes

Currently, this package is experimental.

I'm planning to support categorical variables and random number generation in the future.

There are some problems with univariate models, especially univariate EPDFs. Their first derivative is not continuous which is particularly noticeable in the outer tails. This can interfere with mode computation, especially if trying to compute all modal points.

I tried to create a hybrid Kernel-Quantile approach, however, I wasn't able to get it to work as yet. I will try again later.

The function used to compute modes has had limited testing and there may be additional problems.

The current implementation is slow and needs to be optimized, however, this is not a current development priority.

Univariate models bind two additional data points by default. Univariate models need unique data points, if they're not unique then they're randomized first.

Weighted models have had limited testing too.

Loading the Packages

I'm going to load (and attach) the intoo, empirical, fclust and moments packages:

```
> library (intoo)
> library (empirical)
> library (fclust)
> library (moments)
```

Preparing the Data

I'm going to use the trees data (from the datasets package) and the unemployment data (from the fclust package):

```
> data (trees)
> data (unemployment)
```

And I'm going to convert to metric:

```
> # -> m
> Height = 0.3048 * trees$Height
> # -> cm
> Girth = 2.54 * trees$Girth
> # -> m ^ 3
> Volume = 0.0283168 * trees$Volume

> #total unemployment rate
> un.rate = unemployment$Total.Rate
> #long term unemployment rate
> lt.rate = unemployment$LongTerm.Share
```

New matrix objects:

```
> trees2 = cbind (Height, Girth, Volume)
> unemployment2 = cbind (un.rate, lt.rate)
```

I've provided some more information on the trees2 and unemployment2 data in Appendices.

Univariate Models (and Core Functionality)

Empirical probability density functions are produced by differentiating empirical cumulative distribution functions. We can use the epdfuv() function:

```
> f = epdfuv (Height)
```

We can print the object directly, however, I recommend using the object.info() function from the intoo package, which at the time of writing, needs some improvement:

```
> object.info (f)

value, 1
function (x)
{
    .epdfuv.eval(x)
}
class, 1
[1] "epdfuv"
%$%derandomize, logical, 1
[1] TRUE
%$%preserve, character, 1
[1] "mean"
%$%bind, logical, 1
[1] TRUE
%$%weighted, logical, 1
```

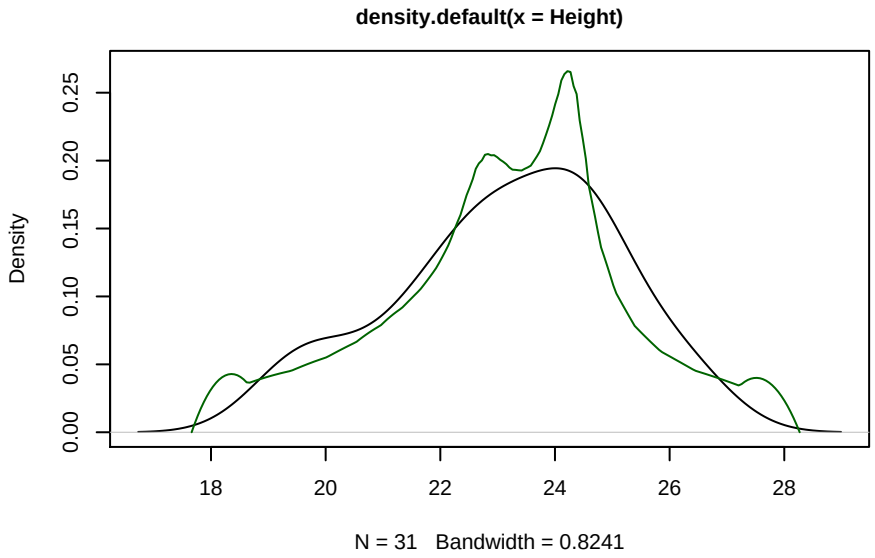
```
[1] FALSE
%$%drp, numeric, 1
[1] 0.5
%$%nhood, numeric, 1
[1] 16
%$%n, numeric, 1
[1] 33
%$%x, numeric, 33
[1] 17.66214 18.63810 19.40029 20.01935 20.53220 20.96086 21.32661 21.64504
%$%y, numeric, 33
[1] 0.00000 0.03125 0.06250 0.09375 0.12500 0.15625 0.18750 0.21875
%$%t, numeric, 33
[1] 0.00000000 0.03595790 0.04524868 0.05521615 0.06638265 0.07867495 0.09135053
[8] 0.10485044
%$%w, logical, 1
[1] NA
```

We can access a single attribute using the attribute operator, also from the intoo package, if required:

```
> f %$% x

 [1] 17.66214 18.63810 19.40029 20.01935 20.53220 20.96086 21.32661 21.64504
 [9] 21.92270 22.16340 22.37016 22.54980 22.71111 22.86475 23.01837 23.17535
[17] 23.33649 23.49821 23.65683 23.80772 23.94759 24.07647 24.19646 24.31483
[25] 24.44206 24.59401 24.78957 25.04712 25.39054 25.84517 26.44033 27.22178
[33] 28.26879
```

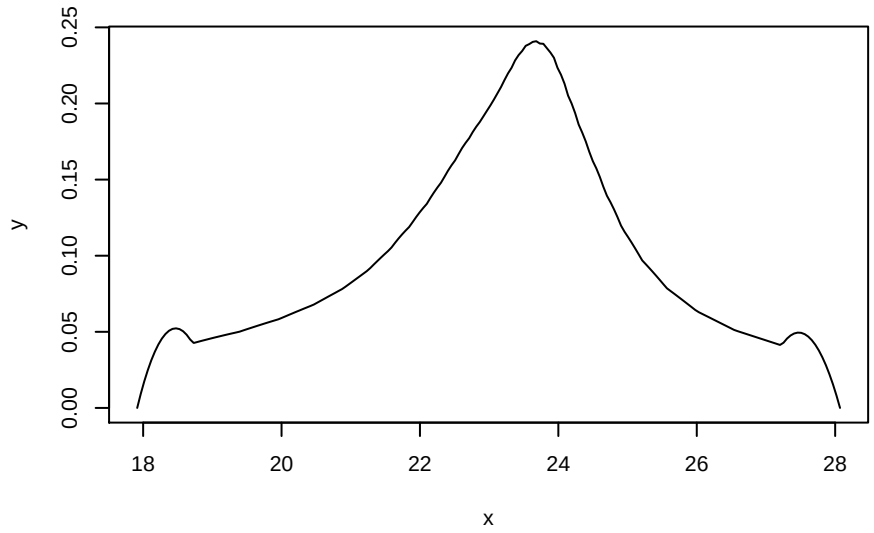
We can plot the densities from both density (from the stats package) and epdfuv objects, and then compare them:

```
> plot (density (Height), ylim=c (0, 0.27) )
> lines (f, col="darkgreen")
```



Or using a different smoothing parameter:

```
> f.75 = epdfuv (Height, drp=0.75)
> plot (f.75)
```



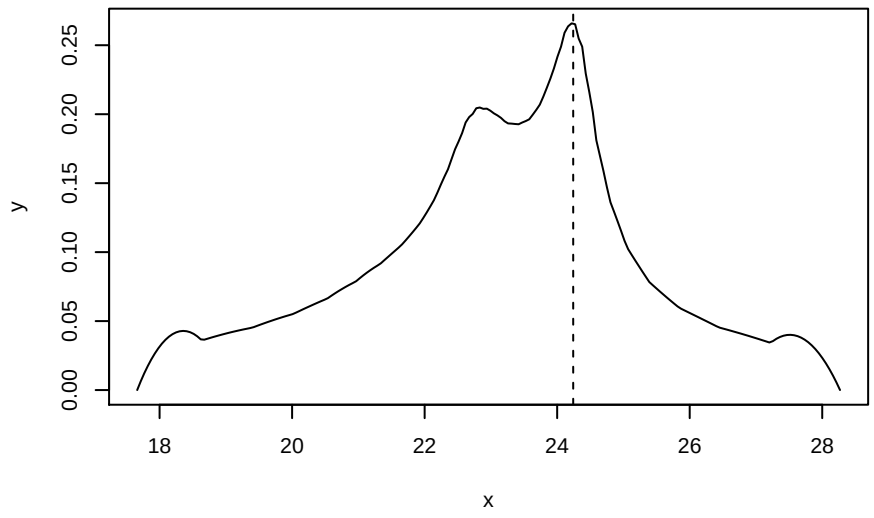
Reiterating, univariate EPDFs do not have continuous first derivatives which is particularly noticeable in the outer tails. Another problem is that density estimates in the outer tails appear too high which implies that the density estimates in other regions are too low.

We can evaluate the object (which is a function), however, this isn't that useful:

```
> mean.Height = mean (Height)
> f (mean.Height)

[1] 0.19687
```

Also, we can compute the empirical mode, using the emode() function:

```
> mh = emode (f)
> plot (f)
> abline (v=mh, lty=2)
> mh

[1] 24.24218
```



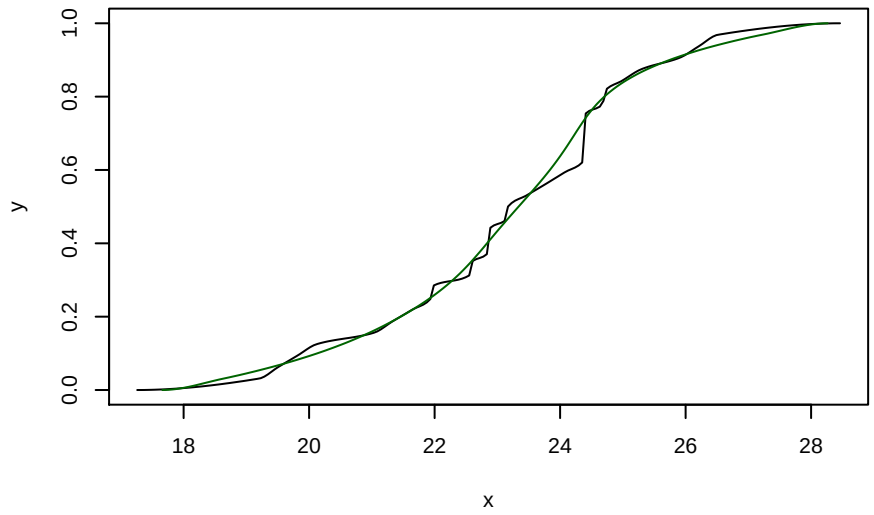
We can compute vertices for a continuous (not step) empirical cumulative distribution function, using the following expression:

$$\mathbb{P}(X \leq x) = F(x) = \frac{\sum_i I(x_i^* \leq x) - 1}{n - 1}$$

Where $I()$ is 1 if the enclosed expression is true and is 0 if false, n is the number of data points and x^* is a vector of data points.

We can use the ecdfuv() function:

```
> F.unsmooth = ecdfuv (Height, FALSE)
> F = F.smooth = ecdfuv (Height)
> plot (F.unsmooth)
> lines (F.smooth, col="darkgreen")
```



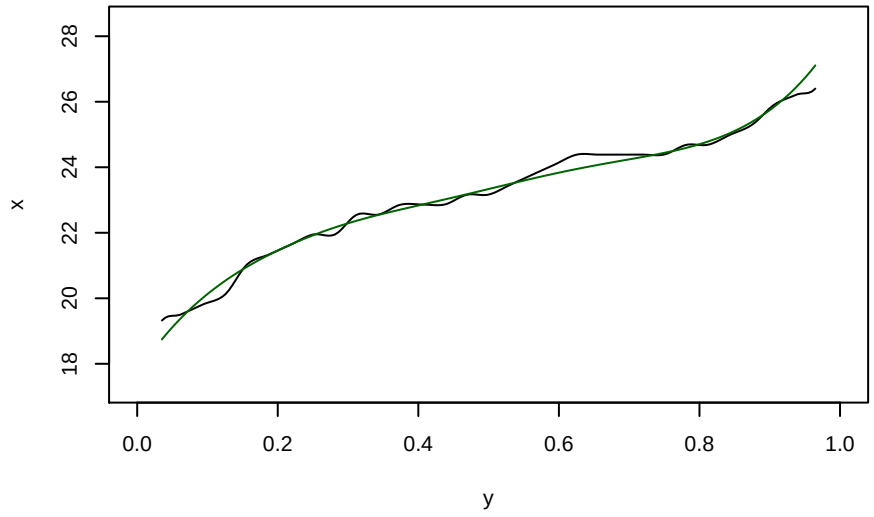
Like epdfuv objects we can evaluate an ecdfuv object.

Let's say that we want to compute the probability that Height is between 20 and 22 meters:

```
> F (22) - F (20)

[1] 0.1668722
```

We can compute an empirical quantile function by inverting the ECDF:

```
> F.inv.unsmooth = ecdfuv.inverse (Height, FALSE)
> F.inv = F.inv.smooth = ecdfuv.inverse (Height)
> plot (F.inv.unsmooth)
> lines (F.inv.smooth, col="darkgreen")
```



Like epdfuv and ecdfuv objects we can evaluate an ecdfuv.inverse object.

Let's say we want to compute the median:

```
> median = F.inv (0.5)
> median
```

```
[1] 23.33547
```

Note that EQFs are not the exact inverse of ECDFs, because of the way that I've implemented them.

```
> F (median)
```

```
[1] 0.4995401
```

Bivariate Models

We can compute bivariate models using the following expressions:

$$f(x_1, x_2) = \frac{\sum_{i \in 2:(n-1)} [g(x^*_{[i,1]}, \text{bw}_1, x_1) * g(x^*_{[i,2]}, \text{bw}_2, x_2)]}{n-2}$$

Where:

$$g(x_0, \text{bw}, x) = \frac{2}{\text{bw}} l(\frac{2}{\text{bw}}(x - x_0))$$

And where $l()$ is the restacking PDF (kernel) and bw is the bandwidth.

$$F(x_1, x_2) = \frac{\sum_{i \in 2:(n-1)} [G(x^*_{[i,1]}, \text{bw}_1, x_1) * G(x^*_{[i,2]}, \text{bw}_2, x_2)]}{n-2}$$

Where:

$$G(x_0, \text{bw}, x) = L(\frac{2}{\text{bw}}(x - x_0))$$

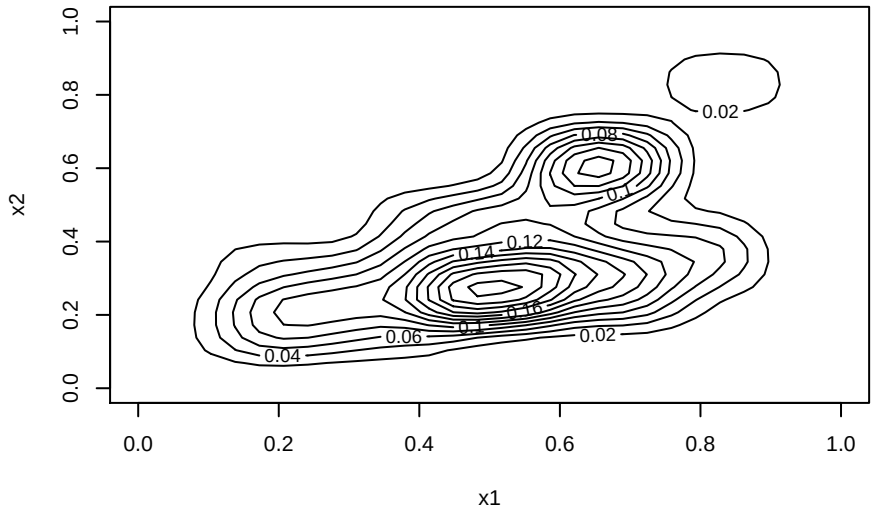
And where $L()$ is the restacking CDF.

I've excluded the first and last data points, so that the ECDF of the first and last data points evaluates to 0 and 1, respectively, for consistency with univariate models.

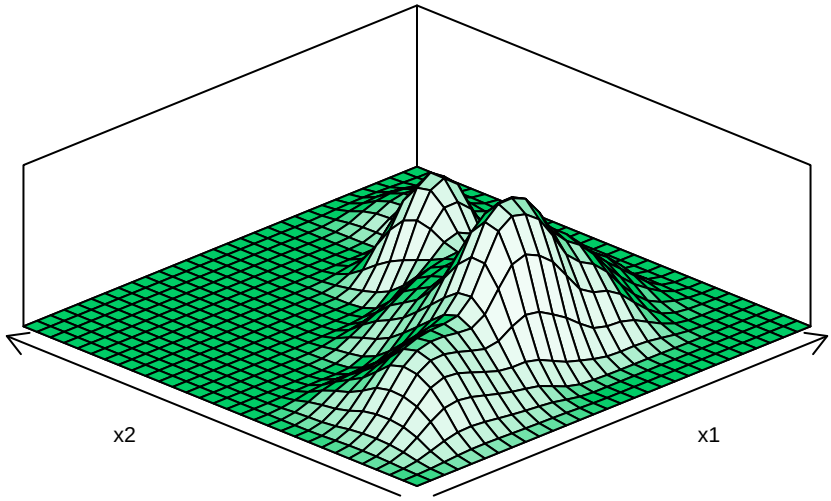
We can construct bivariate EPDFs and ECDFs using the epdfmv() and ecdfmv() functions:

```
> f = epdfmv (cbind (Height, Volume) )
> F = ecdfmv (cbind (Height, Volume) )
```

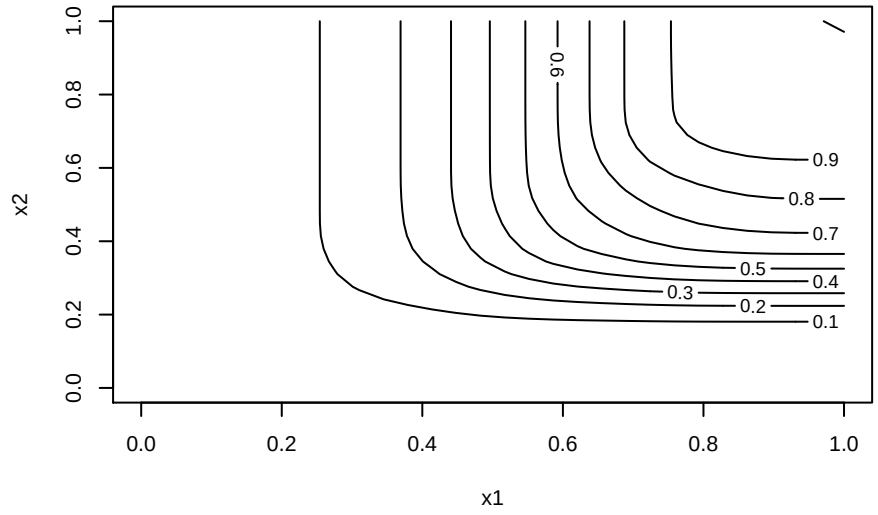
```
> plot (f)
```



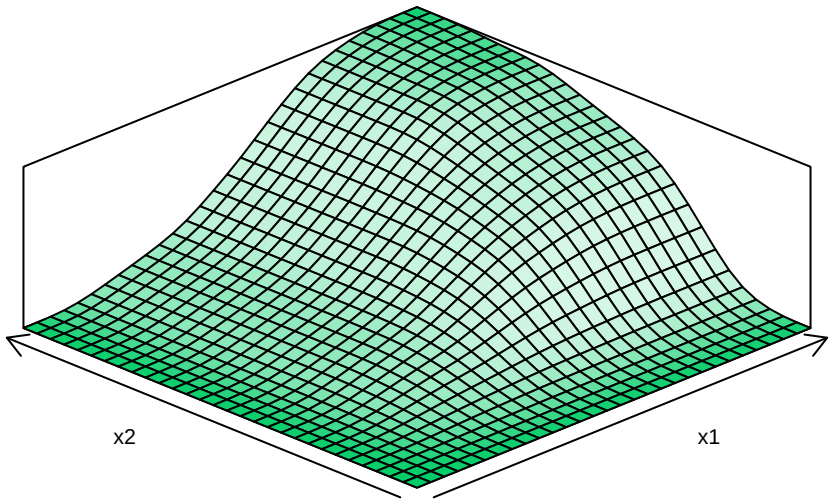

```
> plot (f, TRUE)
```




```
> plot (F)
```




```
> plot (F, TRUE)
```



Or alternatively:

```
> #plot (f, all=TRUE)
```

Bivariate models aren't really useful in themselves except for visualizing the shape of probability distributions. However, we can use bivariate (and multivariate models generally) to compute conditional probability distributions and compute multivariate probabilities, which are discussed later.

Multivariate Models

Bivariate models generalize to multivariate models (with m $>$ 2) easily, however, they're difficult to visualize:

$$f(x_1, x_2, ..., x_m) = \frac{\sum_{i \in 2:(n-1)} [g(x^*_{[i,1]}, \text{bw}_1, x_1) * g(x^*_{[i,2]}, \text{bw}_2, x_2)) * ... * g(x^*_{[i,m]}, \text{bw}_m, x_m)]}{n-2}$$

$$F(x_1, x_2, ..., x_m) = \frac{\sum_{i \in 2:(n-1)} [G(x^*_{[i,1]}, \text{bw}_1, x_1) * G(x^*_{[i,2]}, \text{bw}_2, x_2)) * ... * G(x^*_{[i,m]}, \text{bw}_m, x_m)]}{n-2}$$

Where m is the number of variables.

```
> f = epdfmv (trees2)
> F = ecdfmv (trees2)
```

Currently, you can't plot multivariate models (with m > 2).

Conditional Models

We can derive conditional models from multivariate models.

In theory, we can compute a (univariate) conditional ECDF (on one variable) using:

$$\mathbb{P}(X_2 \leq x_2 \mid X_1 = x_1) = F(x_2) = \int_{-\infty}^{x_2} \frac{f_{X1,X2}(x_1, u)}{f_{X1}(x_1)} du$$

In theory, we can compute a (univariate) conditional ECDF (on two variables) using:

$$\mathbb{P}(X_3 \leq x_3 \mid X_1 = x_1, X_2 = x_2) = F(x_3) = \int_{-\infty}^{x_3} \frac{f_{X1,X2,X3}(x_1, x_2, u)}{f_{X1,X2}(x_1, x_2)} du$$

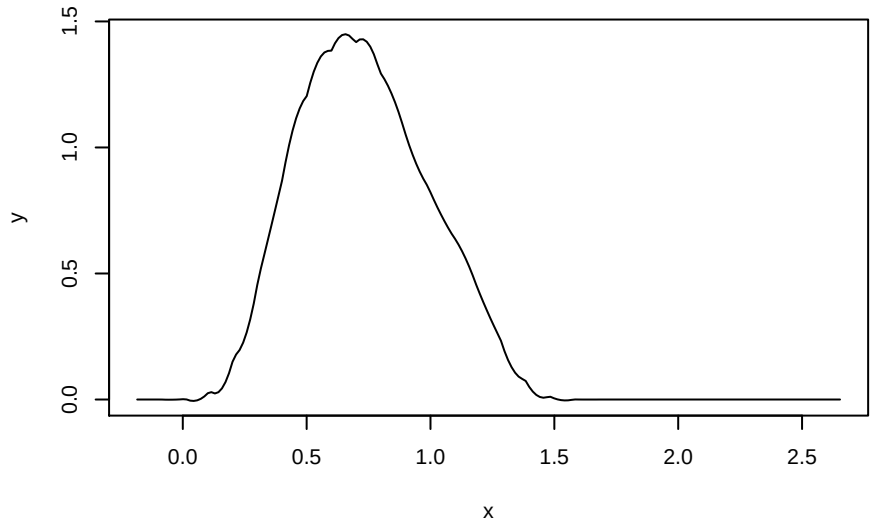
And this can be generalized to m variables.

We can construct conditional empirical models using the epdfc(), ecdfc() and ecdfc.inverse() functions:

```
> mean.Girth = mean (Girth)
> cf = epdfc (Volume, c (Height=mean.Height, Girth=mean.Girth), trees2)
> cF = ecdfc (Volume, c (Height=mean.Height, Girth=mean.Girth), trees2)
> cF.inv = ecdfc (Volume, c (Height=mean.Height, Girth=mean.Girth), trees2)
```

Or alternatively (quoted):

```
> cf = epdfc ("Volume", c (Height=mean.Height, Girth=mean.Girth), trees2,
     is.string=TRUE)

> plot (cf)
```



Computing Multivariate Probabilities

We can compute the probability that two random variables are between two pairs of values as:

$$\mathbb{P}(a_1 \leq X_1 \leq b_1, a_2 \leq X_2 \leq b_2) = F(b_1, b_2) - [F(a_1, b_2) + F(b_1, a_2)] + F(a_1, a_2)$$

Where a is the lower limits and b is the upper limits.

And for three variables:

$$\begin{aligned}\mathbb{P}(a_1 \leq X_1 \leq b_1, a_2 \leq X_2 \leq b_2, a_3 \leq X_3 \leq b_3) = & F(b_1, b_2, b_3) \\ & - [F(a_1, b_2, b_3) + F(b_1, a_2, b_3) + F(b_1, b_2, a_3)] \\ & + [F(a_1, a_2, b_3) + F(a_1, b_2, a_3) + F(b_1, a_2, a_3)] \\ & - F(a_1, a_2, a_3)\end{aligned}$$

This can be generalized to four or more variables.

Note that we are computing the probability over a rectangular region. This approach won't work for nonrectangular regions.

We can use the comb.prob() function, which takes three arguments:

```
> a = c (20, 20, 0.2)
> b = c (30, 24, 0.8)
> #(using multivariate model from earlier section)
> comb.prob (F, a, b)
```

```
[1] 0.03940915
```

Note that it's possible to compute multiple regions at once by making a and b matrices with each row representing one region.

Conditional Probabilities and Conditional Statistics

It's possible to compute conditional probabilities, medians, quantiles or modes from conditional models. I'm planning to support conditional means and variances in the near future.

We can compute the conditional probability in the same way as the univariate case:

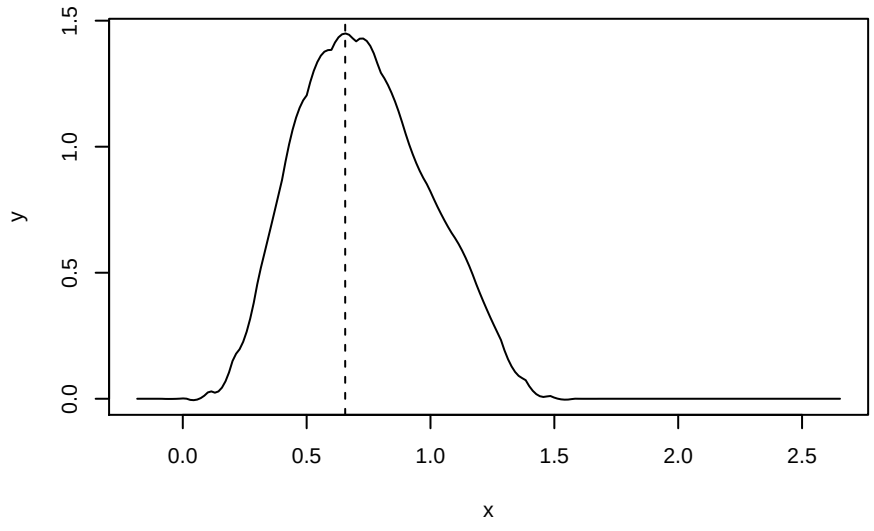
```
> #(using conditional model from earlier section)
> #probability that volume is between
> #0.2 and 0.8 cubic meters given mean height
> cF (0.8) - cF (0.2)

[1] 0.614727
```

We can compute the median or quantile from the quantile function in the same way as the univariate case:

```
> #(using conditional model from earlier section)
> #median
> cF.inv (0.5)

[1] 0.2051548

> #lower quartile
> cF.inv (0.25)

[1] 0.01523924
```

Likewise, we can compute the mode using the emode() function in the same way as the univariate case.

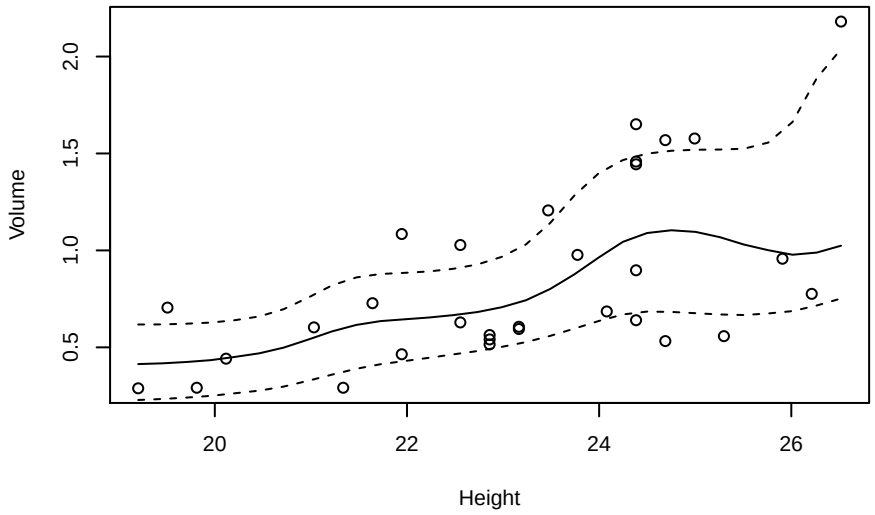
```
> #(again, using conditional model from earlier section)
> mh = emode (cf)
> plot (cf)
> abline (v=mh, lty=2)
> mh

[1] 0.6556134
```



Conditional probabilities and conditional statistics are similar to regression. It's possible to compute conditional probabilities and conditional statistics as functions of one or more variables, which is even more similar to regression. Currently, there's no functions in this package for this purpose. However, it's easy to write a module to do this.

Let's say we want to compute the conditional median and conditional first and third quartiles of Volume, as functions of Height:

```
> x = seq (min (Height), max (Height), length.out=30)
> y = matrix (0, nrow=30, ncol=3)
> for (i in 1:30)
     y [i,] = ecdfc.inverse (Volume, c (Height=x [i]), cbind (Height, Volume),
         npoints=10)(c (0.5, 0.25, 0.75) )

> plot (Height, Volume)
> lines (x, y [,1])
> lines (x, y [,2], lty=2)
> lines (x, y [,3], lty=2)
```



In theory, we should be able to compute the conditional mode in the same way, however, I tried and there are some problems.

Fuzzy Clustering and Weighted Data

Fuzzy clustering computes a membership matrix, with each row corresponding to each data point and each column corresponding to each cluster (not variable). Each row represents the membership of that data point in each cluster as numbers in the interval (0, 1).

The following computes memberships for three clusters and then the weights for the first cluster:

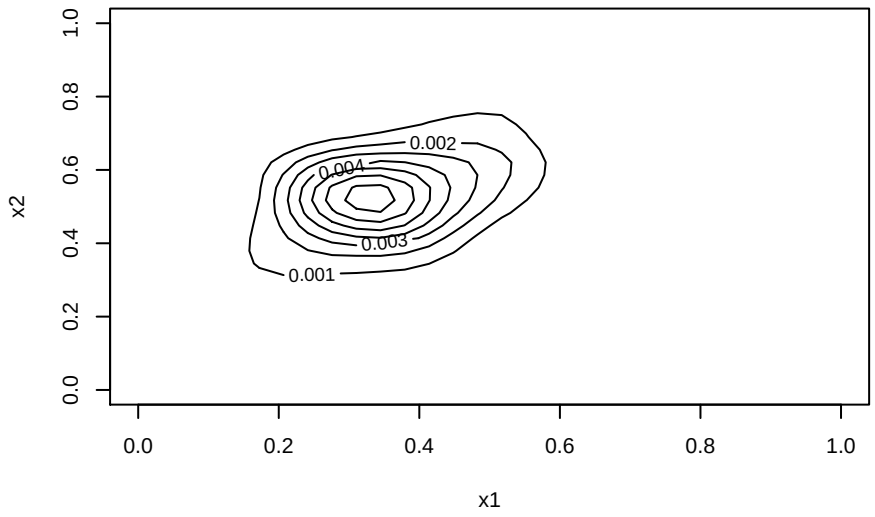
```
> w = FKM.gk (unemployment2, k=3, seed=2)$U [,1]
> w = w / sum (w)
```

Noting that the original dataset contains three variables, however, I'm only using two variables. Also noting that a weighted scatterplot is given in Appendix 6 (at the end of this vignette).

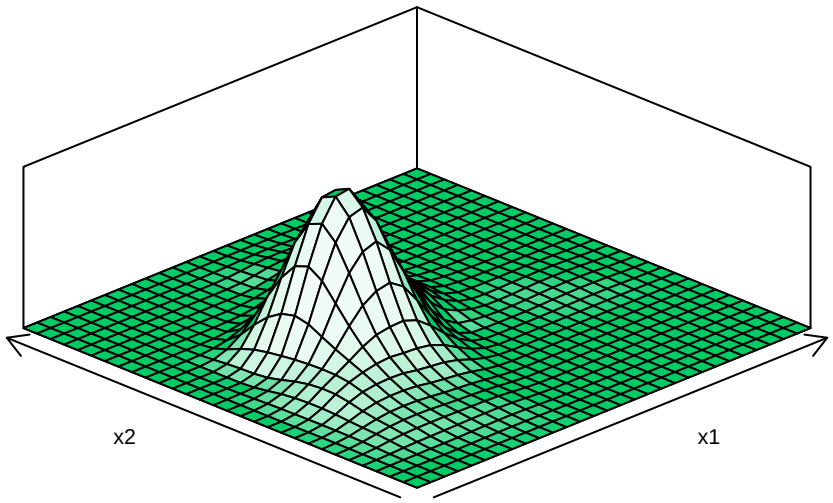
However, fuzzy clustering is limited by itself. I use the term "Membership-Weighted Data Analysis" to refer to the process of modelling the data, weighted according to a membership matrix. I'm going to focus on nonparametric probability distributions, however, there are many ways of modelling such data.

We can compute a weighted bivariate model easily:

```
> f = epdfmv (unemployment2, w=w)
> plot (f)
```

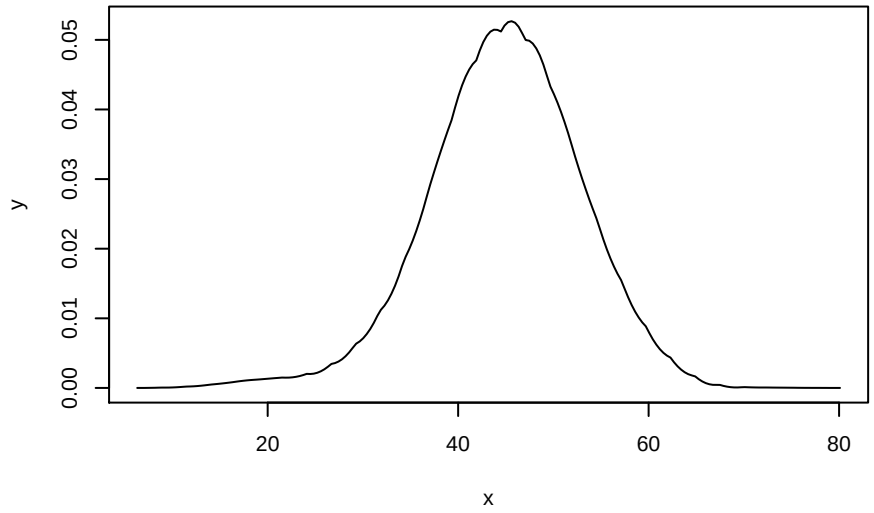



```
> plot (f, TRUE)
```



Likewise, we can compute a weighted conditional model easily:

```
> mean.un.rate = mean (un.rate)
> cf = epdfc (lt.rate, c (un.rate=mean.un.rate), unemployment2, w=w)
> plot (cf)
```



This probability density function is relatively symmetrical, however, this isn't always the case.

todo

I'm planning todo:

- Support categorical variables.
  (So, probability mass functions).
- Improve univariate models, especially EPDFs.
- Improve multivariate restacking formulation.
  (Preferably, implement a hybrid Kernel-Quantile approach, with greater robustness to outliers).
- Support conditional means and variances.
- Investigate conditional modes as functions of other variables.
- Support random number generation.
- Performance optimization.
  (Possibly using C).
- Try to determine optimal smoothing parameters.

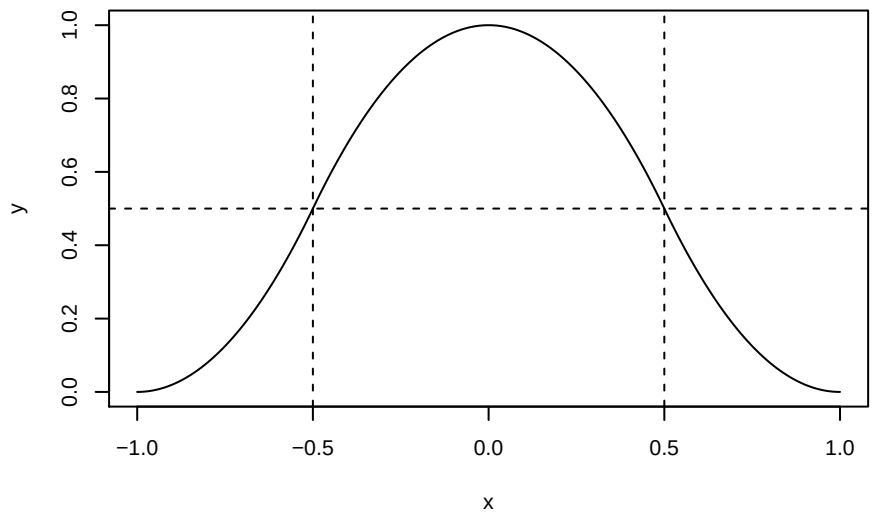
I may do:

- Make some low level functions public.
- Support derivatives of quantile functions.
- Implement some form of multivariate quantile functions, if possible.
- Improve mode computation, including the case for perfectly uniform regions.
- Support conditional statistics as functions.
  (And support some equivalent of partial residual plots).

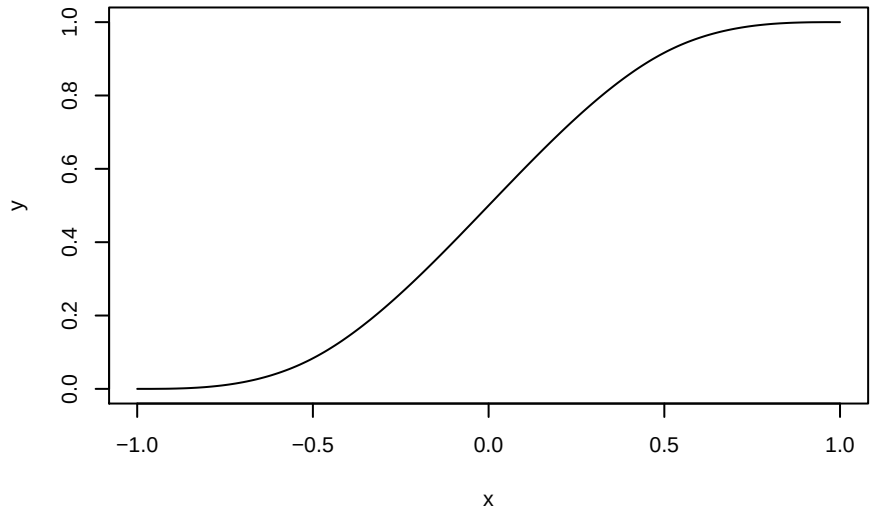
- Implement a preserve = "median" (and IQR) feature.
- Implement plot option to shade regions under EPDFs.
- Implement some form of statistical inference.
- In multivariate models, sort the x attribute.
  (Mainly for consistency with univariate models).

Appendix 1:
Simplified Bell Curves

```
> sbcpdf

function (x)
{
    y = rep(0, length(x))
    I = (x > -1 & x < -0.5)
    y[I] = 2 + 4 * x[I] + 2 * x[I]^2
    I = (x >= -0.5 & x <= 0.5)
    y[I] = 1 - 2 * x[I]^2
    I = (x > 0.5 & x < 1)
    y[I] = 2 - 4 * x[I] + 2 * x[I]^2
    y
}
<bytecode: 0x05b827c8>
<environment: namespace:empirical>

> sbccdf

function (x)
{
    y = rep(0, length(x))
    I = (x > -1 & x < -0.5)
    y[I] = 2/3 + 2 * x[I] + 2 * x[I]^2 + 2/3 * x[I]^3
    I = (x >= -0.5 & x <= 0.5)
    y[I] = 0.5 + x[I] - 2/3 * x[I]^3
    I = (x > 0.5 & x < 1)
    y[I] = 1/3 + 2 * x[I] - 2 * x[I]^2 + 2/3 * x[I]^3
    I = (x >= 1)
    y[I] = 1
    y
}
<bytecode: 0x0636c4e8>
<environment: namespace:empirical>

> x = seq (-1, 1, length.out=200)

> y = sbcpdf (x)
> plot (x, y, type="l")
> abline (v=c (-0.5, 0.5), h=c (0.5), lty=2)
```




```
> y = sbccdf (x)
> plot (x, y, type="l")
```



Appendix 2:
Additional Derandomization Details

The derandomize=TRUE option respaces the data by differencing the data, then transforming the differences, then computing a lowess style smoother on the transformed scale (using simplified bell curves) and then reversing the transformations.

There's n-1 intervals. So, if n is 30 then the number of intervals will be 29. Note that if bind=TRUE then the n value will higher than the original number of data points.

The smoothness (derandomization) parameter is multiplied by the number of intervals to give the nhood (neighborhood size parameter), so:

$$\text{nhood} = \text{drp} * (n - 1)$$

The nhood parameter gives the number of intervals in the smoothing window, assuming that the smoothing window does not fall outside the range of observed data. If it does, then the region outside the range is truncated.

If the nhood parameter is provided then the drp parameter is ignored.

Appendix 3:
Additional Restacking Details

The restacking pdf should be zero centred and symmetrical, with positive densities over the interval (-1, 1). And the restacking cdf should differentiate to a pdf with these properties.

For each variable, the diff.range is equal to the max value less the min value. So, if the min value 5 and the max value is 15, then the diff.range will be 10.

The smoothness (restacking) parameter is multiplied by the diff.range for each variable, to give a (vector) bandwidth parameter, so:

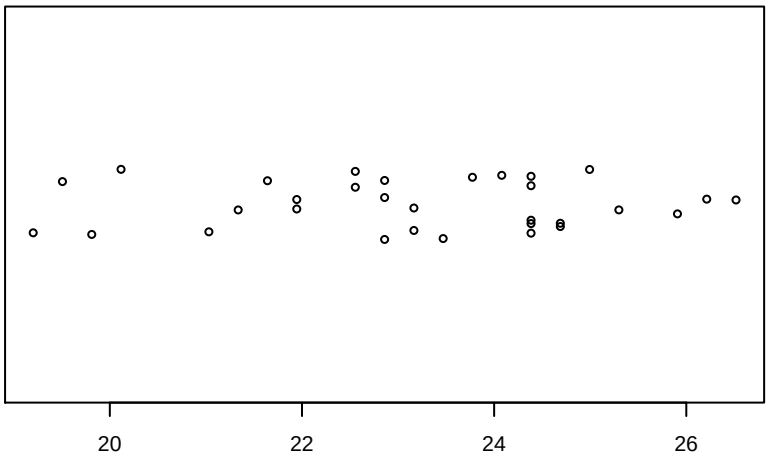
$$\mathrm{bw}_j = \mathrm{rsp} * [\max(x^*_{[,j]}) - \min(x^*_{[,j]})]$$

Like the nhood parameter, the bw parameter defines the width of the smoothing window.

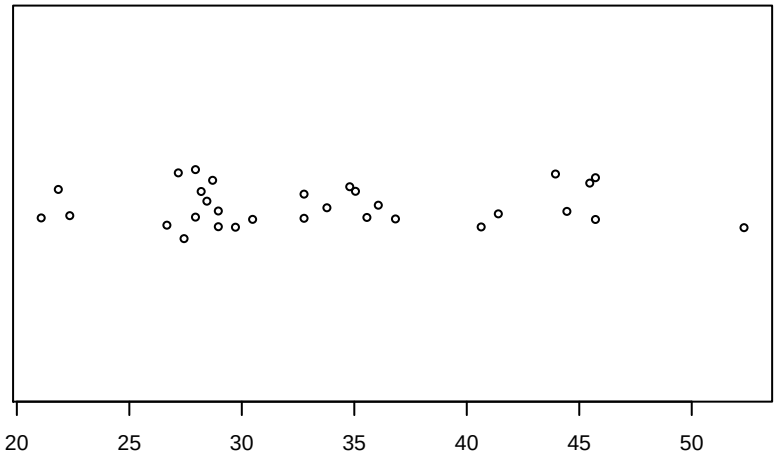
If the bw parameter is provided then the rsp parameter is ignored.

Appendix 4: trees2 Data

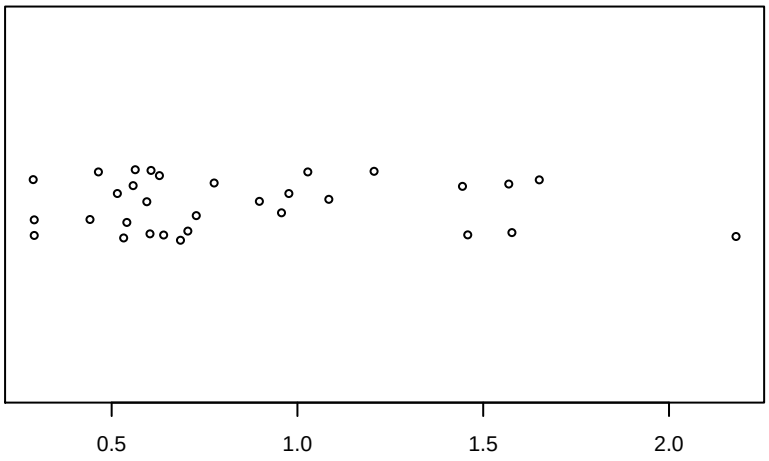
```
> object.info (Height)

value, 31
[1] 21.3360 19.8120 19.2024 21.9456 24.6888 25.2984 20.1168 22.8600
class, 1
[1] "numeric"

> object.info (Girth)

value, 31
[1] 21.082 21.844 22.352 26.670 27.178 27.432 27.940 27.940
class, 1
[1] "numeric"

> object.info (Volume)

value, 31
[1] 0.2916630 0.2916630 0.2888314 0.4643955 0.5323558 0.5578410 0.4417421
[8] 0.5153658
class, 1
[1] "numeric"

> summary (trees2)

     Height          Girth           Volume
 Min.   :19.20   Min.   :21.08   Min.   :0.2888
 1st Qu.:21.95   1st Qu.:28.07   1st Qu.:0.5493
 Median :23.16   Median :32.77   Median :0.6853
 Mean   :23.16   Mean   :33.65   Mean   :0.8543
 3rd Qu.:24.38   3rd Qu.:38.73   3rd Qu.:1.0562
 Max.   :26.52   Max.   :52.32   Max.   :2.1804

> skewness (trees2)

    Height      Girth     Volume
-0.3748690  0.5263163  1.0643575

> stripchart (Height, "jitter", pch=1)
```



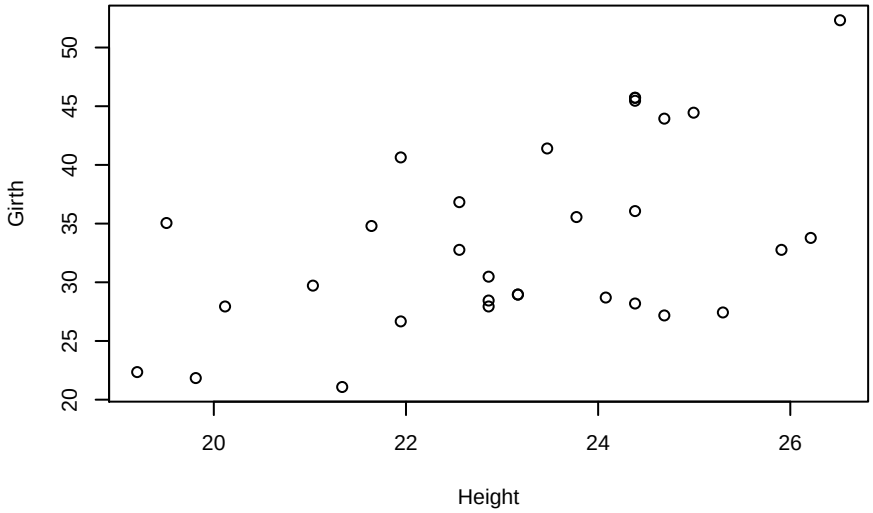

```
> stripchart (Girth, "jitter", pch=1)
```



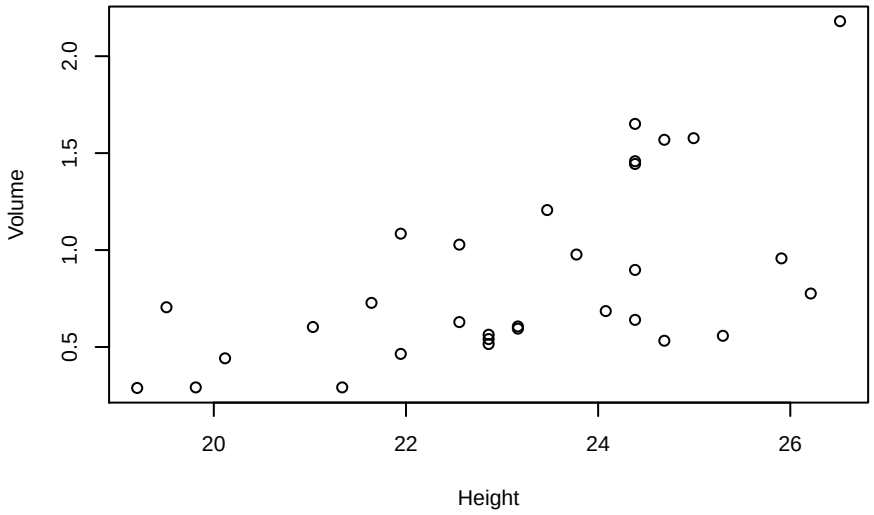

```
> stripchart (Volume, "jitter", pch=1)
```



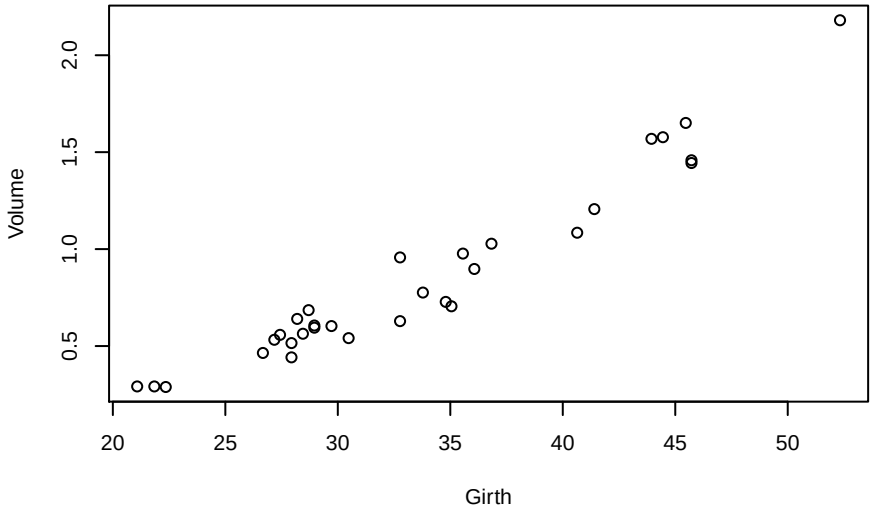

```
> plot (Height, Girth)
```




```
> plot (Height, Volume)
```

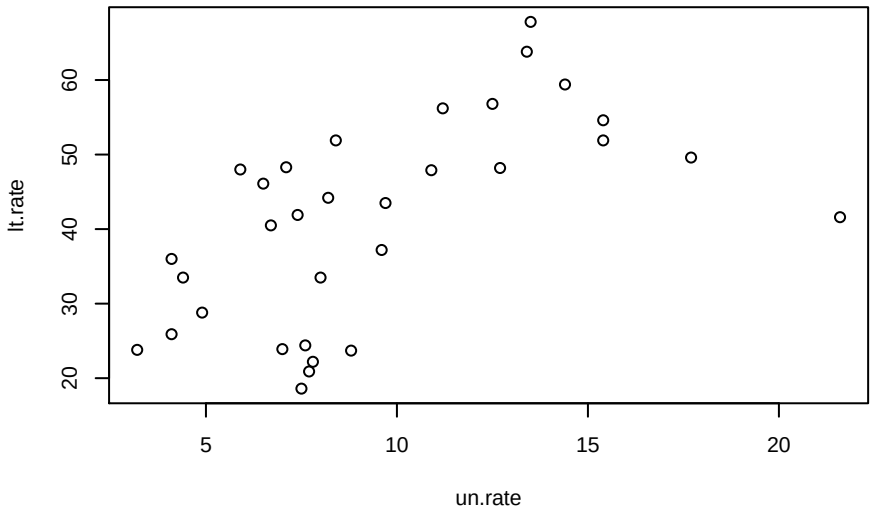



```
> plot (Girth, Volume)
```



Appendix 5: unemployment2 Data

```
> object.info (un.rate)

value, 32
[1]  7.1 11.2  6.7  7.6  5.9 12.5 14.4 17.7
class, 1
[1] "numeric"

> object.info (lt.rate)

value, 32
[1] 48.3 56.2 40.5 24.4 48.0 56.8 59.4 49.6
class, 1
[1] "numeric"

> summary (unemployment2)

    un.rate          lt.rate
 Min.   : 3.200   Min.   :18.60
 1st Qu.: 6.925   1st Qu.:28.07
 Median : 8.100   Median :42.70
 Mean   : 9.478   Mean   :41.08
 3rd Qu.:12.550   3rd Qu.:50.17
 Max.   :21.600   Max.   :67.80

> skewness (unemployment2)

    un.rate     lt.rate
 0.87244093 -0.01797293

> plot (unemployment2)
```



Appendix 6:
Weighted Scatter Plot

```
> s = 1 - w / max (w)
> plot (unemployment2, pch=16, col=rgb (s, s, s) )
```