hyfo Easy Start

Yuanchao Xu

2015-12-06

Contents

Introduction

Official Website is http://yuanchao-xu.github.io/hyfo

hyfo is an R package, initially designed for the European Project EUPORIAS, and cooperated with DHI Denmark, which was then extended to other uses in hydrology, hydraulics and climate.

This package mainly focuses on data process and visulization in hydrology and climate forecasting. Main function includes NetCDF file processing, data extraction, data downscaling, data resampling, gap filler of precipitation, bias correction of forecasting data, flexible time series plot, and spatial map generation. It is a good pre-processing and post-processing tool for hydrological and hydraulic modellers.

If you feel hyfo is of a little help, please cite it as following:

Xu, Yuanchao(2015). hyfo: Hydrology and Climate Forecasting R Package for Data Analysis and Visualization. Retrieved from http://yuanchao-xu.github.io/hyfo/

Author in this corner

TIPS

- For the hydrology tools part, the minimum time unit is a day, i.e., it mainly focuses on water resource and some long term analysis. For flood analysis part, it will be added in future.
- One important characteristic by which hyfo can be distinguished from others is its convenience in multiple plots and series plots. Most data visualization tool in hyfo provides the output that can be directly re-plot by `ggplot2`, if `output = 'ggplot'` is assigned in the argument of the function, which will be easier for the users to generated series/multiple plots afterwards. When `output = 'ggplot'` is selected, you also have to assigne a `name = 'yourname'` in the argument, for the convenience of generating multiplots in future. All the functions ending with `_comb` can generated series/multiple plots, details can be found in the user mannual.
- For the forecasting tools part, `hyfo` mainly focuses on the post processing of the gridData derived from forecasts or other sources. The input is a list file, usually an NetCDF file. There are `getNcdfVar()`, `loadNcdf()` and `writeNcdf()` prepared in hyfo, for you to deal with NetCDF file.
- If you don't like the tile, x axis, y axis of the plot, just set them as '', e.g. `title = ''`
- For R beginners, R provides different functions to write to file. `write.table` is a popular choice, and after write the results to a file, you can directly copy paste to your model or to other uses.
- The functions end with `_anarbe` are the functions designed specially for some case in Spain, those functions mostly are about data collection of the anarbe catchment, which will be introduced in the end of this mannual.

Installation

- Released version from CRAN, for beginners and normal users.

```
install.packages("hyfo")
```

- Development version from github, for experienced users and those who are interested in investigating.

```
install.packages('devtools')
# Ignore the warning that Rtool is not installed, unless you want other #function from devtools.
# If you have "devtools" installed already, you just need to run the following code.
devtools::install_github('Yuanchao-Xu/hyfo')
```

During the installation of the development version, if there is some error, you can just follow the error message and reinstall the package with error. The most common message is

```
cannot remove previously installed XXX package.
or
error in installation of XXX package.
```

If so, just use `install.pakcages('xxx')` to reinstall XXX package. And then reinstall hyfo again. Other errors can be solved by directly reinsalling hyfo.

- You can also go here to download installation file, and use IDE like Rstudio to install from file, both `tar.gz` and `zip` formats are provided.

1. Hydrology

Note If you are an experienced R user, and know how to read data in R, deal with dataframe, generate date and list, please start from next charpter, "1.2 Rainfall Analysis"

1.1 Start from Raw Data

1.1.1 From File

`hyfo` does provide a common tool for collecting data from different type of files, including "txt", "csv" and "excel", which has to be assigned to the argument `fileType`.

Now let's use internal data as an example.

```
library(hyfo)#load the package.
# get the folder containing different csv (or other type) files.
file <- system.file("extdata", "1999.csv", package = "hyfo")
folder <- strsplit(file, '1999')[[1]][1]

# Extract and combine content from different files and in each file,
# the extracted zone is from row 10 to row 20, Column 1 to column2.
a <- collectData(folder, fileType = 'csv', range = c(10, 20, 1, 2))
```

```
## All the files in the folder should have the same format
```

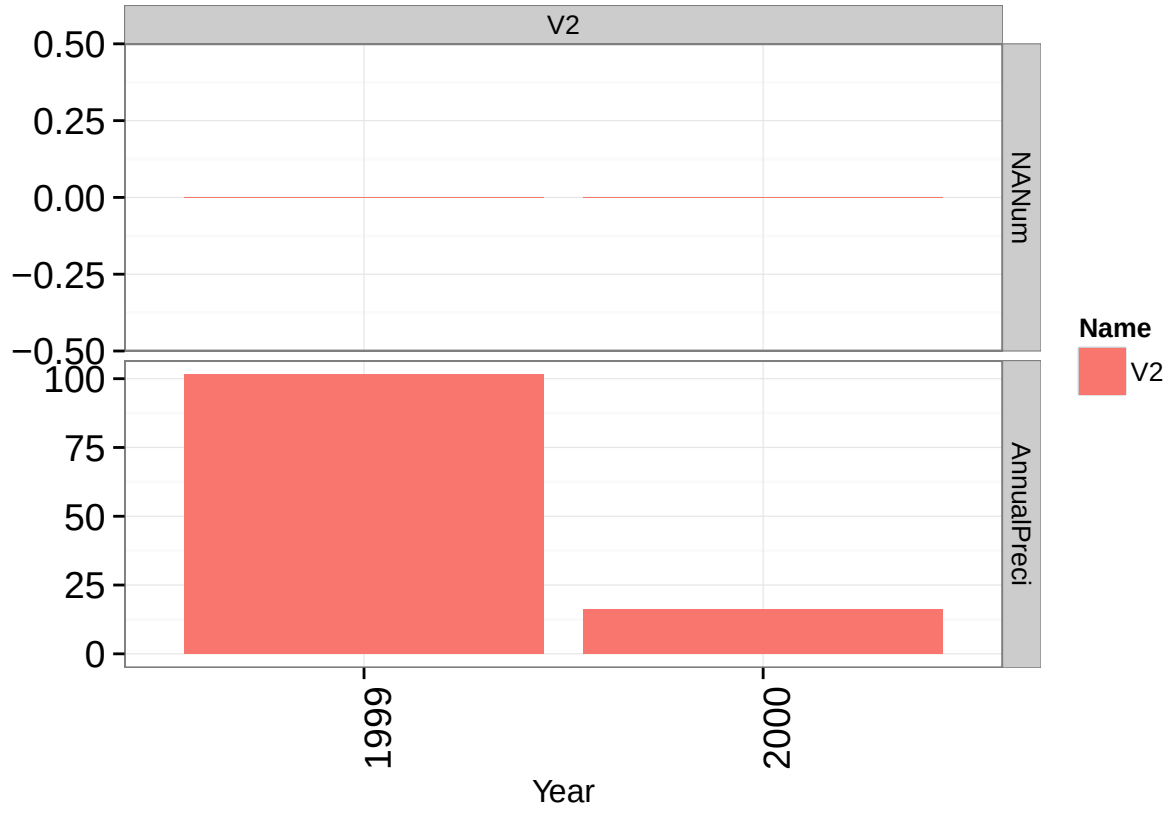
```
str(a)
```

```
## 'data.frame':    22 obs. of  2 variables:
##  $ V1: Factor w/ 722 levels "","01/02/1999",..: 57 69 81 93 105 117 129 141 153 165 ...
##  $ V2: num  0 0 19.7 42.9 4.7 14.5 2 10.9 5.6 0 ...
```

a cannot be directly inputed in `hyfo`, it still needs some process.

```
# Check the date to see if it follows the format in ?as.Date(), if not,
# use as.Date to convert.
a <- data.frame(a)
#get date
date <- a[, 1]

# The original format is d/m/year, convert to formal format.
date <- as.Date(date, format = '%d/%m/%Y')
a[, 1] <- date

# Now a has become `a` time series dataframe, which is the atom element of the analysis.
#`hyfo` deals with list containing different time series dataframe. In this example,
#there is only one dataframe, and more examples please refer to the following chapter.
datalist <- list(a)

# Use getAnnual as an example, here since `a` is not a complete time series,
# the result is only base on the input.
# getAnnual gives the annual precipitation of each year,
# and will be introduced in the next chapter.
getAnnual(datalist)
```

```
## Using Year, Name as id variables
```



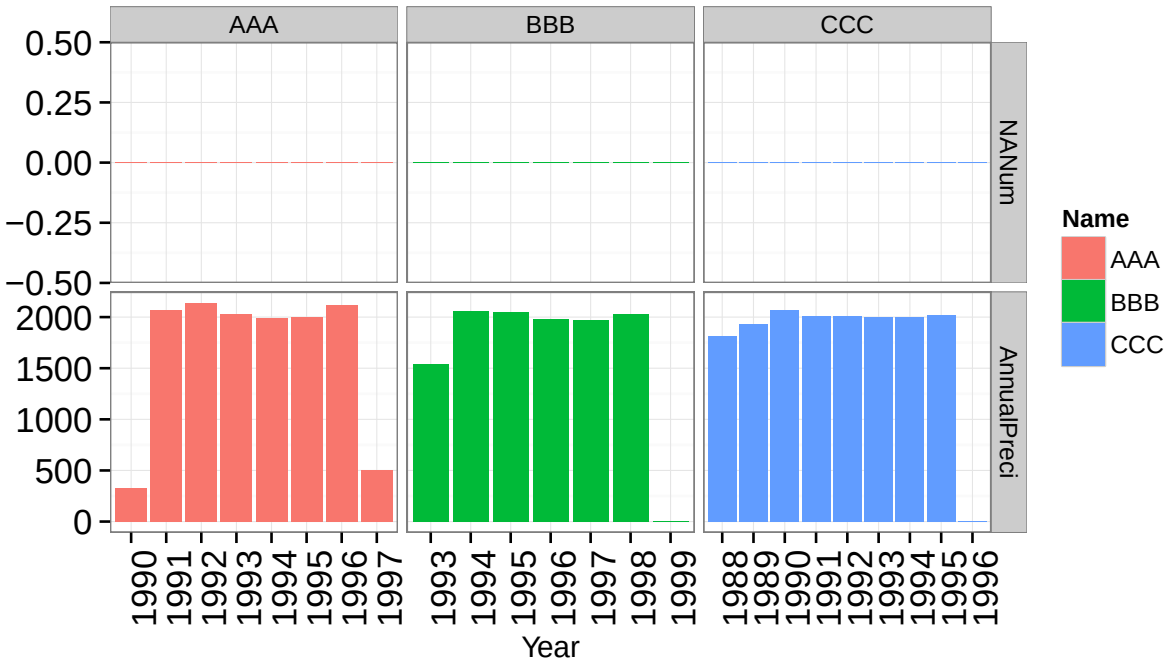
```
##   Year Name AnnualPreci recordNum NANum
## 1 1999   V2       101.5        11     0
## 2 2000   V2        16.0        11     0
```

1.1.2 Mannually

Following example shows a simple way to generate dataframe with start date, end date, and the value. Here in the example, `sample()` is used to generate random values, while in real case it will be a vector containing time series values.

```
# Generate timeseries datalist. Each data frame consists of a Date and a value.
library(hyfo)
AAA <- data.frame(
  Date = seq(as.Date('1990-10-28'), as.Date('1997-4-1'), 1), # Date column
  AAA = sample(1:10, length(seq(as.Date('1990-10-28'), # value column
                                as.Date('1997-4-1'), 1)), repl = TRUE))

BBB <- data.frame(
  Date = seq(as.Date('1993-3-28'), as.Date('1999-1-1'),1),
  BBB = sample(1:10, length(seq(as.Date('1993-3-28'),
                                as.Date('1999-1-1'),1)), repl = TRUE))

CCC <- data.frame(
  Date = seq(as.Date('1988-2-2'), as.Date('1996-1-1'),1),
  CCC = sample(1:10, length(seq(as.Date('1988-2-2'),
                                as.Date('1996-1-1'),1)), repl = TRUE))

datalist <- list(AAA, BBB, CCC)# dput() and dget() can be used to save and load list file.
a <- getAnnual(datalist)
```

```
## Using Year, Name as id variables
```



1.2 Raw Data Analysis

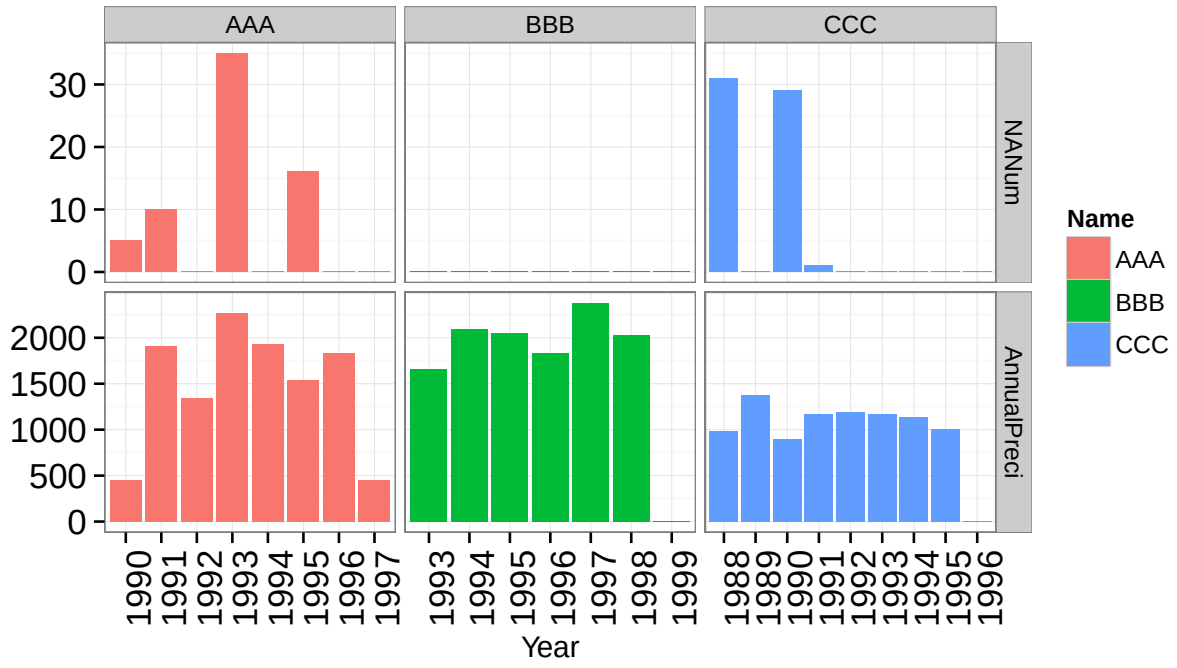
After having the raw data, usually we need to have an overview of the rainfall in order to further process the data, `getAnnual` can provide the information based on annual rainfall for all the input time series.

hyfo also provides time series plot `plotTS` and `plotTS_comb`, for you to plot single time series or multiple time series. And missing values will also be shown in the plot.

Assuming we have three gauging stations named "AAA", "BBB", "CCC", the precipitation information can be get by the following:

```
# testdl is a datalist provided by the package as a test.
# It's a list containing different time series.
data(testdl)
a <- getAnnual(testdl)
```

```
## Using Year, Name as id variables
```



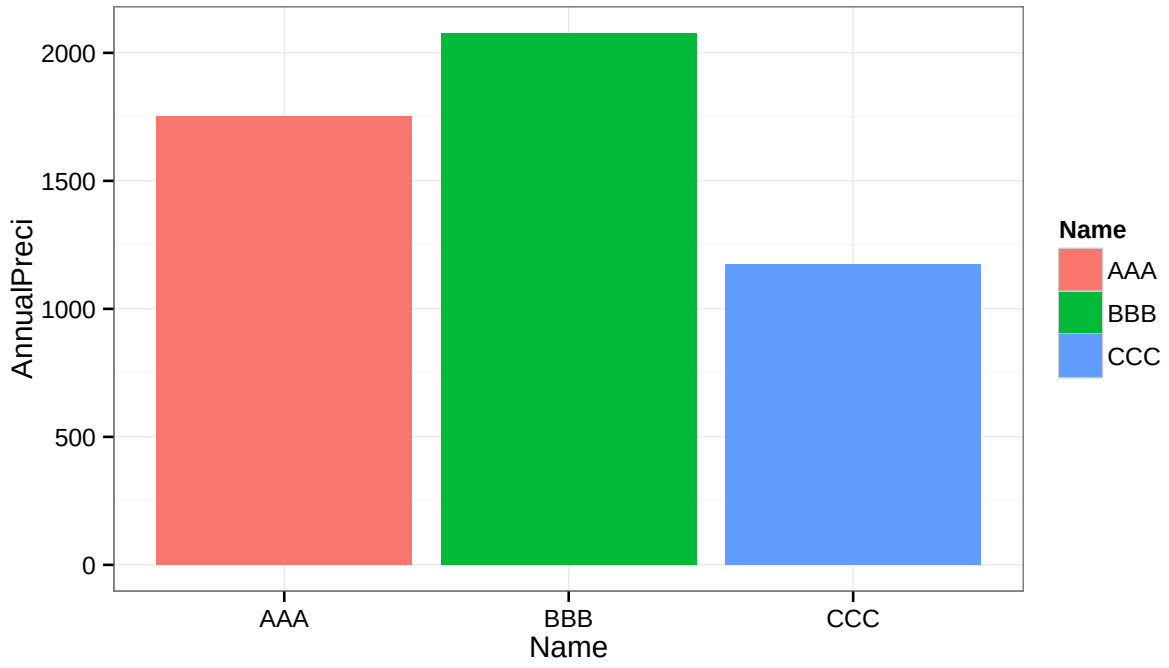
As shown above, the annual precipitation and the number of missing values are shown in the figure. Knowing how many missing values you have is alway important when calculating the mean annual precipitation.

Now we want to get the mean annual precipitation.

```
a <- getAnnual(testdl, output = 'mean')
a
```

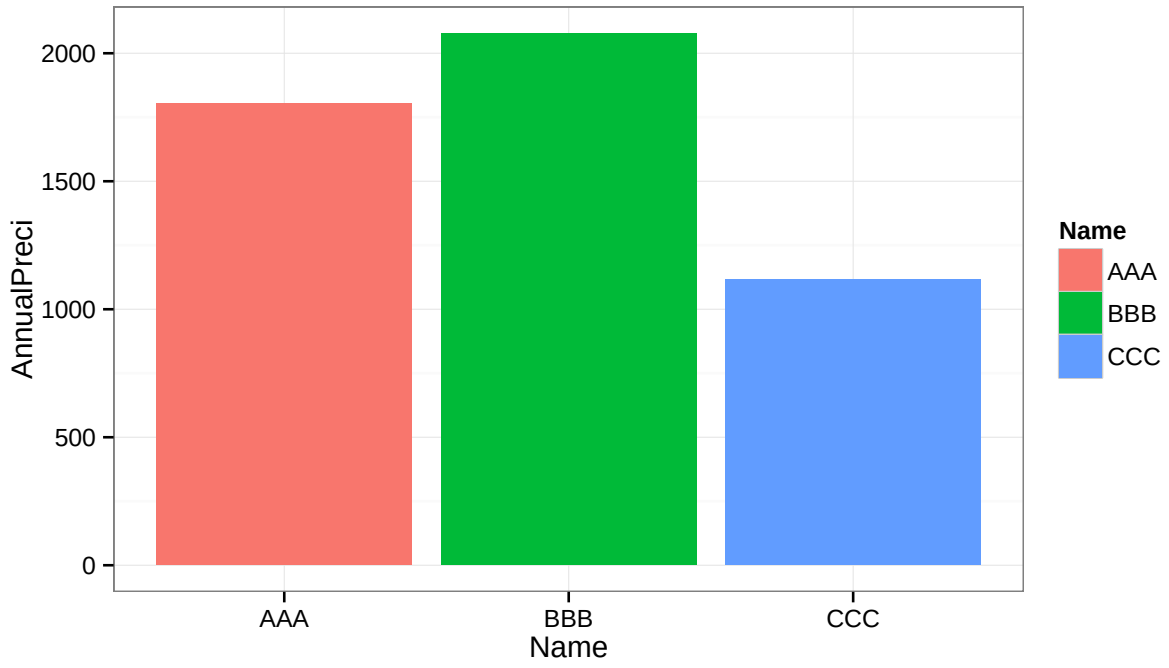
```
##    Year Name AnnualPreci recordNum NANum
## 1  1990  AAA     446.772        60     5
## 2  1991  AAA    1913.661       355    10
## 3  1992  AAA    1340.688       366     0
## 4  1993  AAA    2270.130       330    35
## 5  1994  AAA    1927.704       365     0
## 6  1995  AAA    1543.893       349    16
## 7  1996  AAA    1828.845       366     0
## 8  1997  AAA     454.863        91     0
## 9  1993  BBB    1657.080       279     0
## 10 1994  BBB    2090.970       365     0
## 11 1995  BBB    2056.230       365     0
## 12 1996  BBB    1838.340       366     0
## 13 1997  BBB    2380.500       365     0
## 14 1998  BBB    2024.910       365     0
## 15 1999  BBB       0.090         1     0
## 16 1988  CCC     985.200       303    31
## 17 1989  CCC    1375.020       365     0
## 18 1990  CCC     894.840       336    29
## 19 1991  CCC    1171.380       364     1
## 20 1992  CCC    1190.280       366     0
## 21 1993  CCC    1164.660       365     0
## 22 1994  CCC    1139.640       365     0
```

```
## 23 1995  CCC    1006.260       365     0
## 24 1996  CCC       0.000         1     0
```



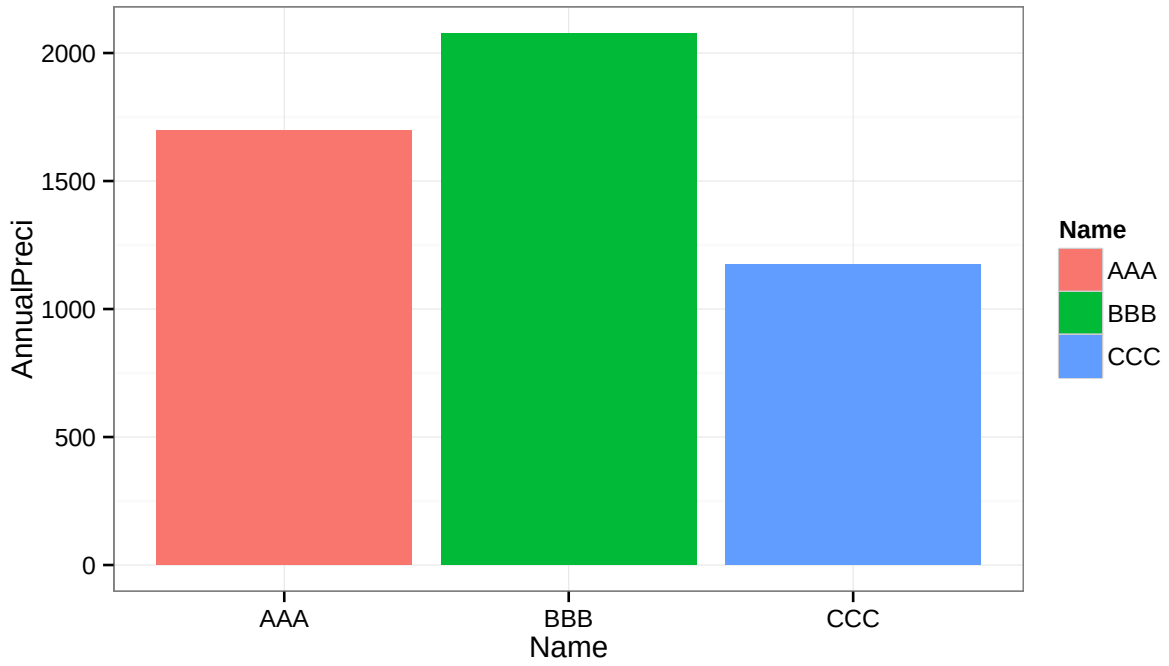
Mean annual precipitation is calculated, but as we can see in the figure before, it's not reliable, since there are a lot of missing values in AAA and CCC, especially in AAA, in 1993, there are more than 30 missing values in a year. So we have to decide which is the threshold for the valid record. the default is 355, which means in a year (355 or 365 days), if the valid records (not missing) exceeds 355, then this year is taken into consideration in the mean annual preicipitation calculation.

```
getAnnual(testdl, output = 'mean', minRecords = 300)
```




```
##    Year Name AnnualPreci recordNum NANum
## 1  1990  AAA     446.772        60     5
## 2  1991  AAA    1913.661       355    10
## 3  1992  AAA    1340.688       366     0
## 4  1993  AAA    2270.130       330    35
## 5  1994  AAA    1927.704       365     0
## 6  1995  AAA    1543.893       349    16
## 7  1996  AAA    1828.845       366     0
## 8  1997  AAA     454.863        91     0
## 9  1993  BBB    1657.080       279     0
## 10 1994  BBB    2090.970       365     0
## 11 1995  BBB    2056.230       365     0
## 12 1996  BBB    1838.340       366     0
## 13 1997  BBB    2380.500       365     0
## 14 1998  BBB    2024.910       365     0
## 15 1999  BBB       0.090         1     0
## 16 1988  CCC     985.200       303    31
## 17 1989  CCC    1375.020       365     0
## 18 1990  CCC     894.840       336    29
## 19 1991  CCC    1171.380       364     1
## 20 1992  CCC    1190.280       366     0
## 21 1993  CCC    1164.660       365     0
## 22 1994  CCC    1139.640       365     0
## 23 1995  CCC    1006.260       365     0
## 24 1996  CCC       0.000         1     0
```

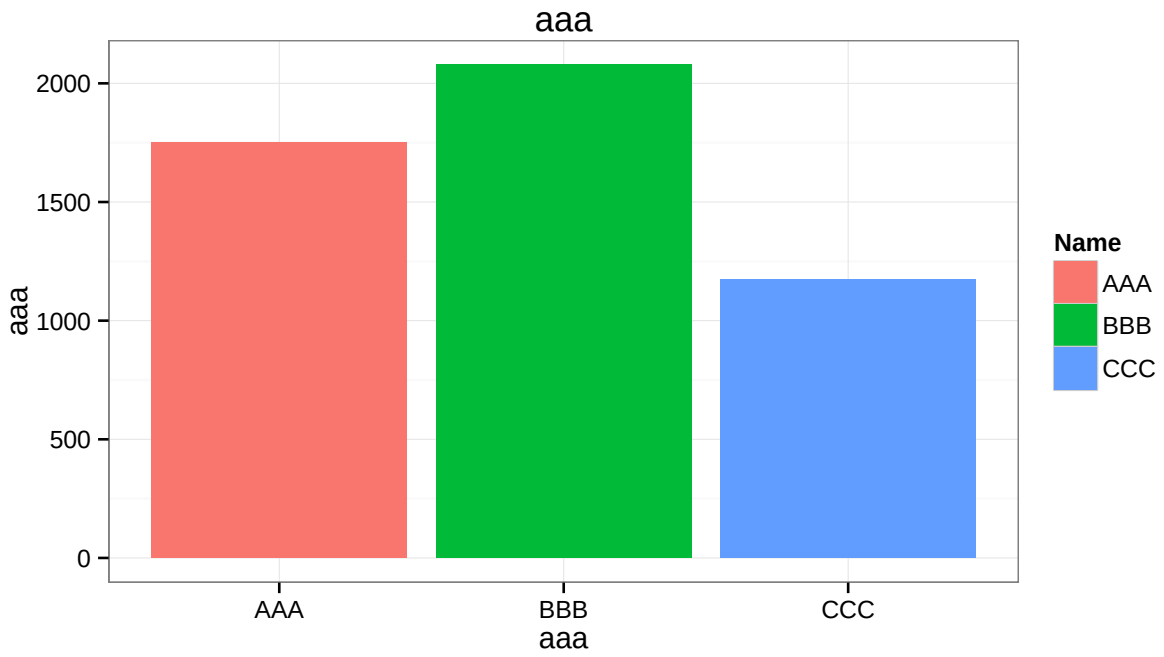
```
getAnnual(testdl, output = 'mean', minRecords = 365)
```




```
##    Year Name AnnualPreci recordNum NANum
## 1  1990  AAA     446.772        60     5
## 2  1991  AAA    1913.661       355    10
## 3  1992  AAA    1340.688       366     0
## 4  1993  AAA    2270.130       330    35
## 5  1994  AAA    1927.704       365     0
## 6  1995  AAA    1543.893       349    16
## 7  1996  AAA    1828.845       366     0
## 8  1997  AAA     454.863        91     0
## 9  1993  BBB    1657.080       279     0
## 10 1994  BBB    2090.970       365     0
## 11 1995  BBB    2056.230       365     0
## 12 1996  BBB    1838.340       366     0
## 13 1997  BBB    2380.500       365     0
## 14 1998  BBB    2024.910       365     0
## 15 1999  BBB       0.090         1     0
## 16 1988  CCC     985.200       303    31
## 17 1989  CCC    1375.020       365     0
## 18 1990  CCC     894.840       336    29
## 19 1991  CCC    1171.380       364     1
## 20 1992  CCC    1190.280       366     0
## 21 1993  CCC    1164.660       365     0
## 22 1994  CCC    1139.640       365     0
## 23 1995  CCC    1006.260       365     0
## 24 1996  CCC       0.000         1     0
```

If you are not satisfied with the title and x axis and y axis, you can assign them yourself.

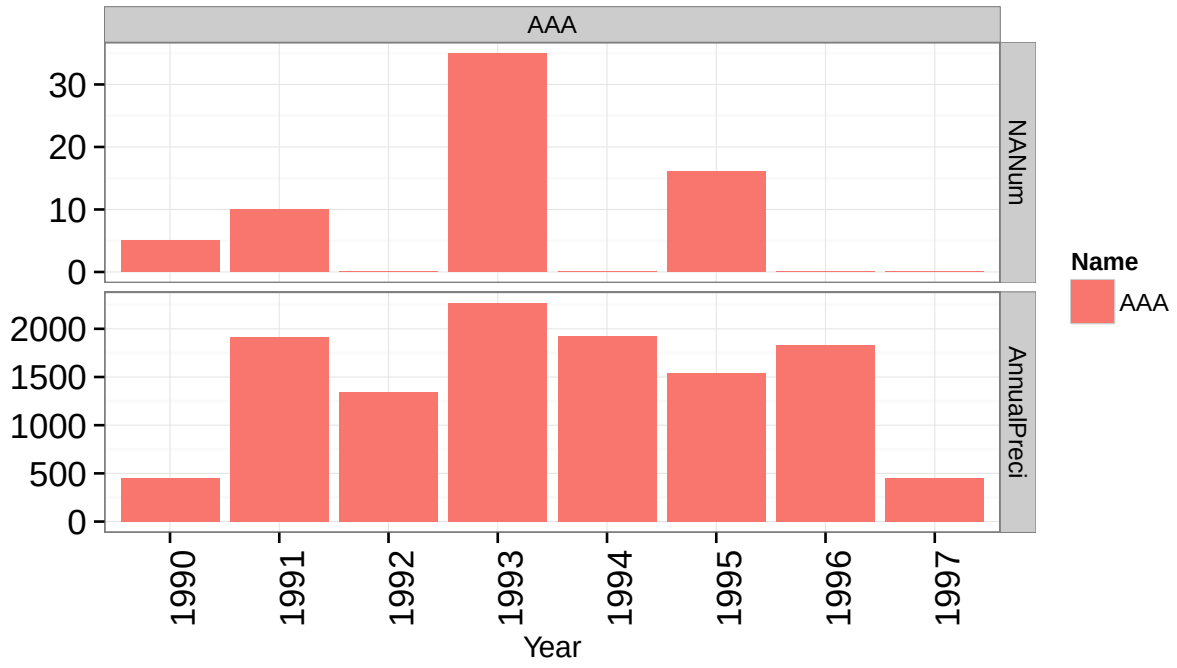
```
a <- getAnnual(testdl, output = 'mean', title = 'aaa', x = 'aaa', y = 'aaa')
```



If you want to calculate annual rainfall for a single dataframe containing one time series. You can use the argument `dataframe =`. NOTE, if you don't put `dataframe =`, hyfo may take it as a list, which will give an error.

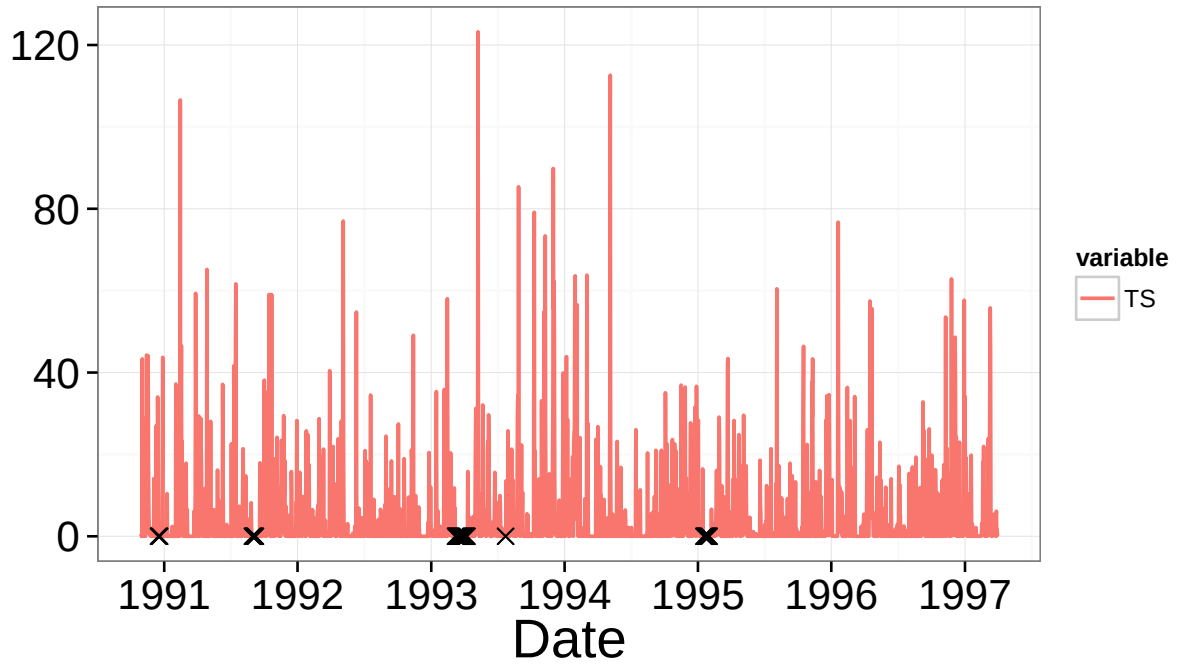
```
a <- getAnnual(testdl[[1]])
```

```
## Using Year, Name as id variables
```



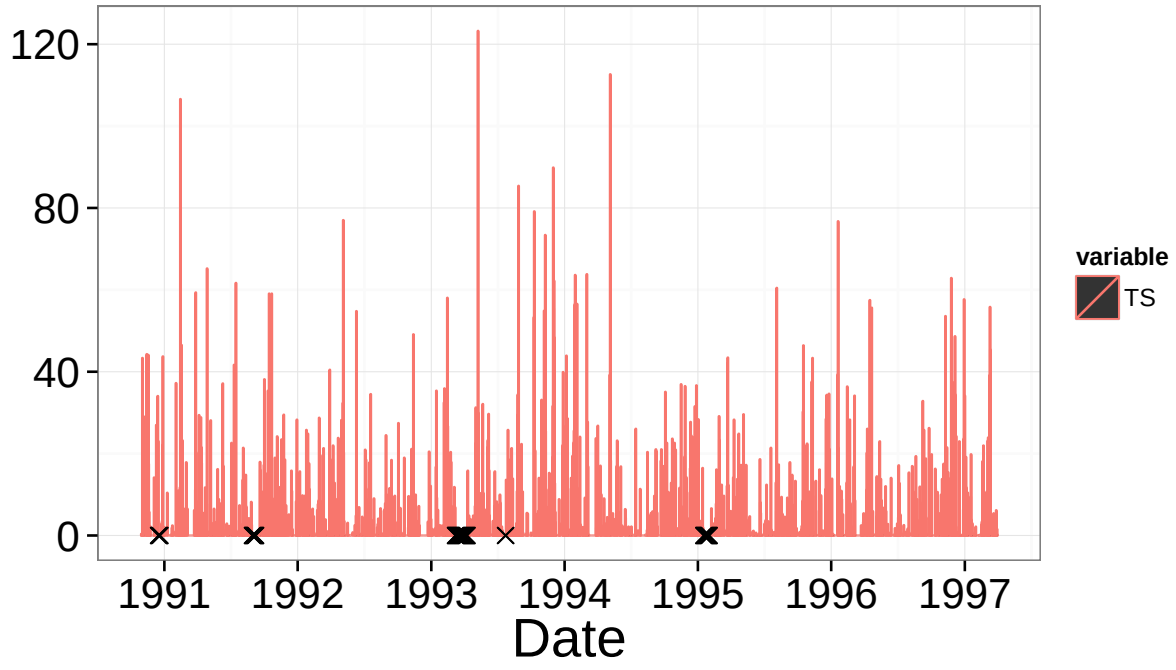
`plotTS` is a powerfull plotting tool for you to plot time series, it can plot different kinds of time series. Sometimes when you finish processing the data, it's very convenient to use `plotTS` to see your results. And also you can use `plotTS_comb` to generate multiple time series plots

```
a1 <- plotTS(TS = testdl[[1]])
```

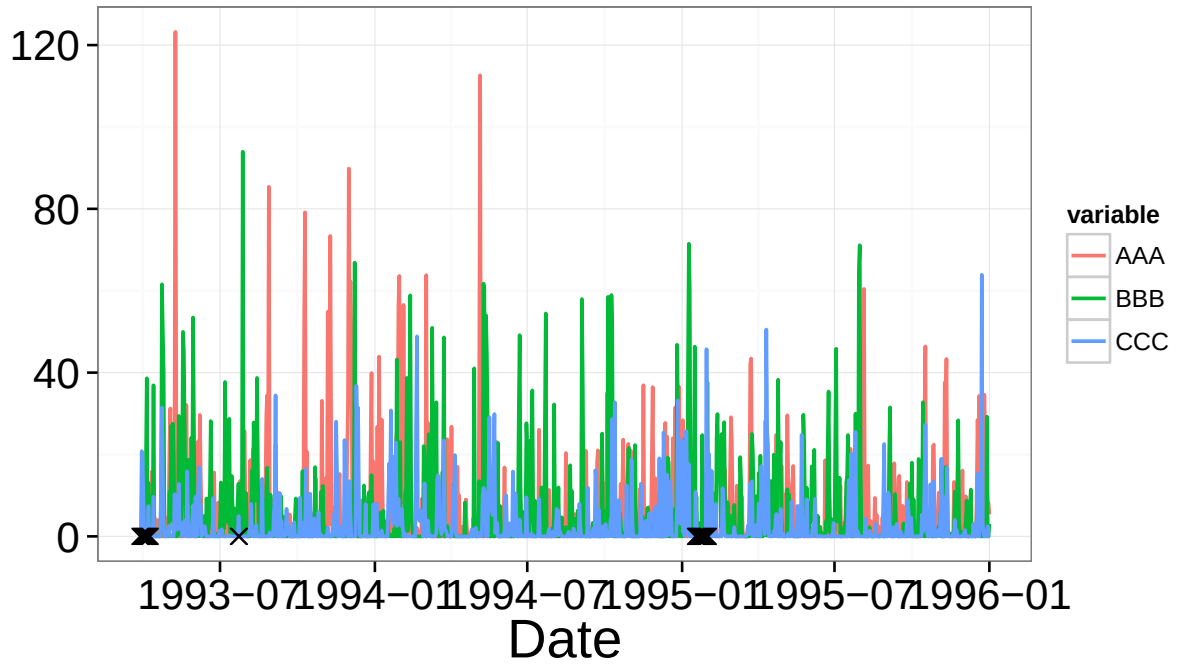


```
# You can also choose 'bar' as time series type, default is 'line'. But most of time,
# they are not # so different.
a2 <- plotTS(TS = testdl[[1]], type = 'bar')
```

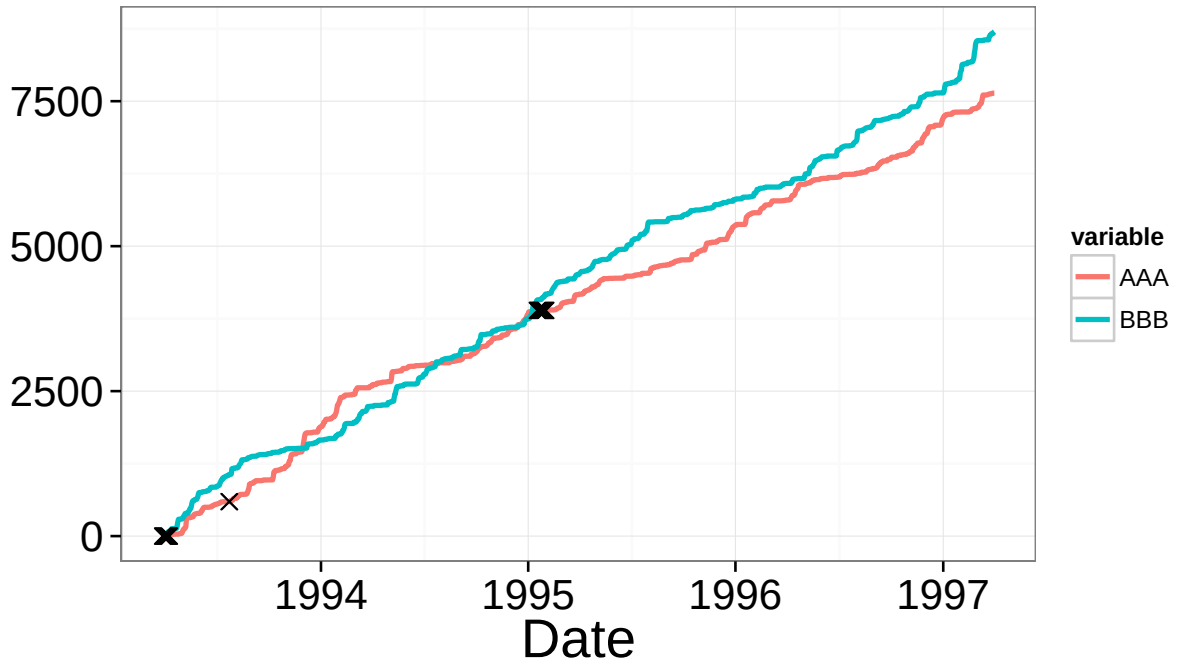
```
## Warning: position_stack requires constant width: output may be incorrect
```




```
# If input is a datalist
plotTS(list = testdl)
```

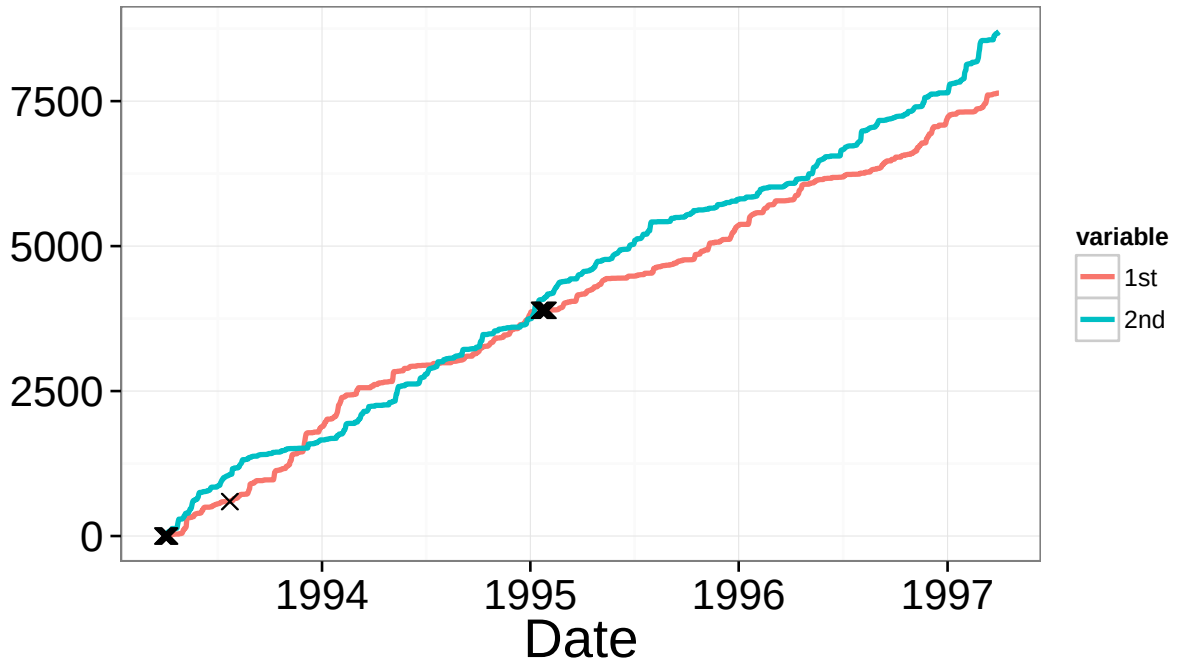



```
# Or if you want to input time series one by one
# if plot = 'cum', then cumulative curve will be plotted.
plotTS(testdl[[1]], testdl[[2]], plot = 'cum')
```

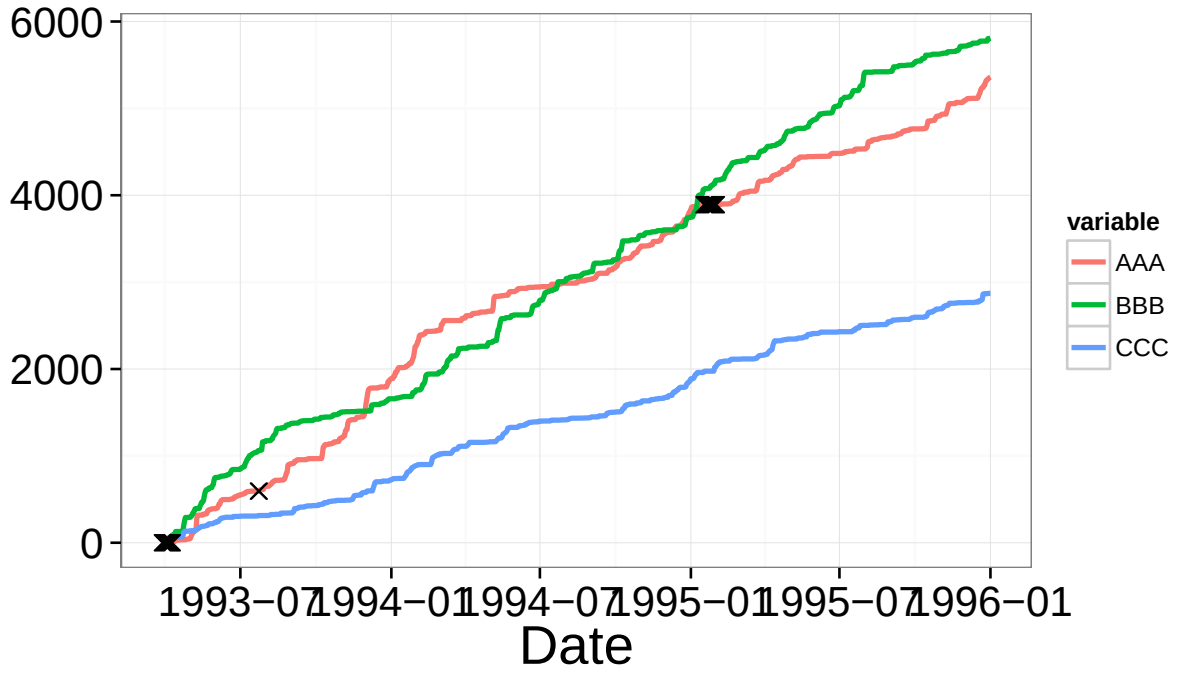



```
# If you want to assign your own name.You can name the list first
TSlist <- list(testdl[[1]], testdl[[2]])
```

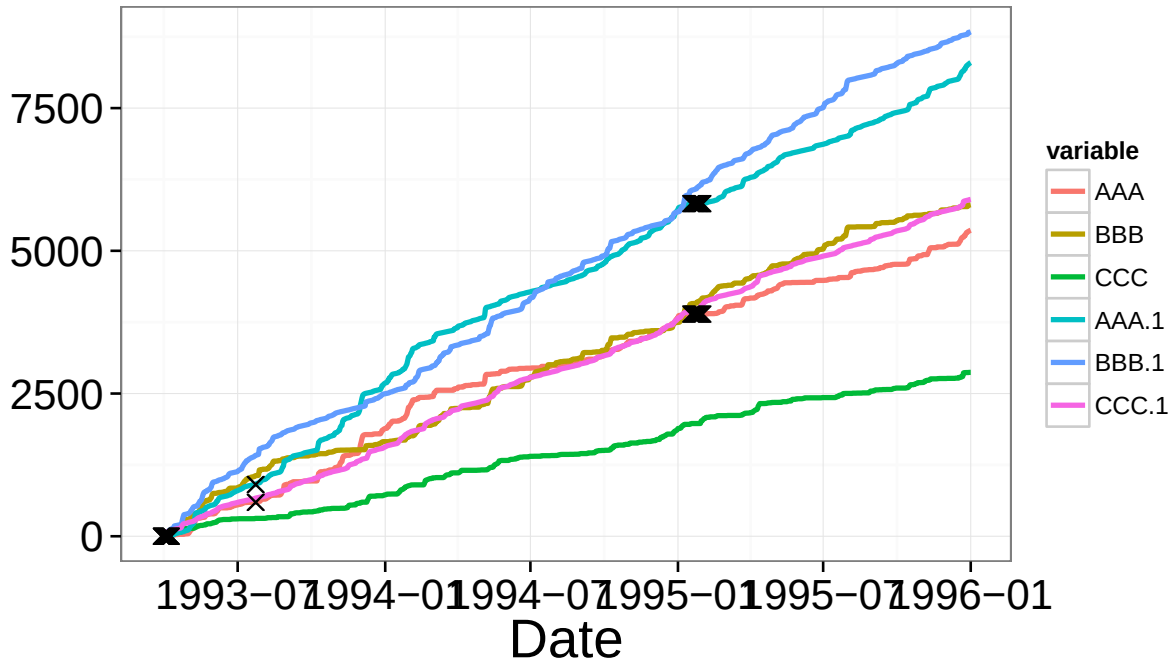
```r
names(TSlist) <- c('1st', '2nd')
plotTS(list = TSlist, plot = 'cum')
```




```r
# You can also directly plot multicolumn dataframe
dataframe <- list2Dataframe(extractPeriod(testdl, commonPeriod = TRUE))
plotTS(dataframe, plot = 'cum')
```



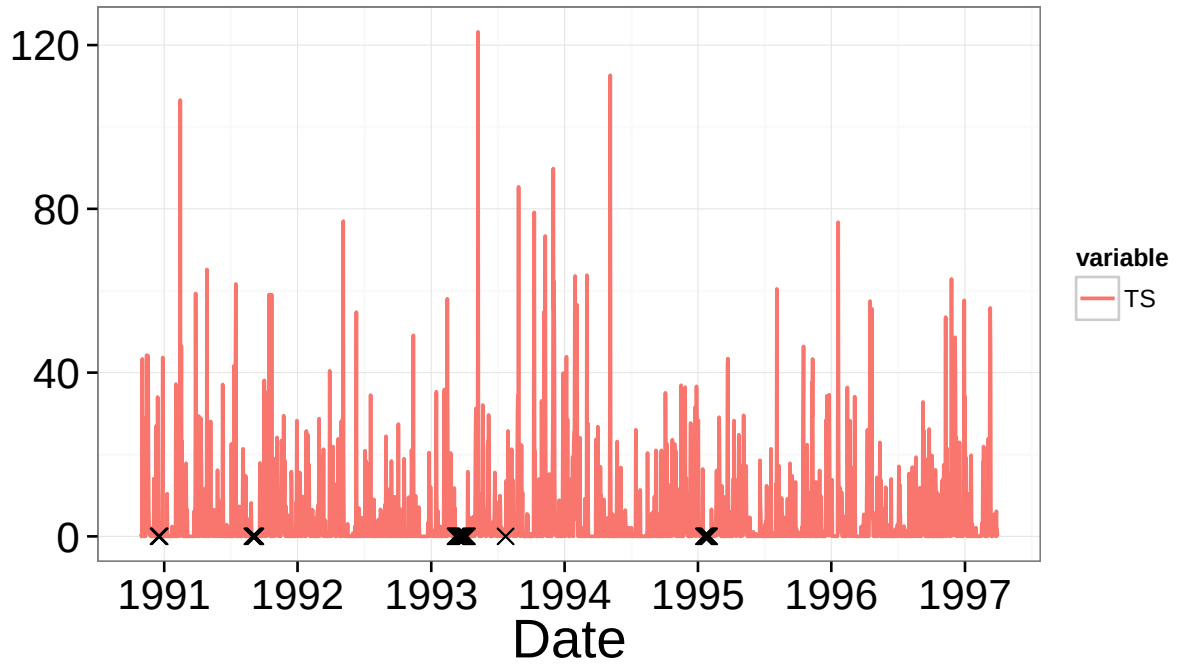
```r
# Sometimes you may want to process the dataframe and compare with the original one
dataframe1 <- dataframe
dataframe1[, 2:4] <- dataframe1[, 2:4] + 3
plotTS(dataframe, dataframe1, plot = 'cum')
```



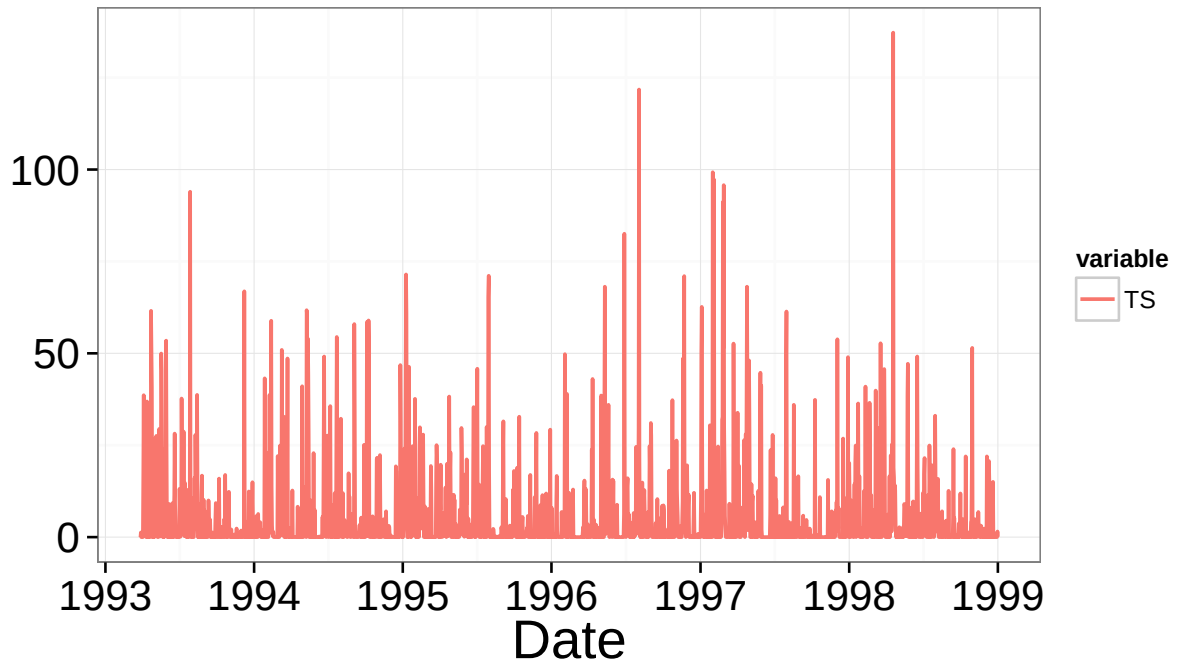

```r
# But note, if your input is a multi column dataframe, it's better to plot one using plotTS,
# and compare them using plotTS_comb. If all data are in one plot, there might be too messy.
```

And also you can use `plotTS_comb` to generate multiple time series plots.

```r
# To use comb function, you have to change output type to 'ggplot'

a1 <- plotTS(TS = testdl[[1]], output = 'ggplot', name = 1)
```

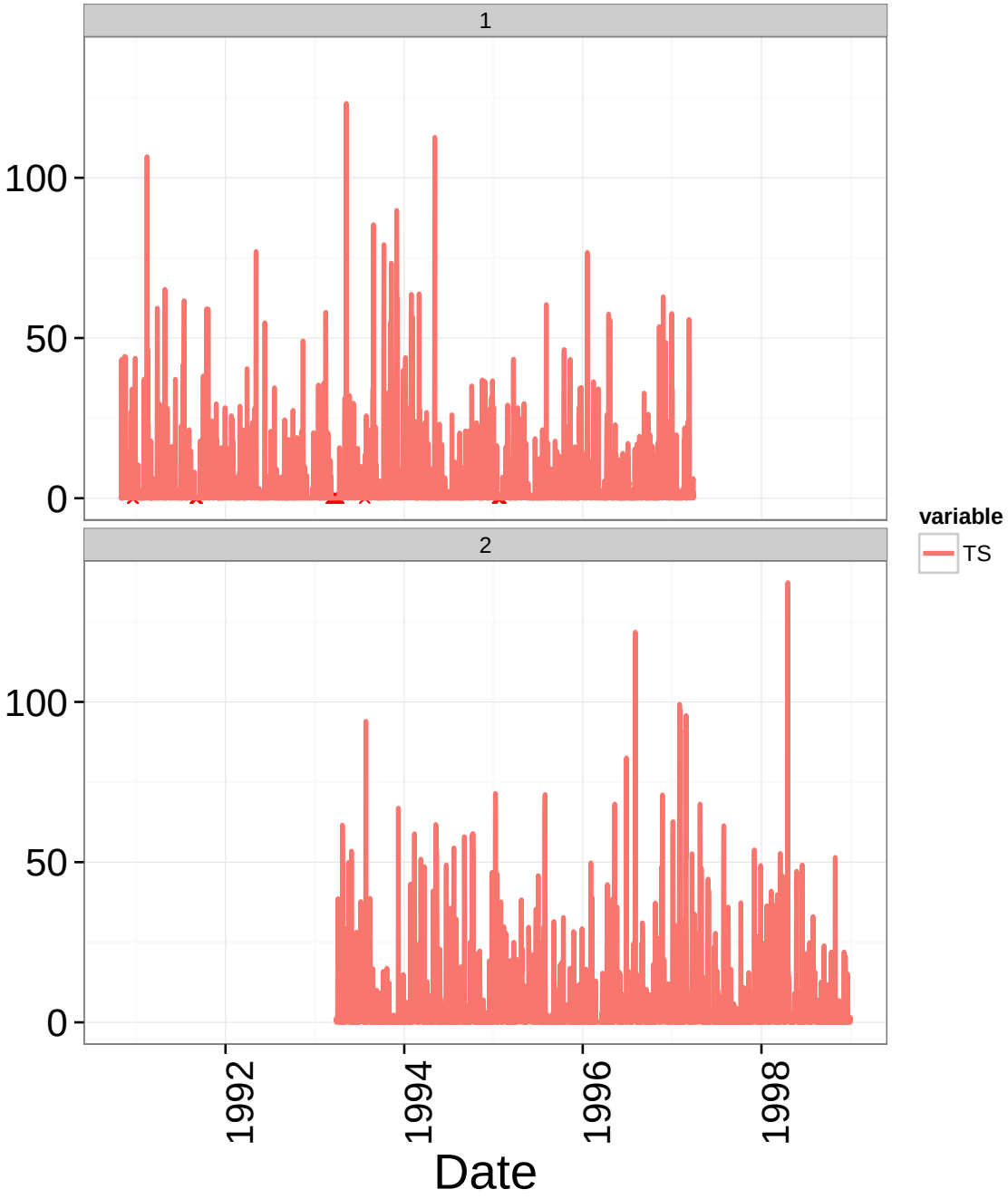



```
a2 <- plotTS(TS = testdl[[2]], output = 'ggplot', name = 2)
```




```
plotTS_comb(a1, a2, nrow = 2)
```

```
## Check if the data list is available for rbind or cbind...
##
## Data list is OK
```



1.3 Further Process for Model Input

1.3.1 Extract Certain Period or Months from Different Time Series

Now we have the general information of the precipitation, if we want to use them in a model, we have to extract the common period of them, and use the common period precipitation to analyze.

```
testdl_new <- extractPeriod(testdl, commonPeriod = TRUE )
str(testdl_new)
```

If we want to extract data from a certain period, we can assgin start and end date.

```
# Extract period of the winter of 1994
testdl_new <- extractPeriod(testdl, startDate = '1994-12-01', endDate = '1995-03-01' )
str(testdl_new)
```

Above is for us to extract period from different datalist, if we have a single time series, and we want to extract certain period from the single time series. We can make a small change to the argument : add `TS =`, a single time series can contain more than 1 column of value, e.g. the result from `list2dataframe`.

```
# First change the list from above process to dataframe
dataframe <- list2Dataframe(testdl_new)
# now we have a dataframe to extract certain period.
dataframe <- extractPeriod(dataframe, startDate = '1994-12-01', endDate = '1995-03-01')
str(testdl_new)
```

For some cases, certain months and years data needs to be extracted from a continuous time sereis (or dataframe), e.g., as introduced in section 2.5, to be used in bias correction.

```
data(testdl)
datalist_com1 <- extractPeriod(testdl, startDate = '1994-1-1', endDate = '1995-10-1')

dataframe <- list2Dataframe(datalist_com1)
# now we have a dataframe to extract certain months and years.
dataframe_new <- extractPeriod(dataframe, month = c(1,2,3))
dataframe_new <- extractPeriod(dataframe, month = c(12,1,2), year = 1995)
```

1.3.2 Fill Gaps (rainfall data gaps)

Although we have got the precipitation of the common period, we can still see that there are some missing values inside, which we should fill.

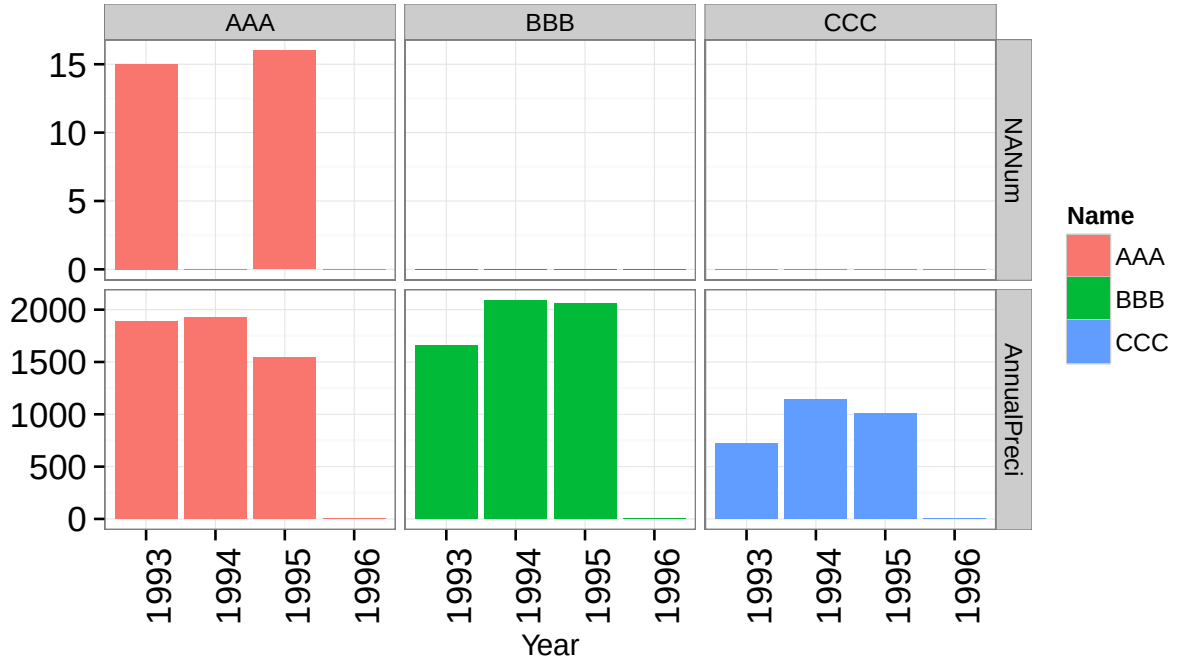
```
testdl_new <- extractPeriod(testdl, commonPeriod = TRUE )
a <- getAnnual(testdl_new)
```

```
## Using Year, Name as id variables
```

```
a
```

```
##    Year Name AnnualPreci recordNum NANum
## 1  1993  AAA    1883.157       264    15
## 2  1994  AAA    1927.704       365     0
## 3  1995  AAA    1543.893       349    16
## 4  1996  AAA       5.394         1     0
## 5  1993  BBB    1657.080       279     0
## 6  1994  BBB    2090.970       365     0
## 7  1995  BBB    2056.230       365     0
```

```
## 8  1996  BBB       3.060         1     0
## 9  1993  CCC     724.560       279     0
## 10 1994  CCC    1139.640       365     0
## 11 1995  CCC    1006.260       365     0
## 12 1996  CCC       0.000         1     0
```



First we have to transform the datalist to dataframe, which can be done by the code below:

```
df <- list2Dataframe(testdl_new)
head(df)
```

```
##         Date AAA  BBB   CCC
## 1 1993-03-28  NA 0.00  0.72
## 2 1993-03-29  NA 1.26  1.56
## 3 1993-03-30  NA 0.00 20.82
## 4 1993-03-31  NA 0.00 18.90
## 5 1993-04-01  NA 0.00  9.54
## 6 1993-04-02  NA 0.00  0.00
```

From above, we can see that in the gauging station "AAA", there are some missing value marked as "NA". Now we are going to fill these gaps.

The gap filling is based on the correlation and linear regression between each two gauging stations, correlation table, correlation Order and Linear Coefficients are also printed when doing the calculation. Details can be found in `?fillGap`.

```
df_filled <- fillGap(df)
```

```
##
## Correlation Coefficient
```

```
##              AAA         BBB         CCC
## AAA  1.000000000 -0.07445112 0.008566204
## BBB -0.074451120  1.00000000 0.039809765
## CCC  0.008566204  0.03980976 1.000000000
##
## Correlation Order
##     1     2
## AAA "CCC" "BBB"
## BBB "CCC" "AAA"
## CCC "BBB" "AAA"
##
## Linear Coefficients
##             1          2
## AAA 0.3308048 0.12015931
## BBB 0.3756172 0.11752878
## CCC 0.1094488 0.09047318
```

```
head(df_filled)
```

```
##         Date   AAA  BBB   CCC
## 1 1993-03-28 0.238 0.00  0.72
## 2 1993-03-29 0.516 1.26  1.56
## 3 1993-03-30 6.887 0.00 20.82
## 4 1993-03-31 6.252 0.00 18.90
## 5 1993-04-01 3.156 0.00  9.54
## 6 1993-04-02 0.000 0.00  0.00
```

Default correlation period is "daily", while sometimes the daily rainfall correlation of precipitation is not so strong, we can also select the correlation period.

```
df_filled <- fillGap(df, corPeriod = 'monthly')
```

```
##
## Correlation Coefficient
##              AAA         BBB       CCC
## AAA  1.00000000 -0.02020277 0.4980004
## BBB -0.02020277  1.00000000 0.2513406
## CCC  0.49800040  0.25134059 1.0000000
##
## Correlation Order
##     1     2
## AAA "CCC" "BBB"
## BBB "CCC" "AAA"
## CCC "AAA" "BBB"
##
## Linear Coefficients
##              1         2
## AAA 0.33080477 0.1201593
## BBB 0.37561723 0.1175288
## CCC 0.09047318 0.1094488
```

```r
head(df_filled)
```

```
##         Date   AAA  BBB   CCC
## 1 1993-03-28 0.238 0.00  0.72
## 2 1993-03-29 0.516 1.26  1.56
## 3 1993-03-30 6.887 0.00 20.82
## 4 1993-03-31 6.252 0.00 18.90
## 5 1993-04-01 3.156 0.00  9.54
## 6 1993-04-02 0.000 0.00  0.00
```

```r
df_filled <- fillGap(df, corPeriod = 'yearly')
```

```
##
## Correlation Coefficient
##             AAA       BBB        CCC
## AAA 1.00000000 0.1894243 0.02040045
## BBB 0.18942426 1.0000000 0.97659734
## CCC 0.02040045 0.9765973 1.00000000
##
## Correlation Order
##     1     2
## AAA "BBB" "CCC"
## BBB "CCC" "AAA"
## CCC "BBB" "AAA"
##
## Linear Coefficients
##              1          2
## AAA 0.1201593 0.33080477
## BBB 0.3756172 0.11752878
## CCC 0.1094488 0.09047318
```

```r
head(df_filled)
```

```
##         Date   AAA  BBB   CCC
## 1 1993-03-28 0.000 0.00  0.72
## 2 1993-03-29 0.151 1.26  1.56
## 3 1993-03-30 0.000 0.00 20.82
## 4 1993-03-31 0.000 0.00 18.90
## 5 1993-04-01 0.000 0.00  9.54
## 6 1993-04-02 0.000 0.00  0.00
```

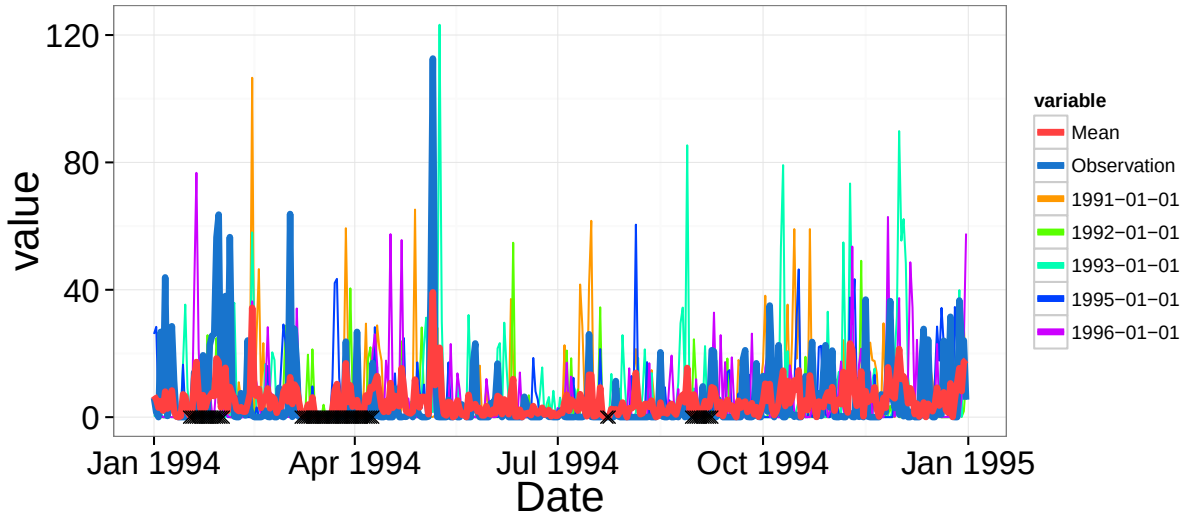
1.3.3 Get Ensemble Hydrological Forecast from Historical Data (ESP method)

The basic forecasts are made from the historical data, to see, how the historical data act in the same situation. Using the same period from the historical data to generate an ensemble forcast.

E.g., we have a period of data from 2000 to 2007, we assume 2004 to be the forecast year. Then, use 2004 as an example, the data in 2000, 2001, 2002, 2003, 2005, 2006, 2007 will be taken to generate an ensemble forecast of 6 members(except 2004).

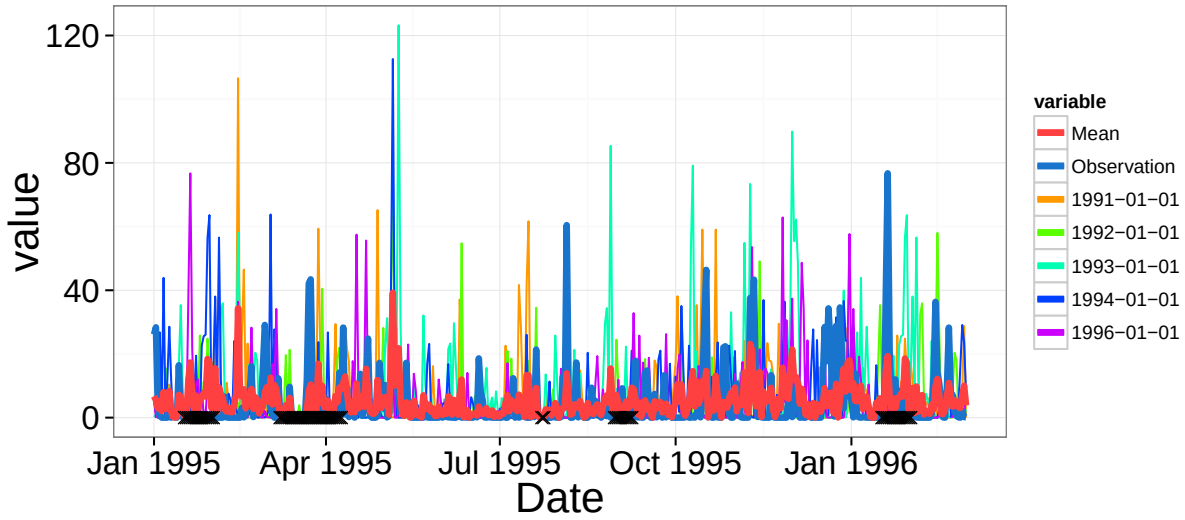
Set example year, e.g., year 1994. mean value of each ensemble will be returned in the second column, while if you don't want the mean value, you can filter mean value by output[, -2], and you will get purely the output.

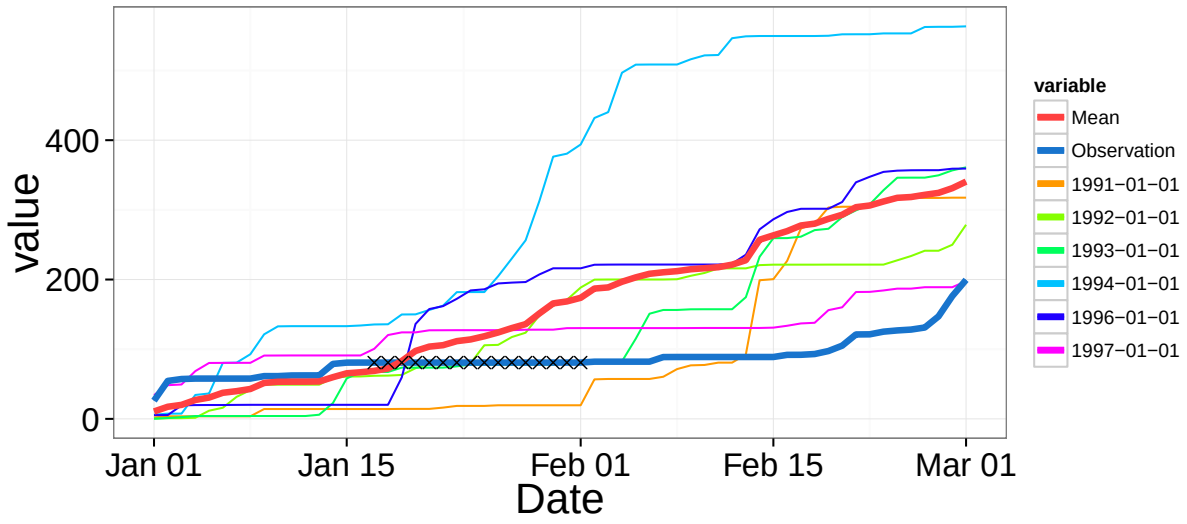
```
data(testdl)

a <- testdl[[1]]
a1 <- getHisEnsem(a, example = c('1994-1-1', '1994-12-31'))
```



Both cumulative and normal plot are provided, default is "norm", means normal plot without any process. If words other that "norm", "plot", there will be no plot. If there are missing values inside, cumulative plot will stop when finds missing values. As can be seen from below.

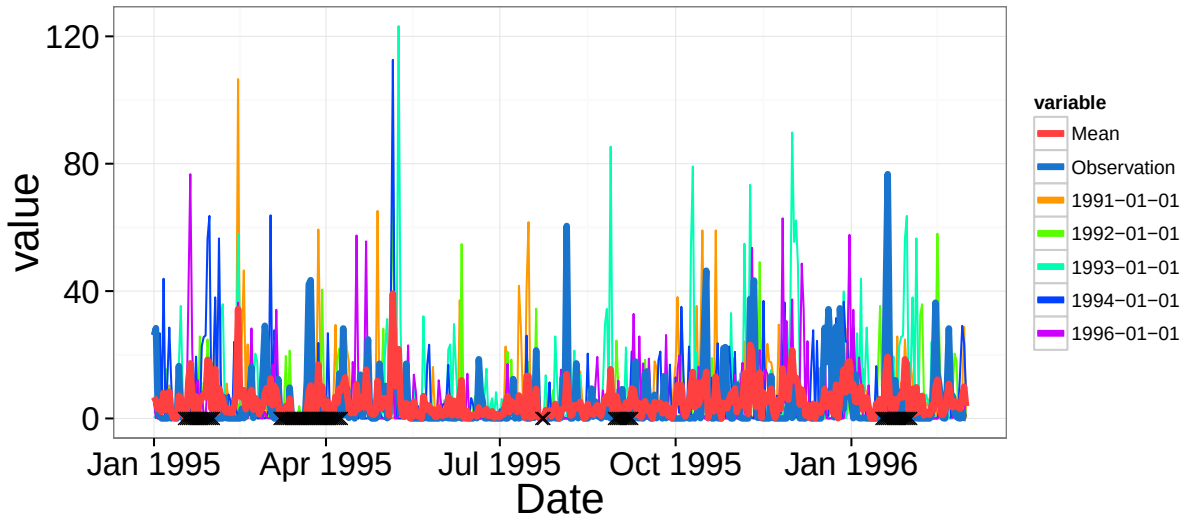
```
a2 <- getHisEnsem(a, example = c('1995-1-1', '1996-3-1'))# Default is plot = 'norm'
a3 <- getHisEnsem(a, example = c('1995-1-1', '1995-3-1'), plot = 'cum')
```

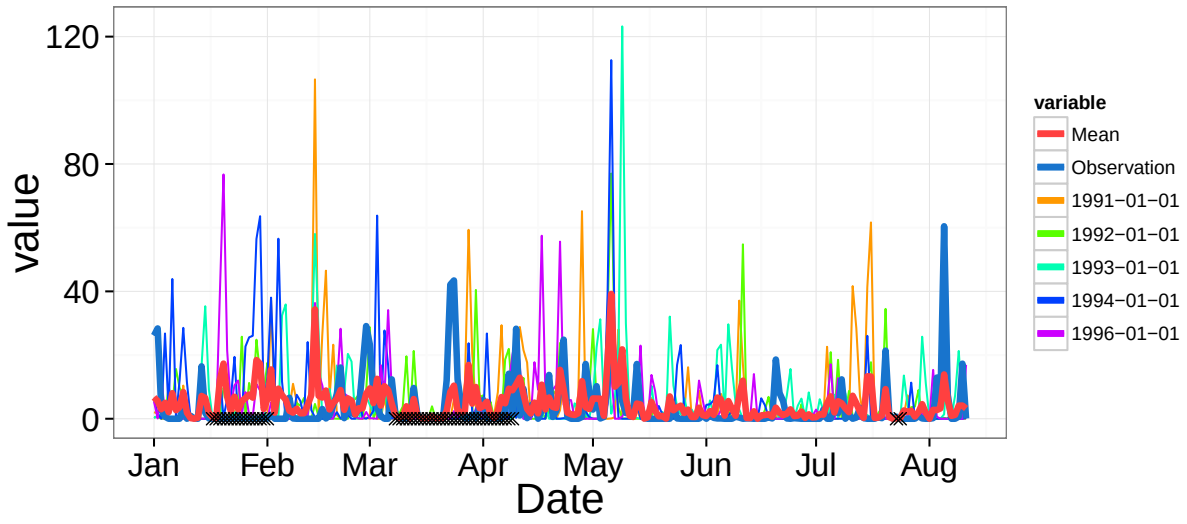




Example period can be any time, can be a year or some months.

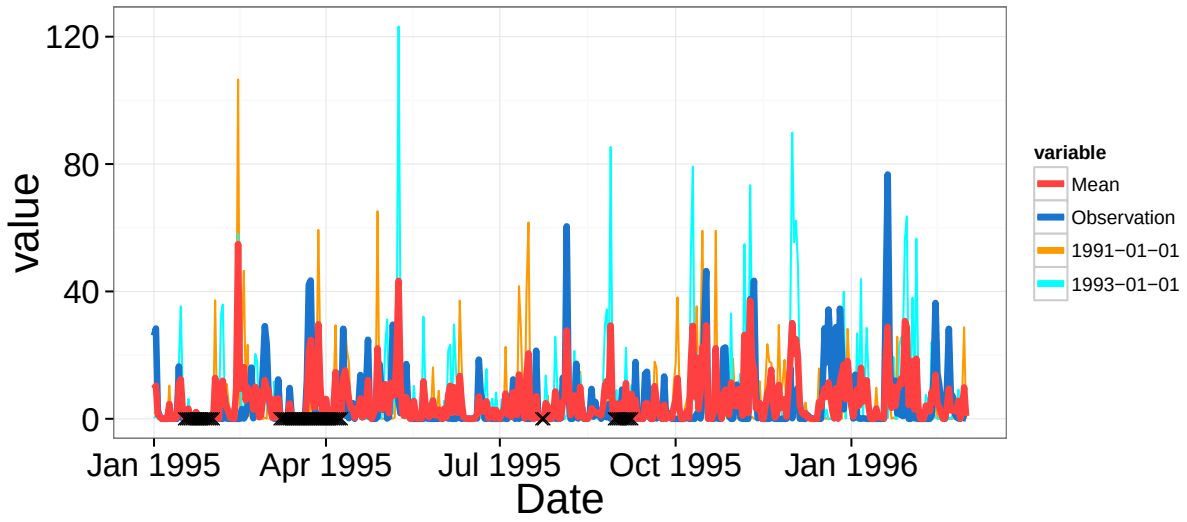
```
a2 <- getHisEnsem(a, example = c('1995-1-1', '1996-3-1'))
a3 <- getHisEnsem(a, example = c('1995-1-1', '1995-8-11'))
```



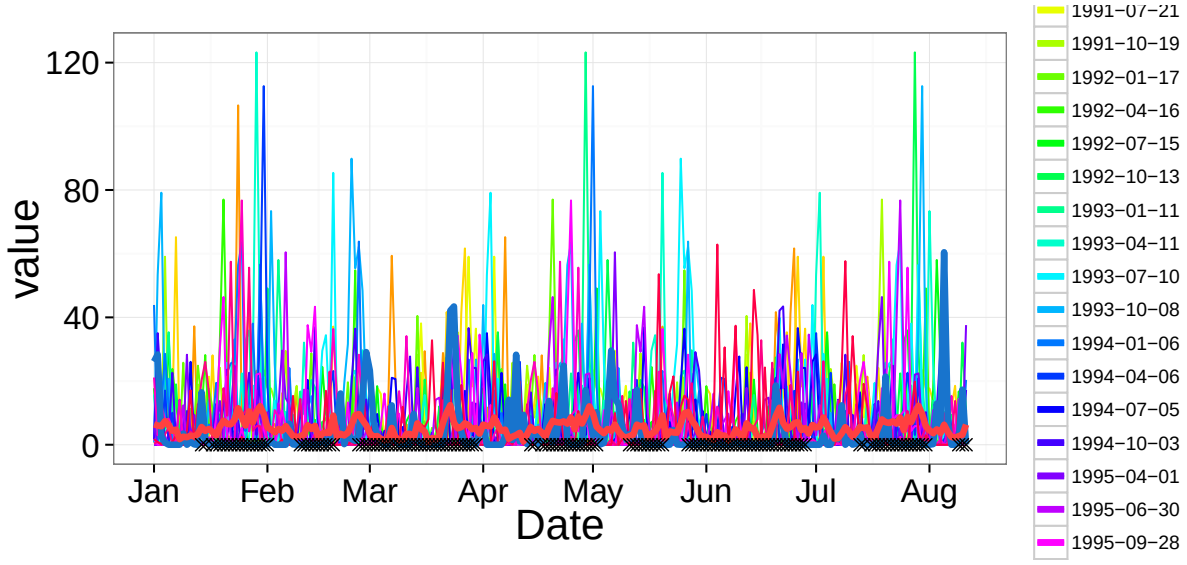


`interval` means the interval between each member. Check `?getHisEnsem` for detailed instruction. Default is 365, representing one year.

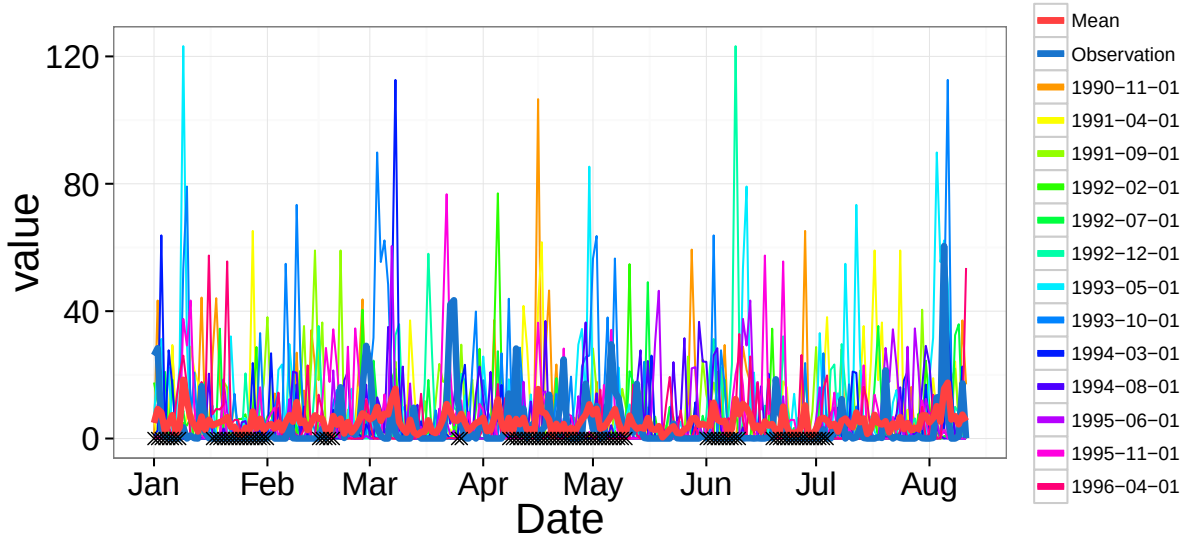
```
# If interval is two years.
a2 <- getHisEnsem(a, example = c('1995-1-1', '1996-3-1'), interval = 730)
```




```
str(a2)
# If interval is three months.
a3 <- getHisEnsem(a, example = c('1995-1-1', '1995-8-11'), interval = 90)
```



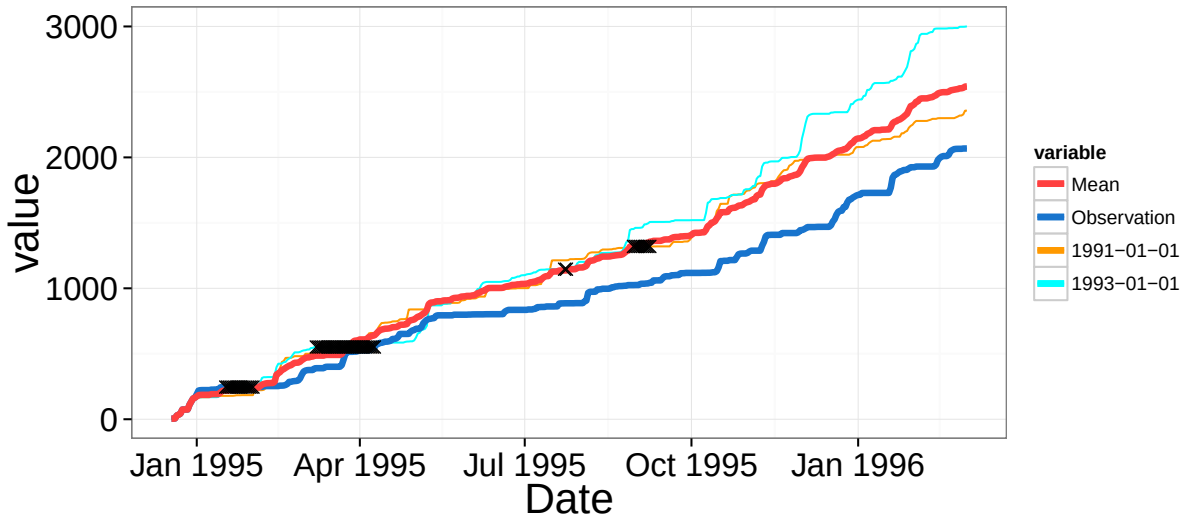

```
str(a3)
# If interval is 171 days.
a4 <- getHisEnsem(a, example = c('1995-1-1', '1995-8-11'), interval = 171)
```




```
str(a4)
```

For some models, like MIKE NAM, it's necessary to run model a few days before the forecasting time, to warm up the model. In this case `buffer` is needed to generate the "warm up period".

```
# If the model needs 14 days to warm up.
a2 <- getHisEnsem(a, example = c('1995-1-1', '1996-3-1'), interval = 730,
                  buffer = 14, plot = 'cum')
```

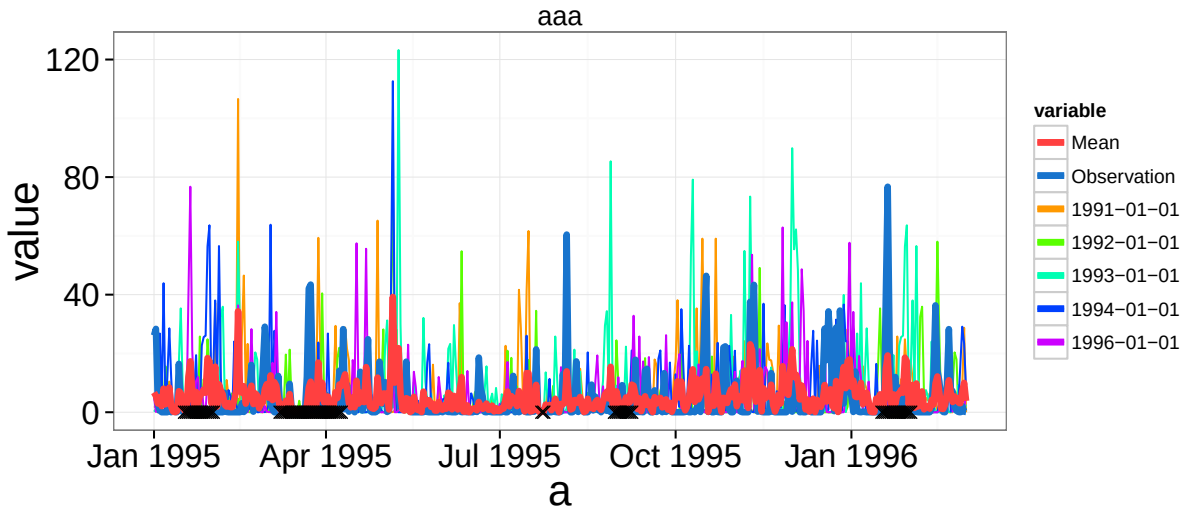



```
str(a2)
```

From `str(a2)` we can see that the data has 14 more rows, and the start date is changed to "1994-12-18"

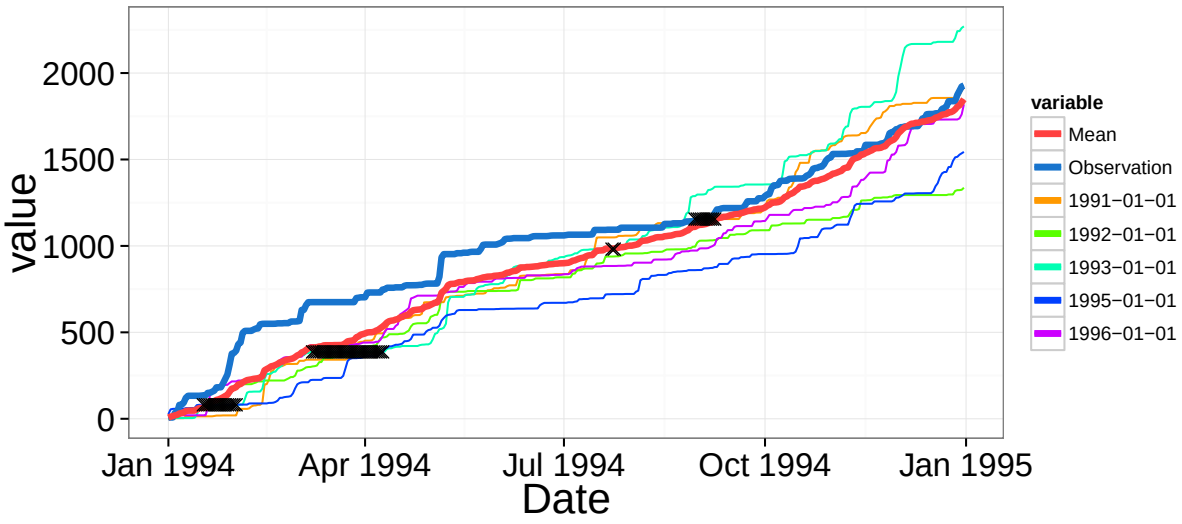
Also, if costomized title and xy axis are needed, you can set yourself.

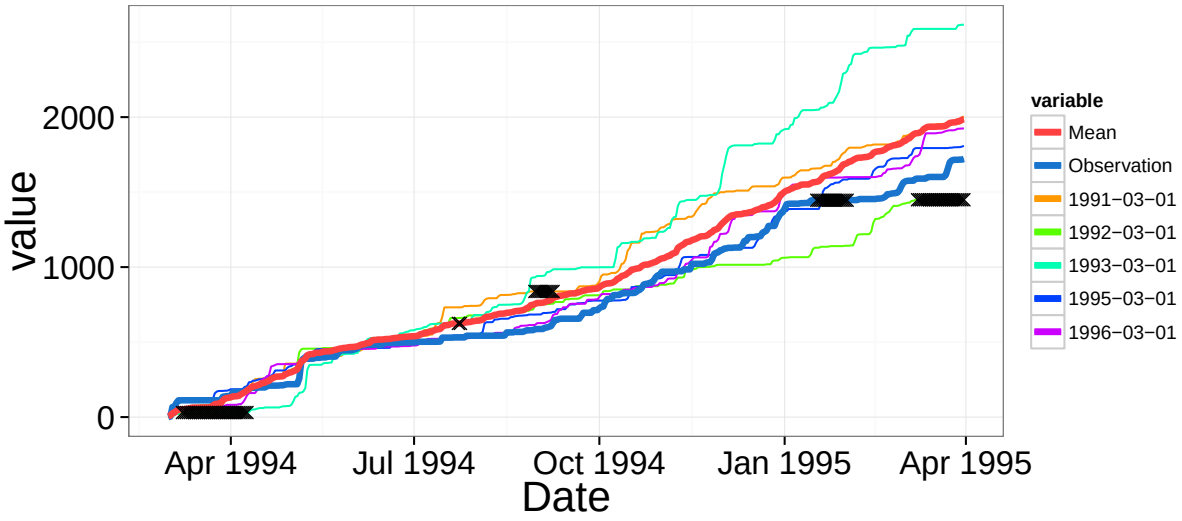
```
a2 <- getHisEnsem(a, example = c('1995-1-1', '1996-3-1'), title = 'aaa', x = 'a')
```

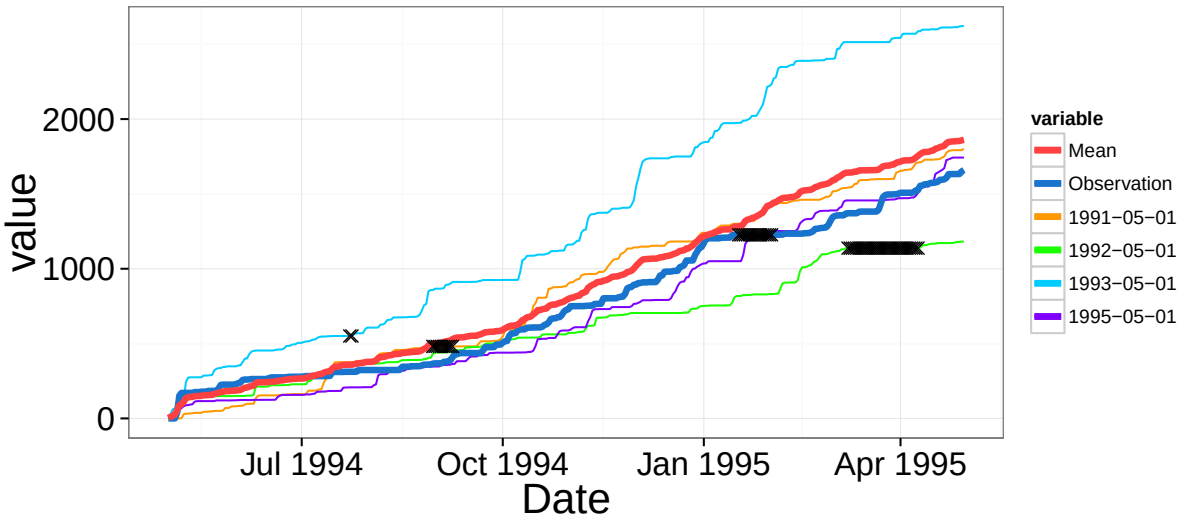


If you want to combine different ensemble together, there is a regular `_comb` function `getEnsem_comb` to combine different plots together.

```
a1 <- getHisEnsem(a, example = c('1994-1-1', '1994-12-31'), plot = 'cum',
                  output = 'ggplot', name = 1)
a2 <- getHisEnsem(a, example = c('1994-3-1', '1995-3-31'), plot = 'cum',
                  output = 'ggplot', name = 2)
a3 <- getHisEnsem(a, example = c('1994-5-1', '1995-4-30'), plot = 'cum',
                  output = 'ggplot', name = 3)
```

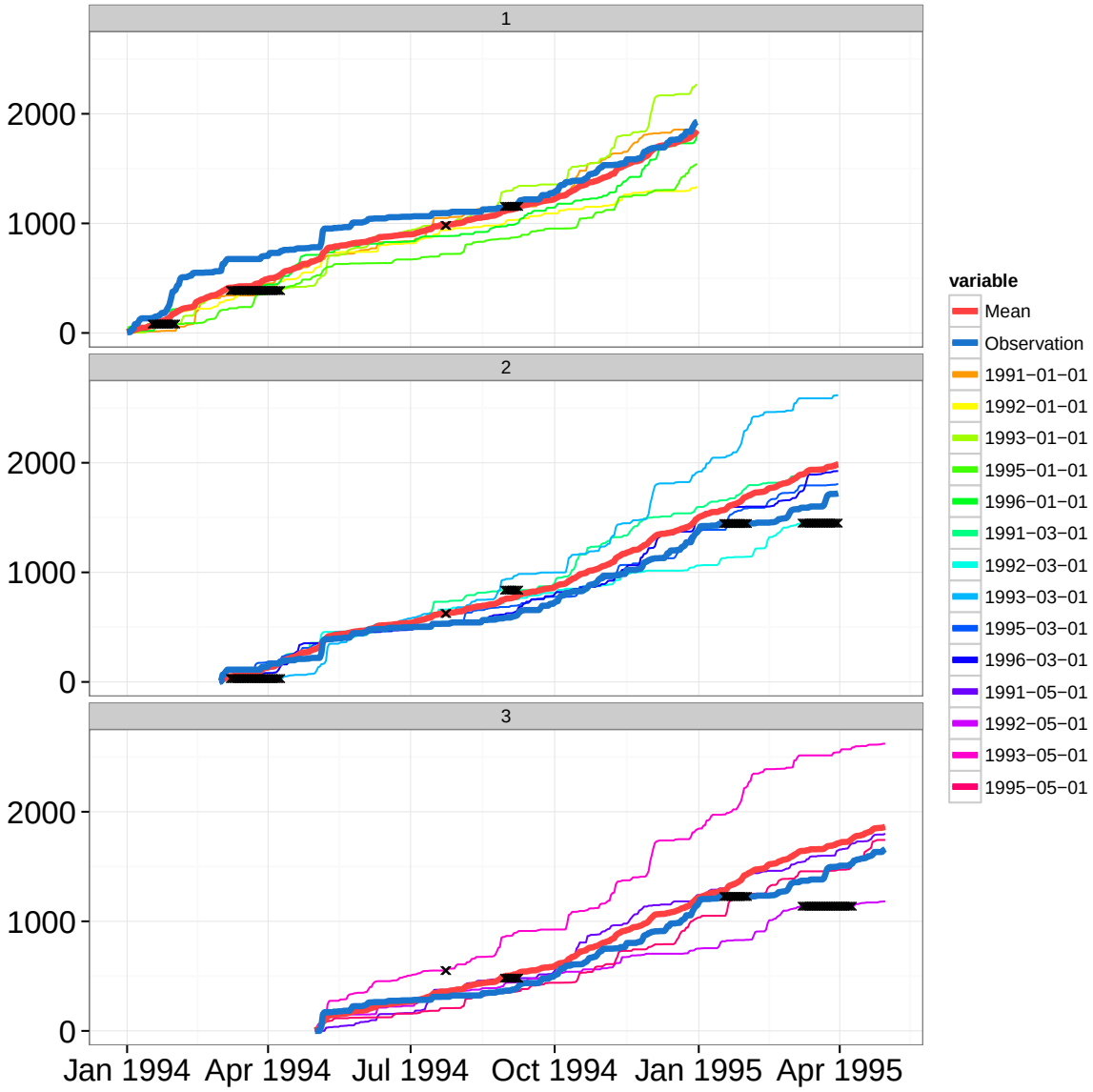

value
2000
1500
1000
500
0
Jan 1994
Apr 1994
Jul 1994
Oct 1994
Jan 1995
Date
variable
Mean
Observation
1991−01−01
1992−01−01
1993−01−01
1995−01−01
1996−01−01


value
2000
1000
0
Apr 1994
Jul 1994
Oct 1994
Jan 1995
Apr 1995
Date
variable
Mean
Observation
1991−03−01
1992−03−01
1993−03−01
1995−03−01
1996−03−01


value
2000
1000
0
Jul 1994
Oct 1994
Jan 1995
Apr 1995
Date
variable
Mean
Observation
1991−05−01
1992−05−01
1993−05−01
1995−05−01

```
getEnsem_comb(a1, a2, a3, nrow = 3)
```

```
## Check if the data list is available for rbind or cbind...
##
## Data list is OK
```



1.3.4 Resample Data

Sometimes you have the monthly data, and want to generate the daily data, sometimes the opposite situation. `resample` can help you with the conversion. For now, hyfo provides monthly mean data to daily data conversion, method `mon2day` and daily data to monthly mean data conversion, method `day2mon`.

If you have daily data and want to convert it to a monthly data.

```
data(testdl)
TS <- testdl[[2]] # Get daily data
TS_new <- resample(TS, method = 'day2mon')
```

If you have monthly data and want to convert it to a daily data.

```
# First generate a monthly data.
TS <- data.frame(Date = seq(as.Date('1999-9-15'), length = 30, by = '1 month'),
                 stats::runif(30, 3, 10))
TS_new <- resample(TS, method = 'mon2day')
```

Not only for the time series, but also the hyfo grid data can be resampled.

```
filePath <- system.file("extdata", "tnc.nc", package = "hyfo")
varname <- getNcdfVar(filePath)
nc <- loadNcdf(filePath, varname)
```

```
## Loading data...
## Processing...
```

```
nc_new <- resample(nc, 'day2mon')
```

More information please check `?resample`.

1.4 Seasonal and Monthly Precipitation Analysis

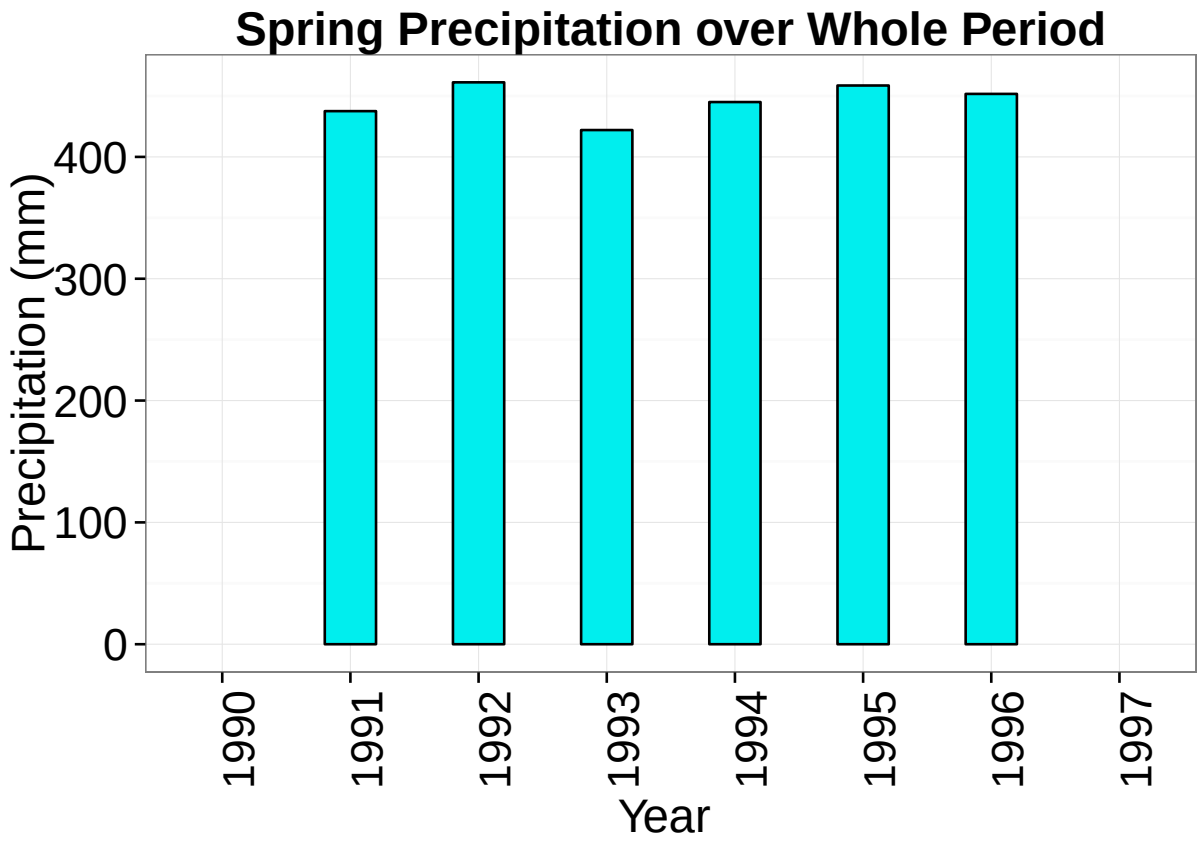
Sometimes we need to know not only the annual precipitation, but also the precipitation of a certain month or certain season. `getPreciBar` is in charge of different analysis. It can analyze both grid file and singe timeseries. IF the input is a time series, the argument `TS =` must be put.

`info` argument will give information about max, min, mean, and median, if selected TRUE.

```
data(testdl)
TS  <- testdl[[1]]
a <- getPreciBar(TS, method = 'spring')
```

```
## There is no plotRange for this method
```

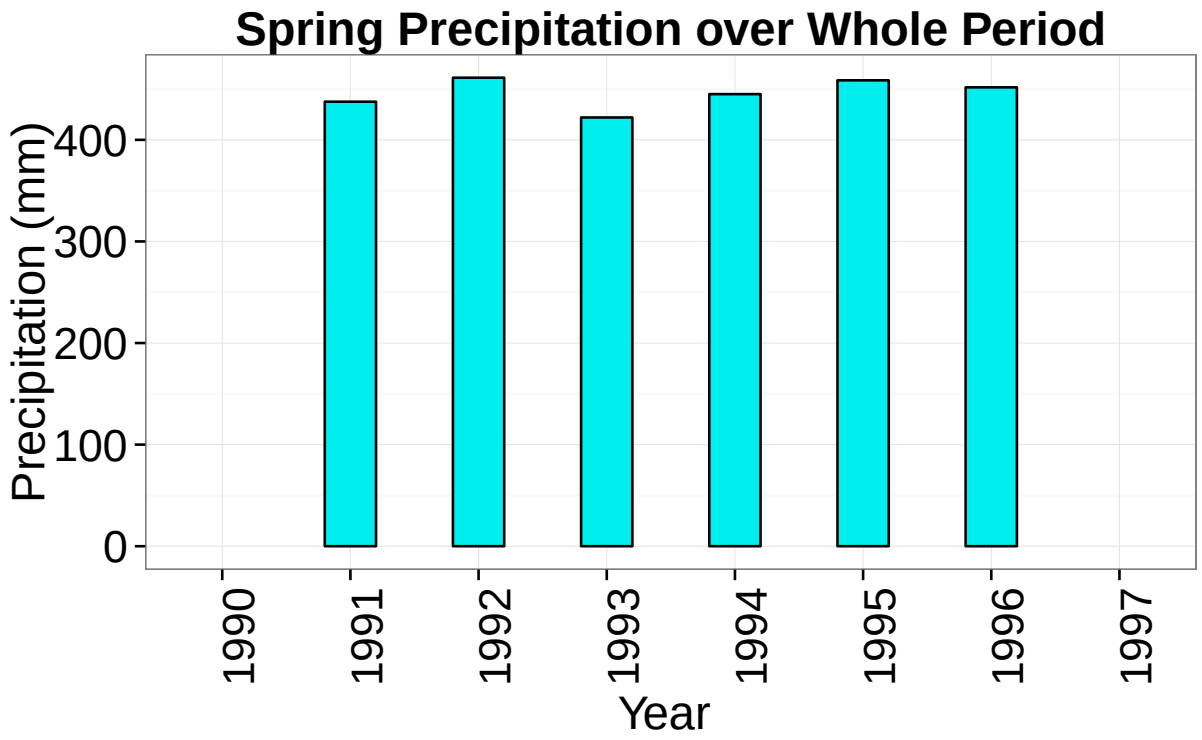
```
## Warning: Removed 2 rows containing missing values (position_stack).
```




```
# if info = T, the information will be given at the bottom.
a <- getPreciBar(TS, method = 'spring', info = TRUE)
```

```
## There is no plotRange for this method
```

```
## Warning: Removed 2 rows containing missing values (position_stack).
```



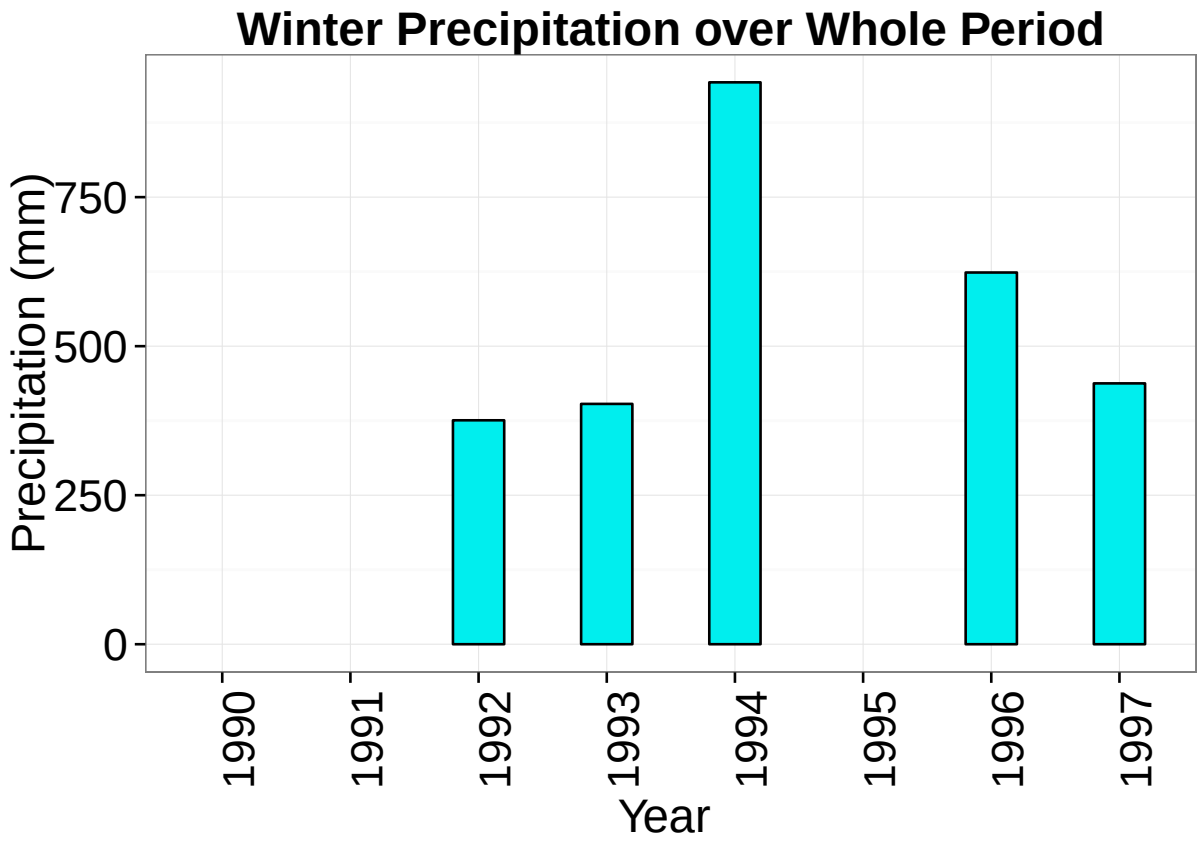
Max = 461.19 , Min = 422.03 , Mean = 446.01 , Median =

If missing value is wanted, set `omitNA = FALSE`.

```
a <- getPreciBar(TS, method = 'winter', omitNA = FALSE)
```

```
## There is no plotRange for this method
```

```
## Warning: Removed 3 rows containing missing values (position_stack).
```

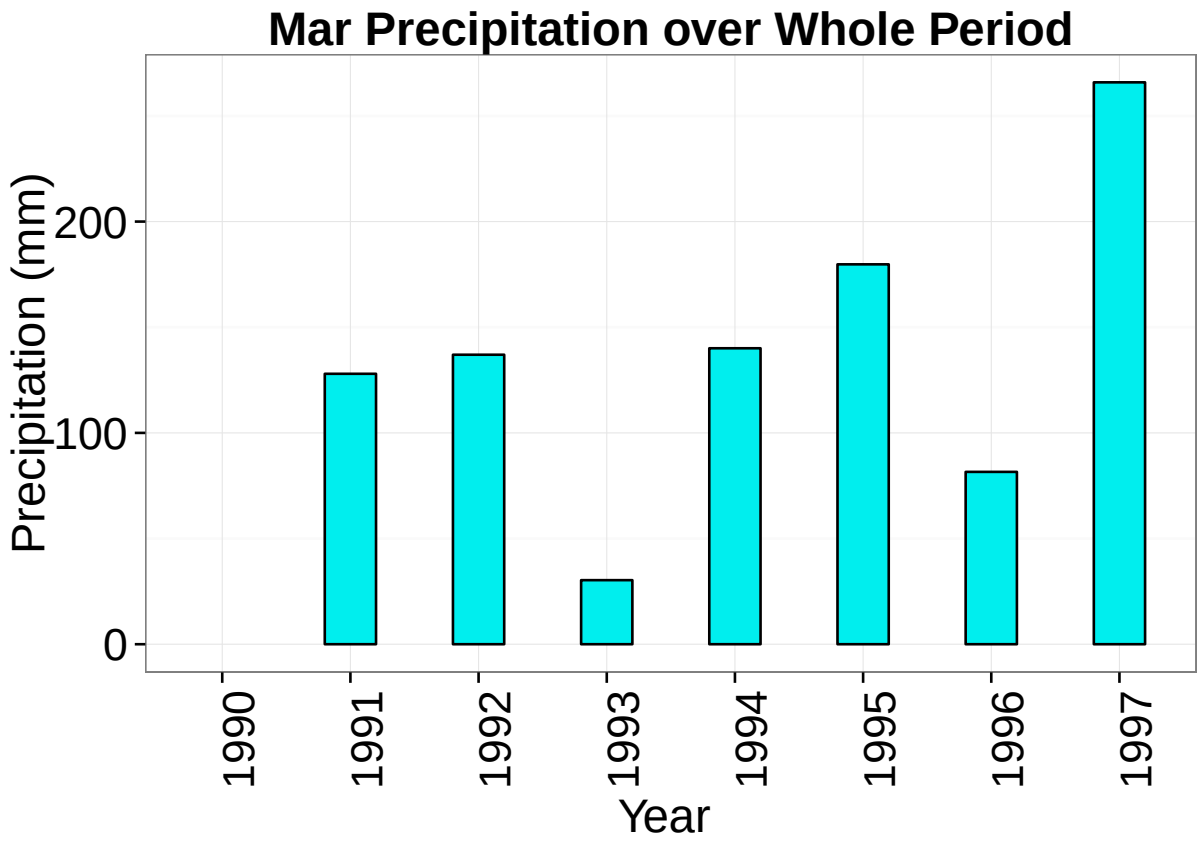


Get special month precipitation, e.g. march.

```
a <-getPreciBar(TS, method = 3)
```

```
## There is no plotRange for this method
```

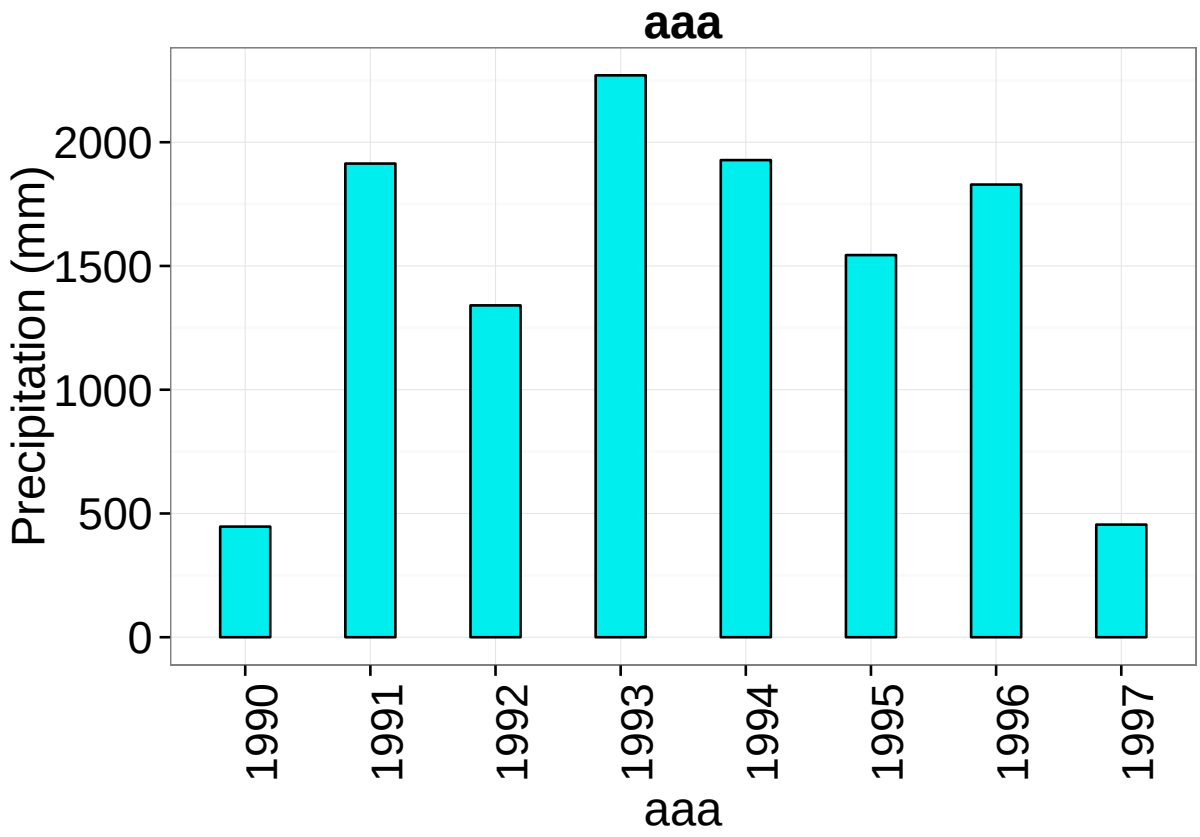
```
## Warning: Removed 1 rows containing missing values (position_stack).
```



We can also get annual precipitation, plot figure and assign title ans axis.

```
a <- getPreciBar(TS, method = 'annual', x = 'aaa', title = 'aaa')
```

```
## There is no plotRange for this method
```

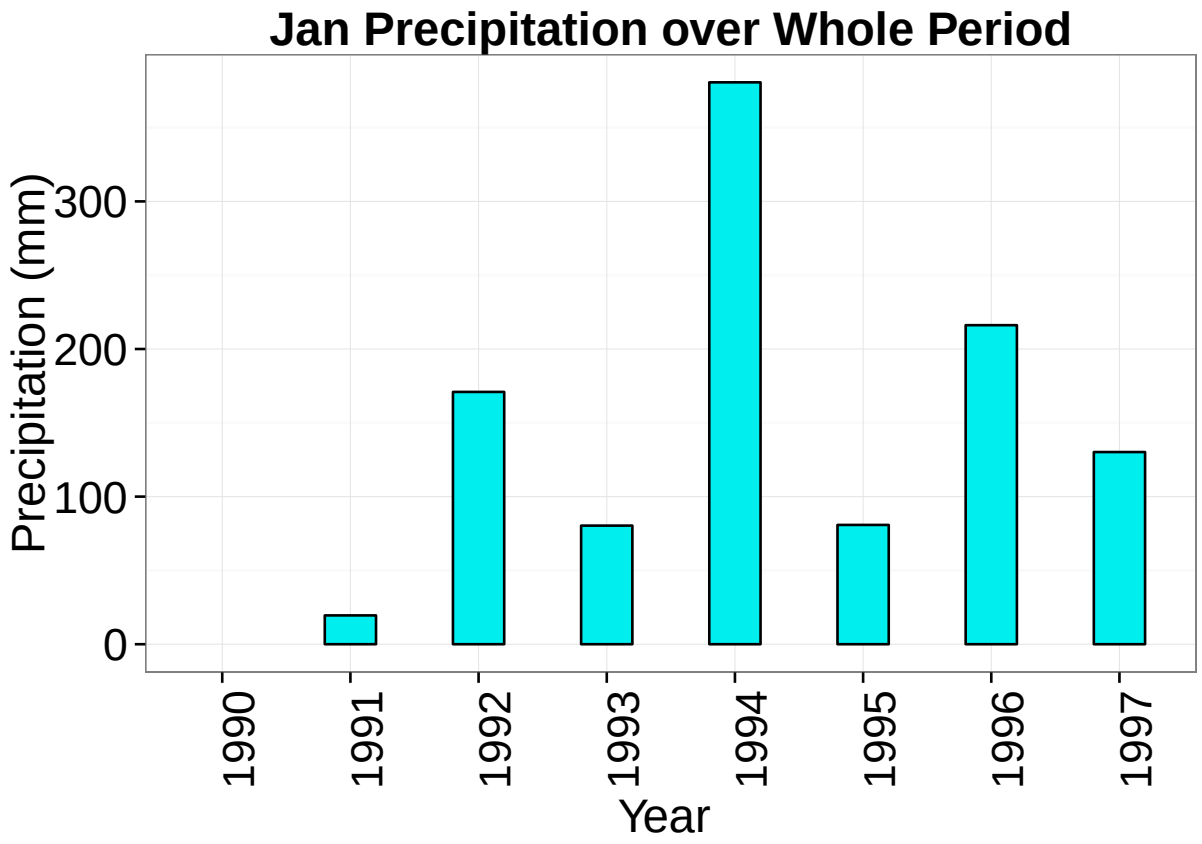


If `output = 'ggplot'` is chosen, the the output can be used in `getPreciBar_comb`, to generate multiple plots.

```
a1 <- getPreciBar(TS, method = 1, output = 'ggplot', name = 'Jan')
```

```
## There is no plotRange for this method
```

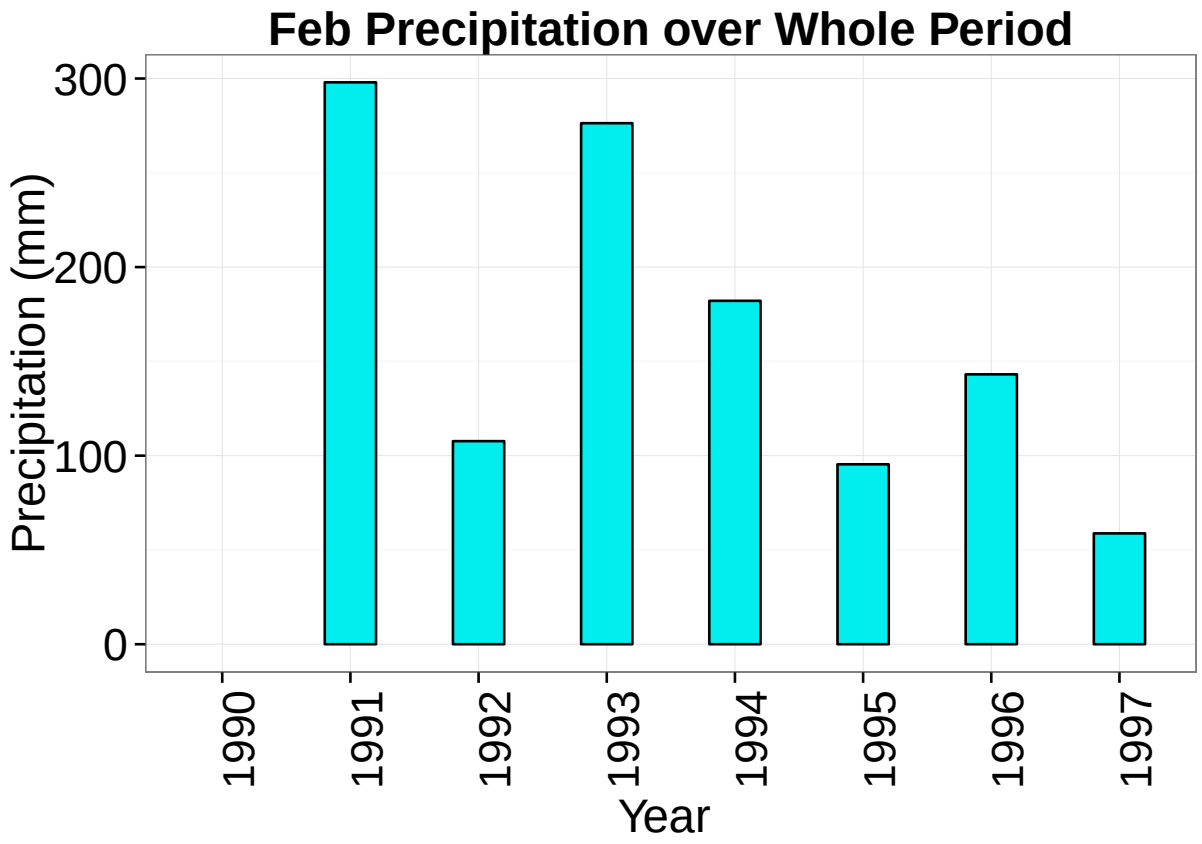
```
## Warning: Removed 1 rows containing missing values (position_stack).
```




```
a2 <- getPreciBar(TS, method = 2, output = 'ggplot', name = 'Feb')
```

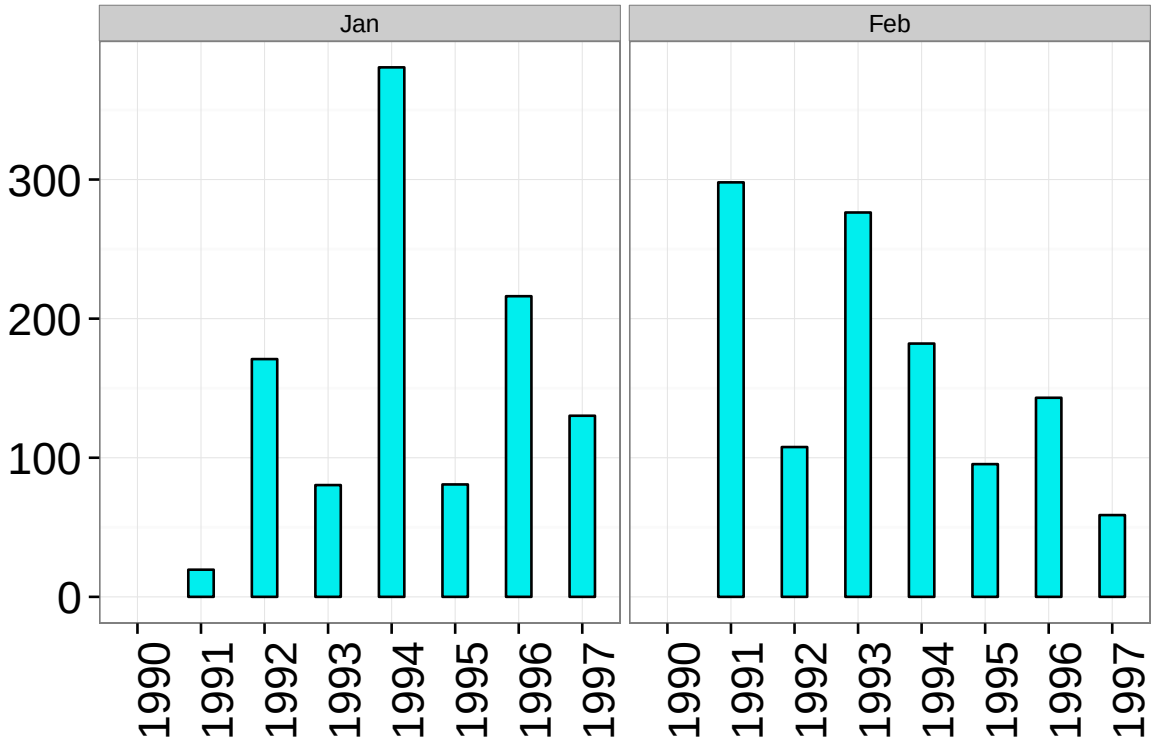
```
## There is no plotRange for this method
```

```
## Warning: Removed 1 rows containing missing values (position_stack).
```




```
getPreciBar_comb(a1, a2)
```

```
## Check if the data list is available for rbind or cbind...
##
## Data list is OK
```



2. Climate Forecasting

- For the climate forecasting part, `hyfo` mainly focuses on the post processing of the gridData derived from forecasts or other sources. The input is a list file, usually an NetCDF file. There are `getNcdfVar()`, `loadNcdf()` and `writeNcdf()` prepared in hyfo, for you to deal with NetCDF file. `loadNcdf()` will give a list based hyfo output file.

Note If an **ensemble forecast** data is loaded, there will be one dimension called "member", by default, `hyfo` will calculate the mean of different members. If you want to see a special member, add `member` argument to `getSpatialMap`, e.g., `getSpatialMap(tgridData, method = 'meanAnnual', member = 3)`, `getPreciBar(tgridData, method = 'annual', member = 14)`

2.1 Load, write and downscale NetCDF file

There are three main functions dealing with `getNcdfVar()`, `loadNcdf()` and `writeNcdf()`. `getNcdfVar()` is for get the variable name if you don't know the name. Then you can load NetCDF file, and get a hyfo output, from which you will use in further analysis. Maybe you want to change some thing with the original NetCDF file. You can load first, then, make changes to hyfo output file and then write back to NetCDF file. Following examples shows the procedure.

```r
# First open the test NETcDF file.
filePath <- system.file("extdata", "tnc.nc", package = "hyfo")

# Then if you don't know the variable name, you can use \code{getNcdfVar} to
# get variable name
varname <- getNcdfVar(filePath)

nc <- loadNcdf(filePath, varname)
```

```
## Loading data...
## Processing...
```

```r
# nc is a list based hyfo output file, with this file, you can make further
# analysis.

# E.g. you want to make some changes to the data
nc$Data <- nc$Data - 0.3
```

```r
# Then write it back to file
writeNcdf(gridData = nc, filePath = 'test.nc')
```

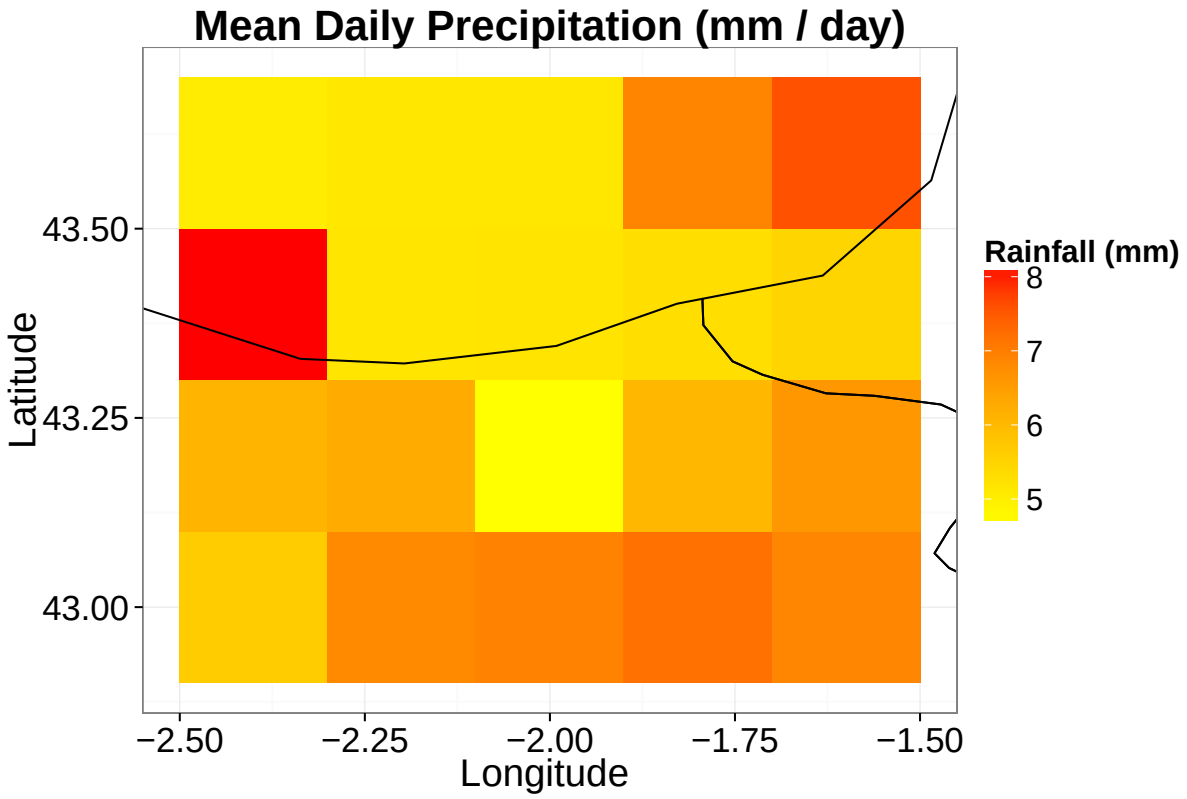
`hyfo` can also do downscale job. When you load the file, you can directly assign the year, month, longitude and latitude. And if you already have a hyfo list file, you can use `downscaleNcdf` to downscale your list file.

```r
# First open the test NETcDF file.
filePath <- system.file("extdata", "tnc.nc", package = "hyfo")

# Then if you don't know the variable name, you can use \code{getNcdfVar} to
# get variable name
varname <- getNcdfVar(filePath)

nc <- loadNcdf(filePath, varname, year = 2005:2006, month = c(2:9),
               lon = c(-2.4, -1.6), lat = c(41, 43.5))
```

```
## Loading data...
## Processing...
```

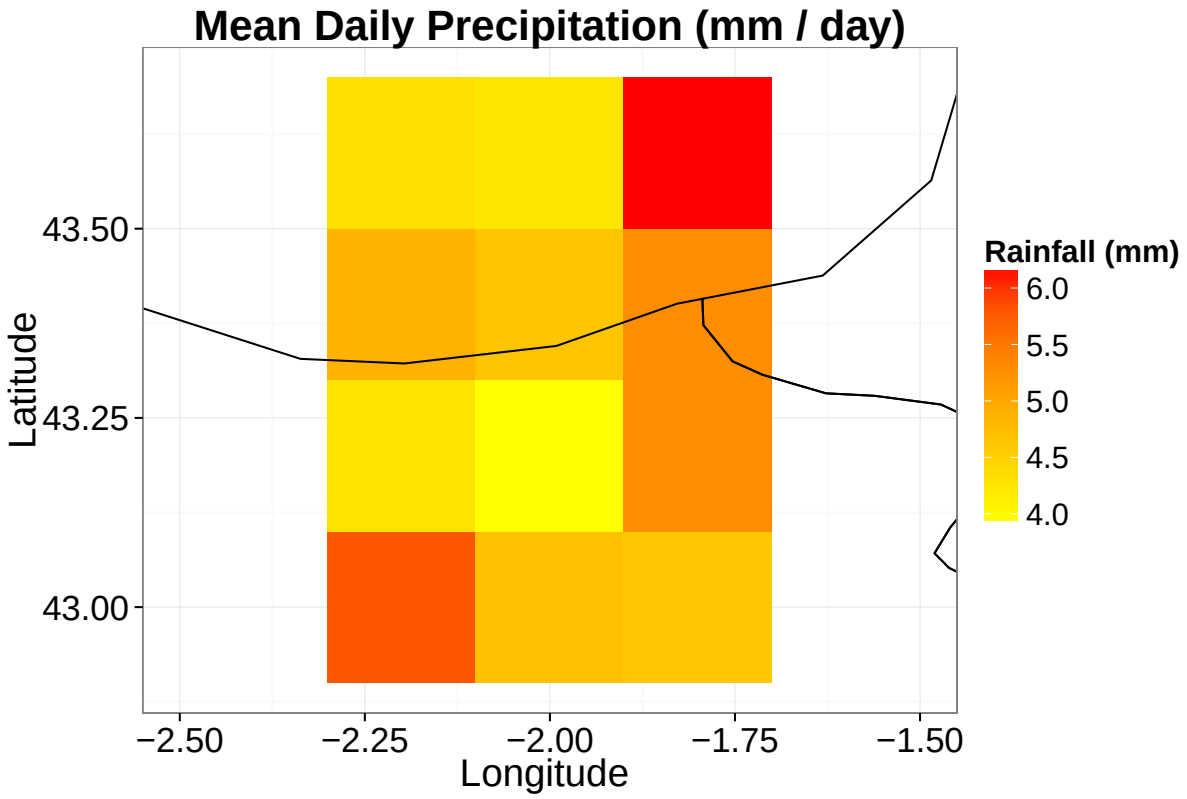
```r
nc_plot <- getSpatialMap(nc, 'mean')
```

```
## Mean value of the members are returned.
```




```
# if you want to further downscale nc, you can use the following function.
nc1 <- downscaleNcdf(nc, year = 2005, month = c(5:8), lon = c(-2.2, -1.75),
                     lat = c(43, 44))
nc1_plot <- getSpatialMap(nc1, 'mean')
```

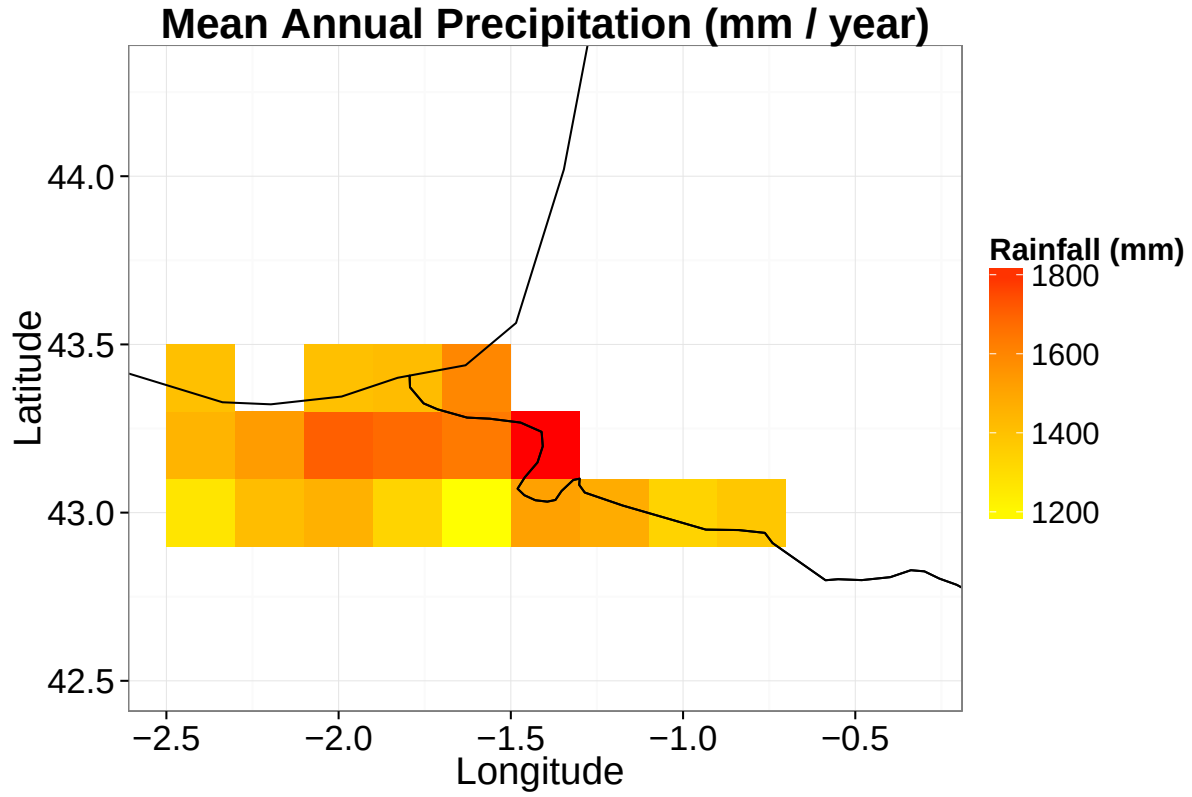
```
## Mean value of the members are returned.
```



2.2 Spatial Map Plot

As described before, the following analysis is based the list based hyfo output file. You can call elements by `$`. If we want to see the mean daily precipitation.

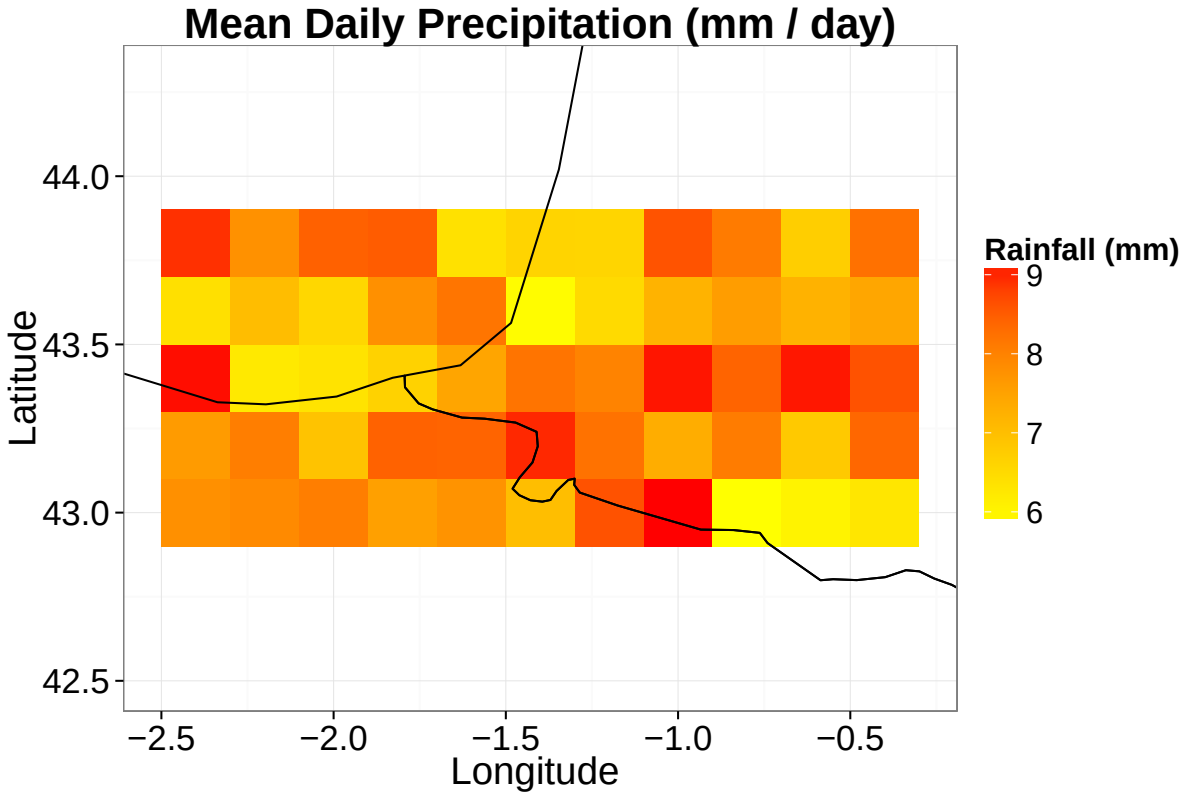
```
data(tgridData)
a <- getSpatialMap(tgridData, method = 'meanAnnual')
```




```
# If a dataset is an ensemble forecast, you can use argument member to choose
filePath <- system.file("extdata", "tnc.nc", package = "hyfo")

# Then if you don't know the variable name, you can use \code{getNcdfVar} to
# get variable name
varname <- getNcdfVar(filePath)

nc <- loadNcdf(filePath, varname)
```

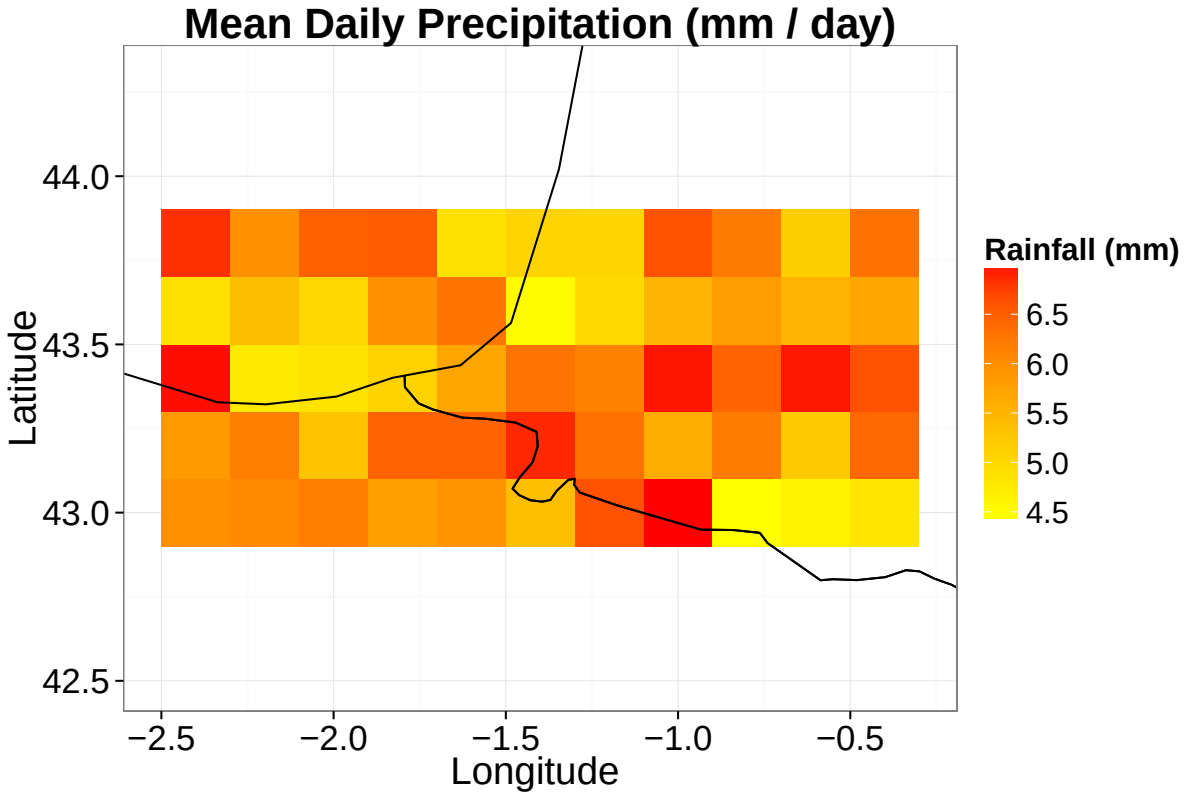
```
## Loading data...
## Processing...
```

```
# choose the 3rd member
a <- getSpatialMap(nc, method = 'mean', member = 2)
```




```
# If member not assigned, the mean value of the members will be plotted.
a <- getSpatialMap(nc, method = 'mean')
```

```
## Mean value of the members are returned.
```

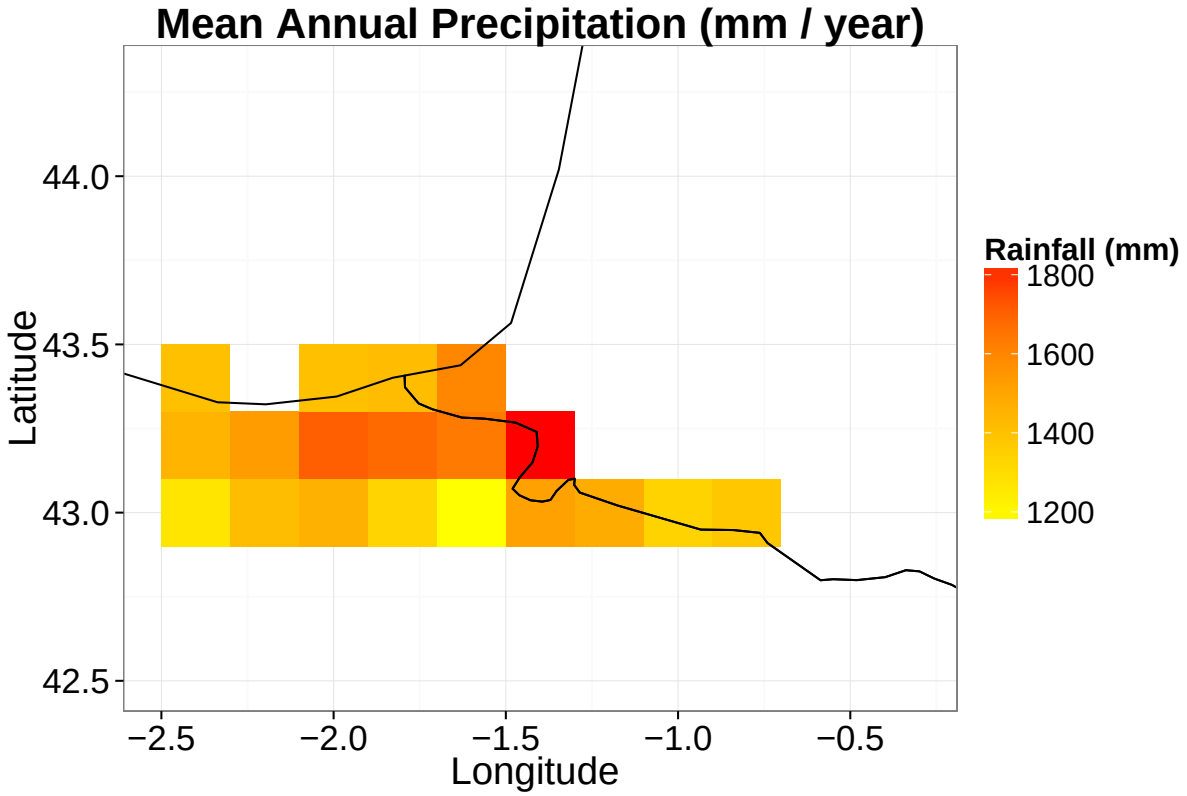


There are several methods to be seleted in the function, details can be found by `?getSpatialMap`.

Sometimes there exists a great difference in the whole map, e.g., the following value, `c(100, 2, 2,6, 1,7)`, since the maximum value is too large, so in the plot, by normal plot scale, we can only recognize value 100 and the rest, it's hard for us to tell the difference between 2, 2.6, and 1.7 from the plot. In this situation, the value needs to be processed before plotting. Here `scale` provides a way to decide the plot scale.

`scale` passes the arguments to the `trans` argument in `ggplot2`. The most common scale is "sqrt" and "log10", which focus more on the minutiae. Default is "identity", which means no change to the plot scale.

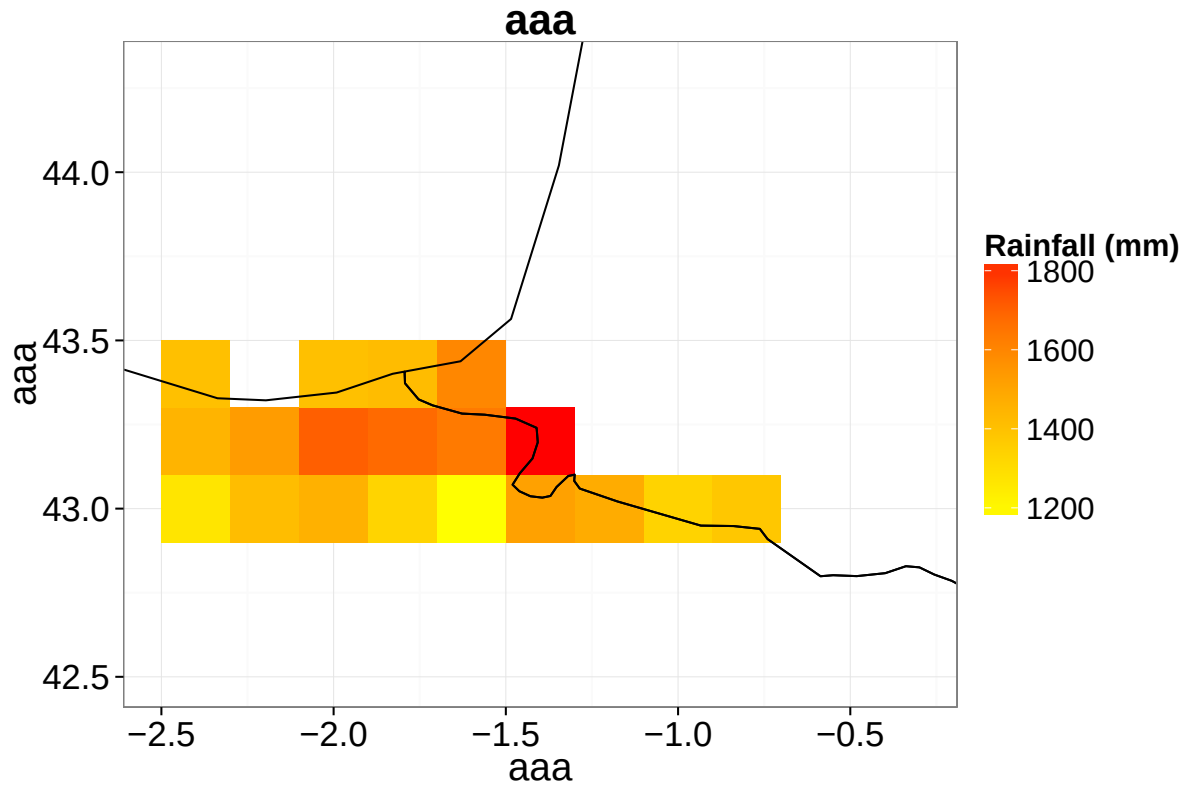
```
a <- getSpatialMap(tgridData, method = 'meanAnnual', scale = 'sqrt')
```



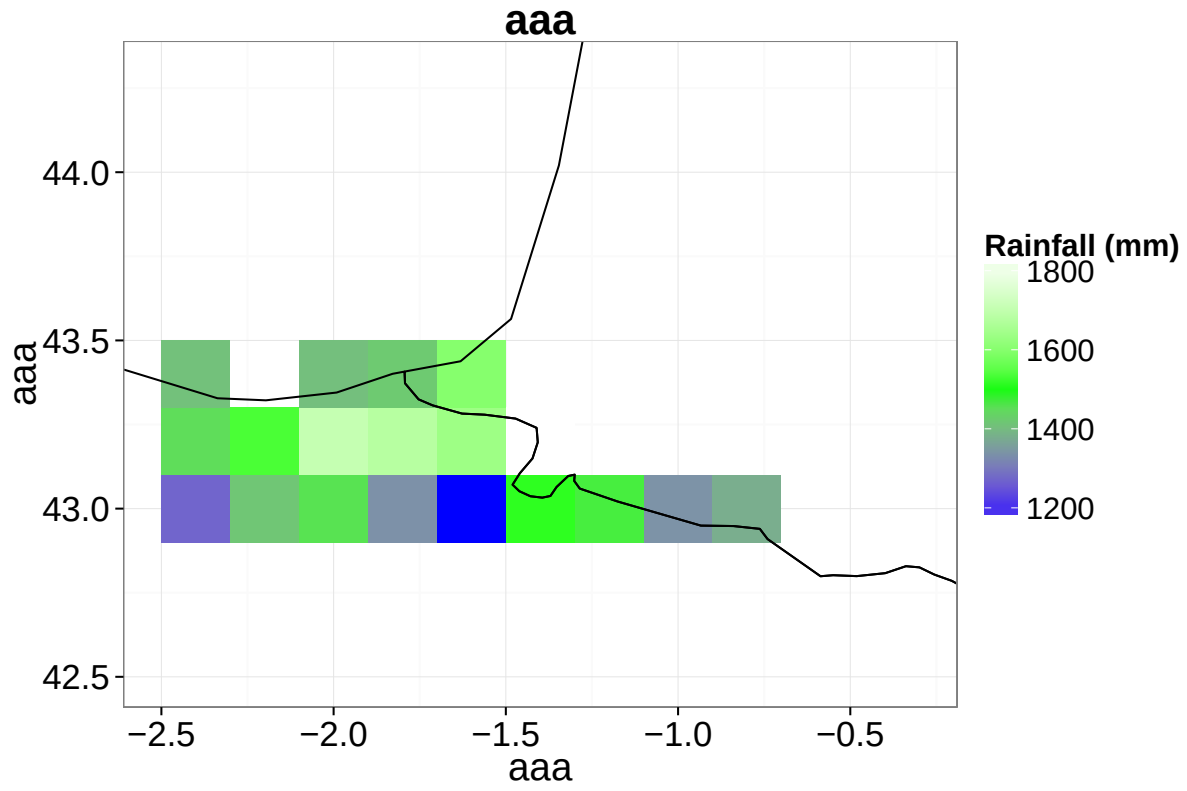
Here in our example, because the region is too small, and the differences is not so big, so it's not so obvious to tell from the plot. But if in a map, both dry region and wet region is included, that will be more obvious to see the difference between the plot scales.

Also, if you are not satisfied with the title, x axis and y axis, you can assgin yourself, and also the color of the map.

```
a <- getSpatialMap(tgridData, method = 'meanAnnual', scale = 'sqrt',
                   title = 'aaa', x = 'aaa', y = 'aaa')
```




```
# If you want to use your own color to plot the map, add color = yourcolor
a <- getSpatialMap(tgridData, method = 'meanAnnual', scale = 'sqrt',
                   title = 'aaa', x = 'aaa', y = 'aaa',
                   color = c('blue', 'green', 'white'))
```



2.3 Add Background Information (catchment and gauging stations)

The default background is the world map, while if you have other backgrounds like catchment shape file and station location file, you are welcome to import them as background.

2.3.1 Add catchment shape file

Catchment shape file needs to be processed with a very simple step. It's based on the package `rgdal`, details can be found by `?shp2cat`

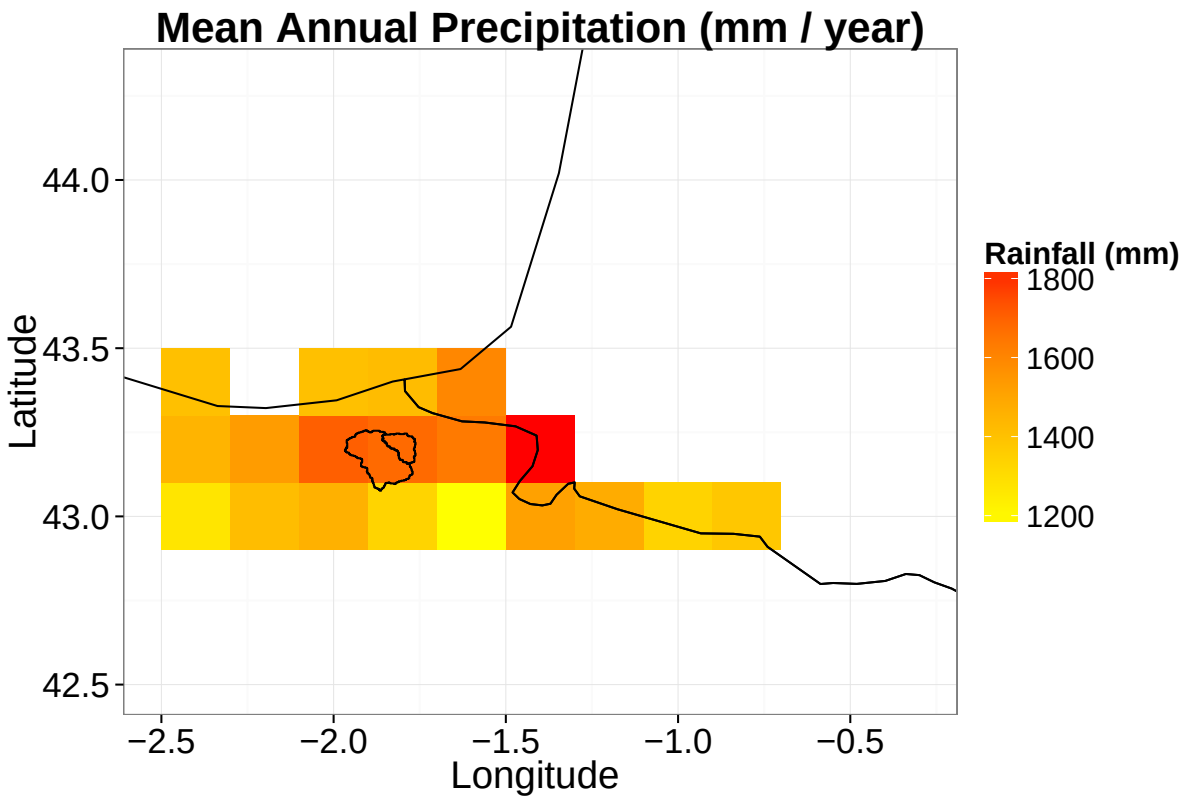
```
# Use the test file provided by hyfo
file <- system.file("extdata", "testCat.shp", package = "hyfo")
cat <- shp2cat(file)
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "C:/Users/Yuanchao/Documents/R/win-library/3.2/hyfo/extdata", layer: "testCat"
## with 2 features
## It has 4 fields
```

```
# cat is the catchment file.
```

Then the catchment file `cat` can be inputed as background.

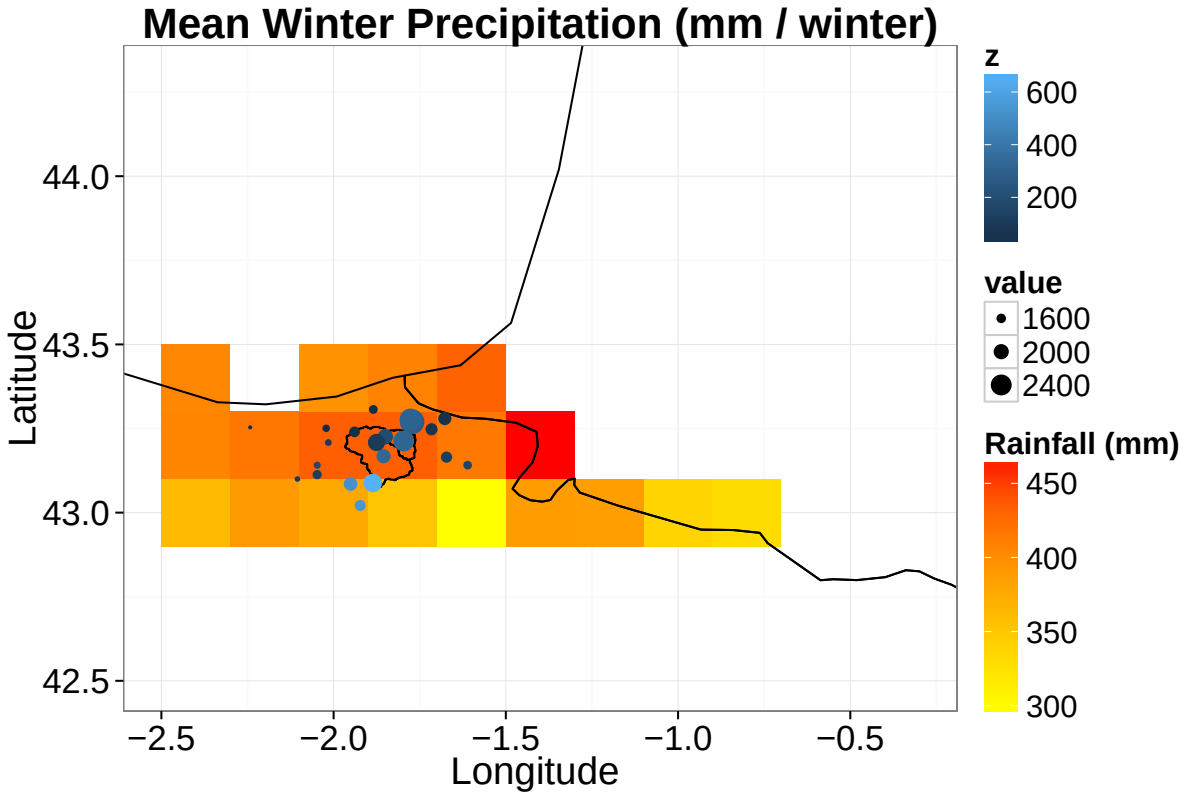
```
a <- getSpatialMap(tgridData, method = 'meanAnnual', catchment = cat)
```



2.3.2 Add station locations

Points file needs to be read into dataframe, and special column has to be assigned, details can be found by `?getSpatialMap_mat`

```
# Use the points file provided by hyfo
file <- system.file("extdata", "point.txt", package = "hyfo")
point <- read.table(file, header = TRUE, sep = ',' )
getSpatialMap(tgridData, method = 'winter', point = point, catchment = cat)
```



As can be seen above, the color of the points represents the elevation, the size of the points represents the value, e.g., rainfall value.

You can generate your own point file and use it as background, or you can also find the original file in the package, and replace the old information with your information.

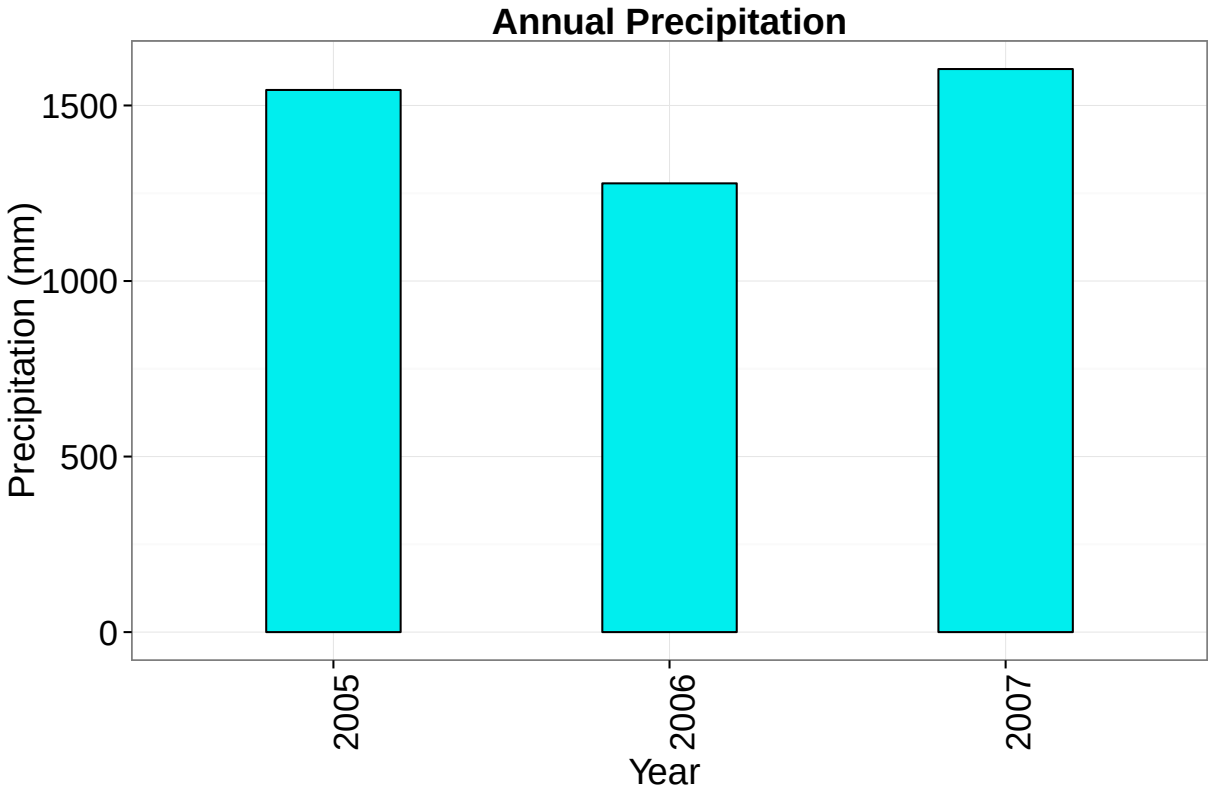
2.4 Variable Bar Plot

Bisides spatial map, bar plot can also be plotted. The value in the bar plot is spatially averaged, i.e. the value in the bar plot is the mean value over the region.

Annual precipitation.

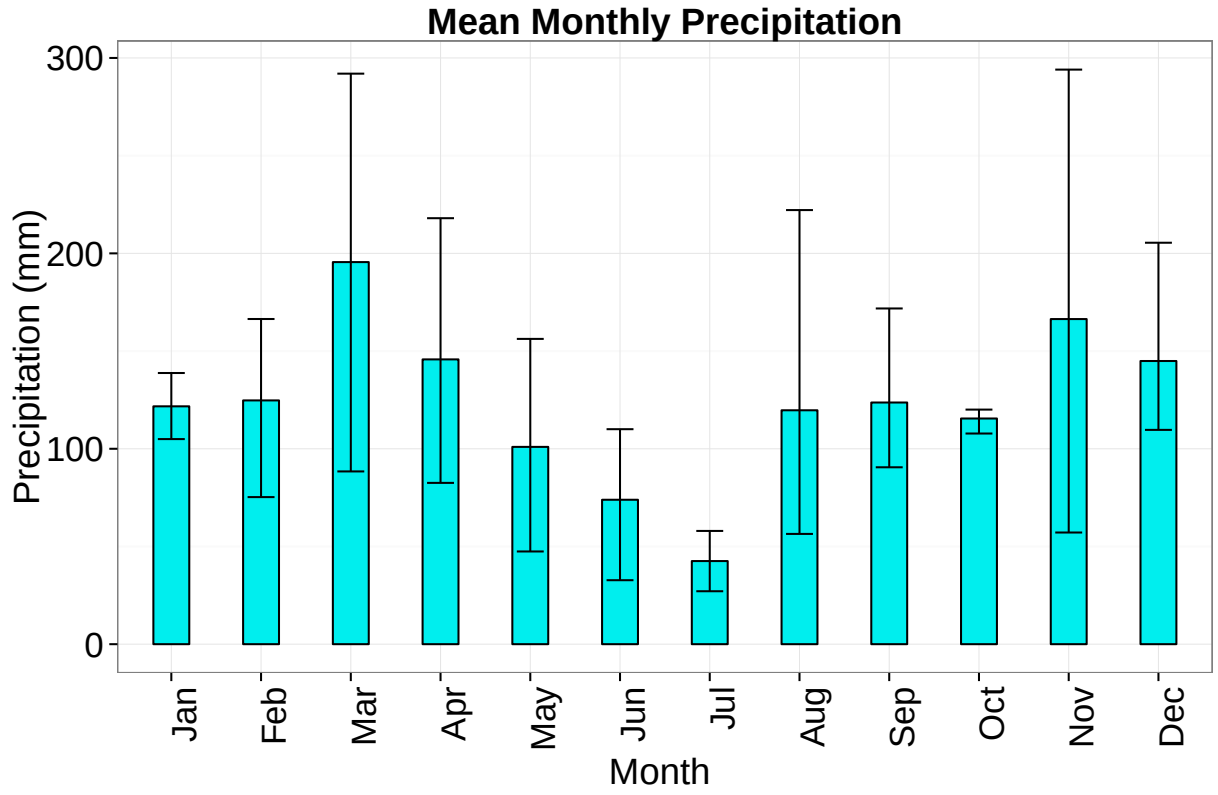
```
data(tgridData)
a <- getPreciBar(tgridData, method = 'annual')
```

```
## There is no plotRange for this method
```

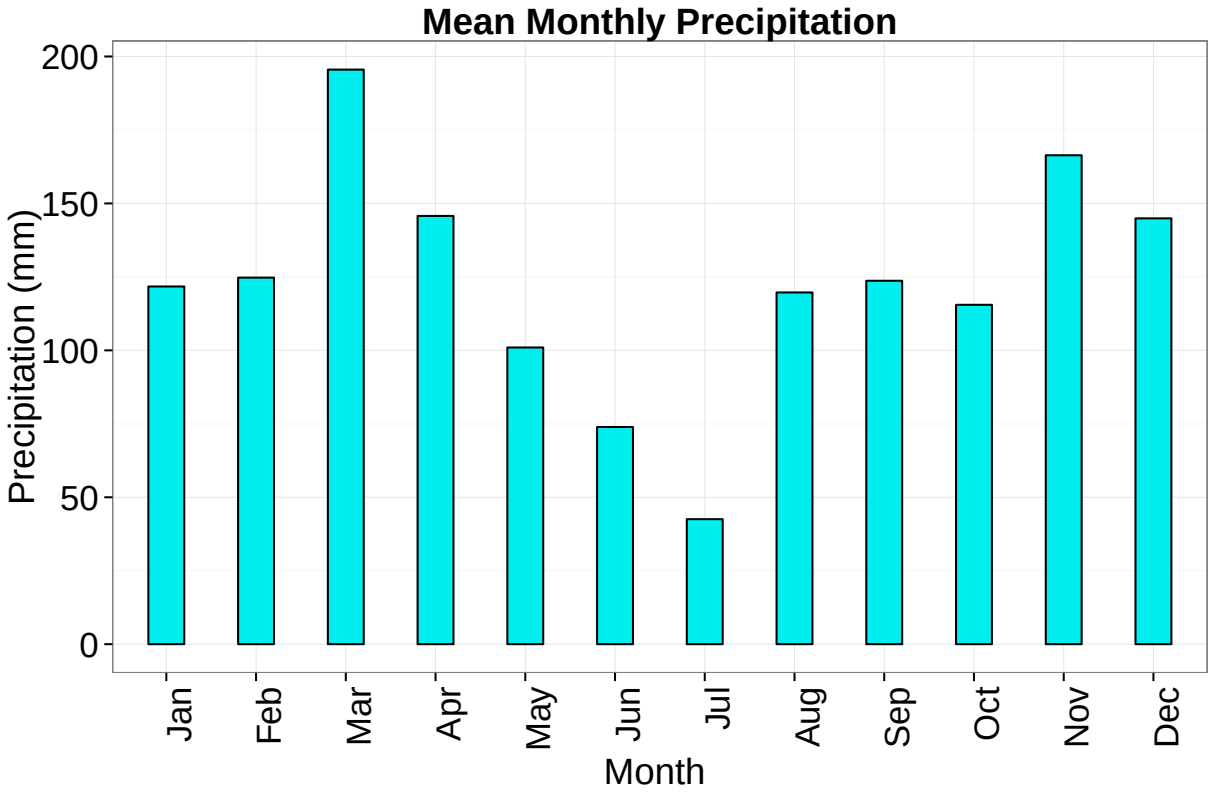


Mean monthly precipitation over the whole period, with the ranges for each month. But not all kinds of bar plot have a plot range.

```
a <- getPreciBar(tgridData, method = 'meanMonthly')
```



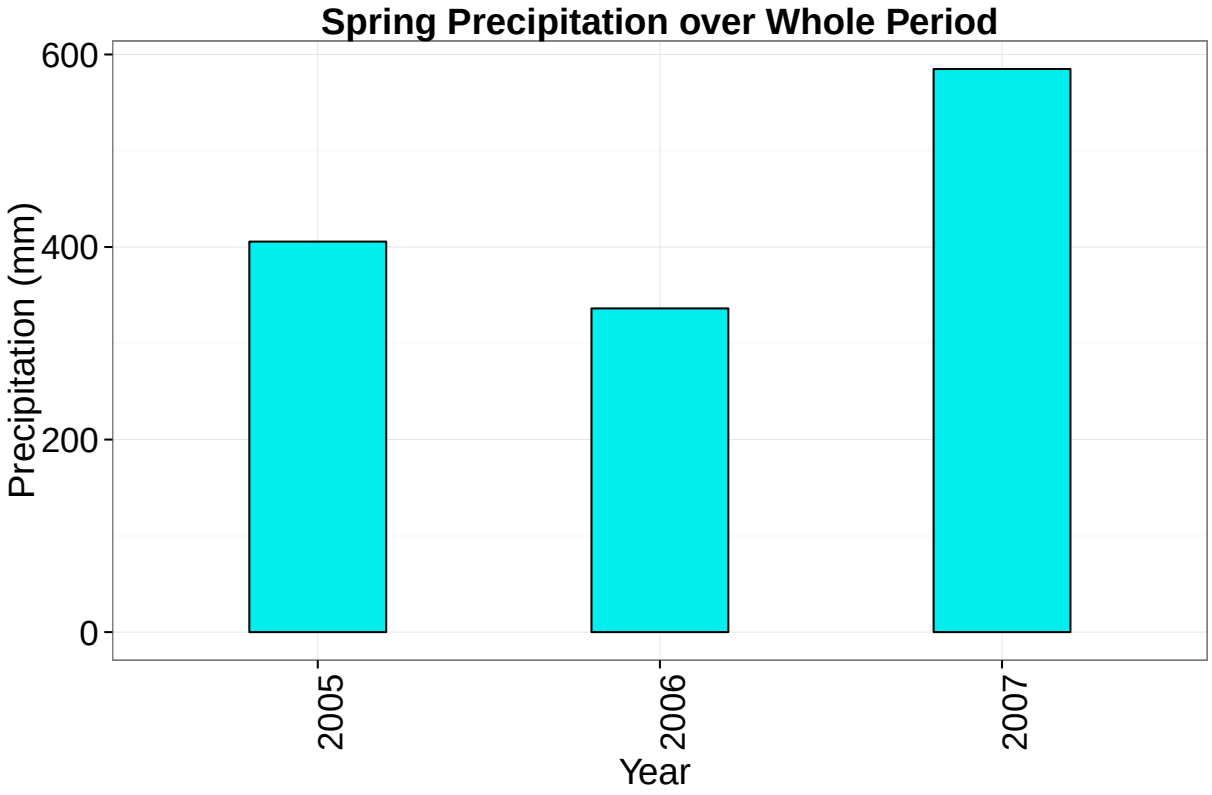

```
a <- getPreciBar(tgridData, method = 'meanMonthly', plotRange = FALSE)
```



Seasonal precipitation, and monthly precipitation can also be plotted.

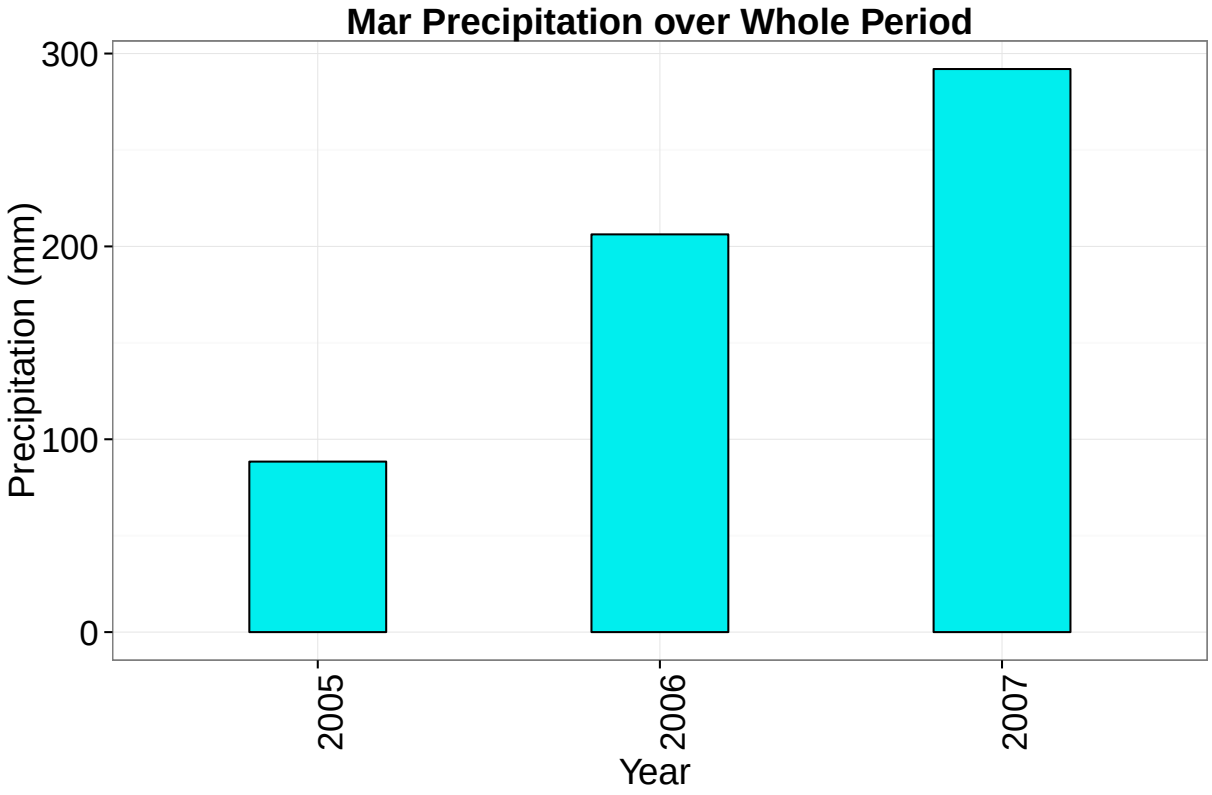
```
a <- getPreciBar(tgridData, method = 'spring')# spring precipitation for each year
```

```
## There is no plotRange for this method
```




```
a <- getPreciBar(tgridData, method = 3) # march precipitation for each year
```

```
## There is no plotRange for this method
```



2.5 Bias Correction

Usually climate forecasting is based on global scale. if it is downscaled to certain research area, there may exist some bias. In order to get rid of the bias, forecasts needs to be bias-corrected.

hyfo provides bias correction for both grid data and time series, among which time series bias correction can generate time series output available for model input. More Details and principles behind the bias correction and more biasCorrection methods can be found by type in `?biasCorrect`

No need to designate input type, R will detect automatically.

For hyfo grid file bias correction.

```
######## hyfo grid file biascorrection
########

# If your input is obtained by \code{loadNcdf}, you can also directly biascorrect
# the file.

# First load ncdf file.
filePath <- system.file("extdata", "tnc.nc", package = "hyfo")
varname <- getNcdfVar(filePath)
nc <- loadNcdf(filePath, varname)
```

```
## Loading data...
## Processing...
```

```
data(tgridData)
# Since the example data, has some NA values, the process will include some warning     # message, whic

# Then we will use nc data as forecasting data, and use itself as hindcast data,
# use tgridData as observation.
newFrc <- biasCorrect(nc, nc, tgridData)
newFrc <- biasCorrect(nc, nc, tgridData, scaleType = 'add')
newFrc <- biasCorrect(nc, nc, tgridData, method = 'eqm', extrapolate = 'constant',
preci = TRUE)
newFrc <- biasCorrect(nc, nc, tgridData, method = 'gqm', preci = TRUE)
```

For time series bias correciton.

```
# Use testdl as an example, we take frc, hindcast and obs from testdl.
data(testdl)

# common period has to be extracted in order to make them have the same time period.

datalist <- extractPeriod(testdl, startDate = '1994-1-1', endDate = '1995-10-1')

frc <- datalist[[1]]
hindcast <- datalist[[2]]
obs <- datalist[[3]]

# default method is delta
frc_new <- biasCorrect(frc, hindcast, obs)

# If precipitation data is input, than further process needs to be done with the data,
# that you only have to simply add one argument

frc_new <- biasCorrect(frc, hindcast, obs, preci = TRUE)


# For different bias correction method.
frc_new <- biasCorrect(frc, hindcast, obs, method = 'scaling', scaleType = 'multi')

frc_new <- biasCorrect(frc, hindcast, obs, method = 'eqm', scaleType = 'constant')

# because gqm only applys to precipitation biascorrection, you have to set 'preci = TRUE'
frc_new <- biasCorrect(frc, hindcast, obs, method = 'gqm', preci = TRUE)
```

If the forecasts you extracted only has incontinuous data for certain months and years, e.g., for seasonal forecasting, forecasts only provide 3-6 months data, so the case can be for example Dec, Jan and Feb of every year from year 1999-2005. In such case, you need to extract certain months and years from observed time series, `extractPeriod()` (section 1.3.1) can be then used.

2.5.1 Multi/Operational/Real Time Bias Correction

When you do multi/operational/real time bias correction. It's too expensive to input hindcast and obs every time. Especially when you have a long period of hindcast and obs, but only a short period of frc, it's too unecessary to read and compute hindcast and obs everytime. Therefore, biasFactor is designed. Using `getBiasFactor`, you can get the biasFactor with hindcast and observation, then you can use `applyBiasFactor` to apply the biasFactor to different forecasts.

Details can be found in `?getBiasFactor` or `?applyBiasFactor`, for bias correction methods, details can be found in `?biasCorrect`.

For hyfo grid file real time bias correction.

```
######## hyfo grid file biascorrection
########

# If your input is obtained by \code{loadNcdf}, you can also directly biascorrect
# the file.

# First load ncdf file.
filePath <- system.file("extdata", "tnc.nc", package = "hyfo")
varname <- getNcdfVar(filePath)
nc <- loadNcdf(filePath, varname)
```

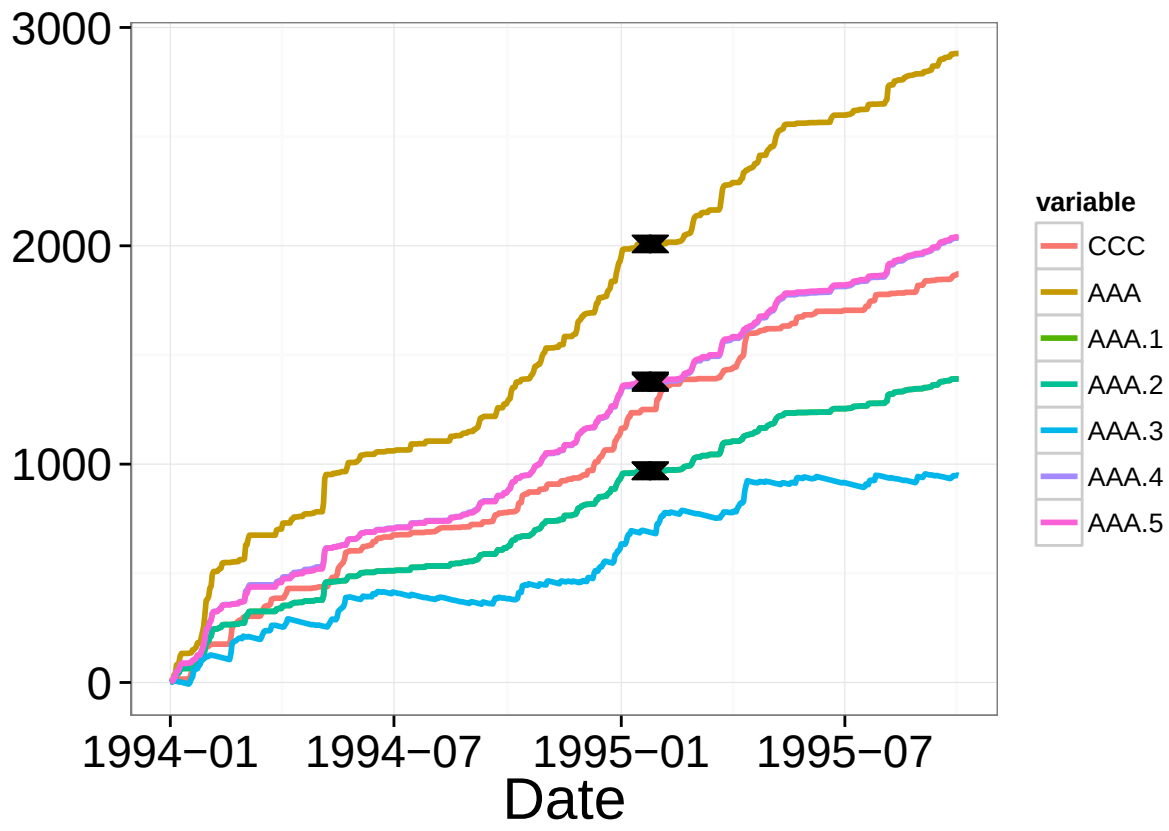
```
## Loading data...
## Processing...
```

```
data(tgridData)
# Since the example data, has some NA values, the process will include some warning #message, which can

# Then we will use nc data as forecasting data, and use itself as hindcast data,
# use tgridData as observation.

biasFactor <- getBiasFactor(nc, tgridData)
newFrc <- applyBiasFactor(nc, biasFactor)

biasFactor <- getBiasFactor(nc, tgridData, method = 'eqm', extrapolate = 'constant',
preci = TRUE)
# This method needs obs input.
newFrc <- applyBiasFactor(nc, biasFactor, obs = tgridData)

biasFactor <- getBiasFactor(nc, tgridData, method = 'gqm', preci = TRUE)
newFrc <- applyBiasFactor(nc, biasFactor)
```

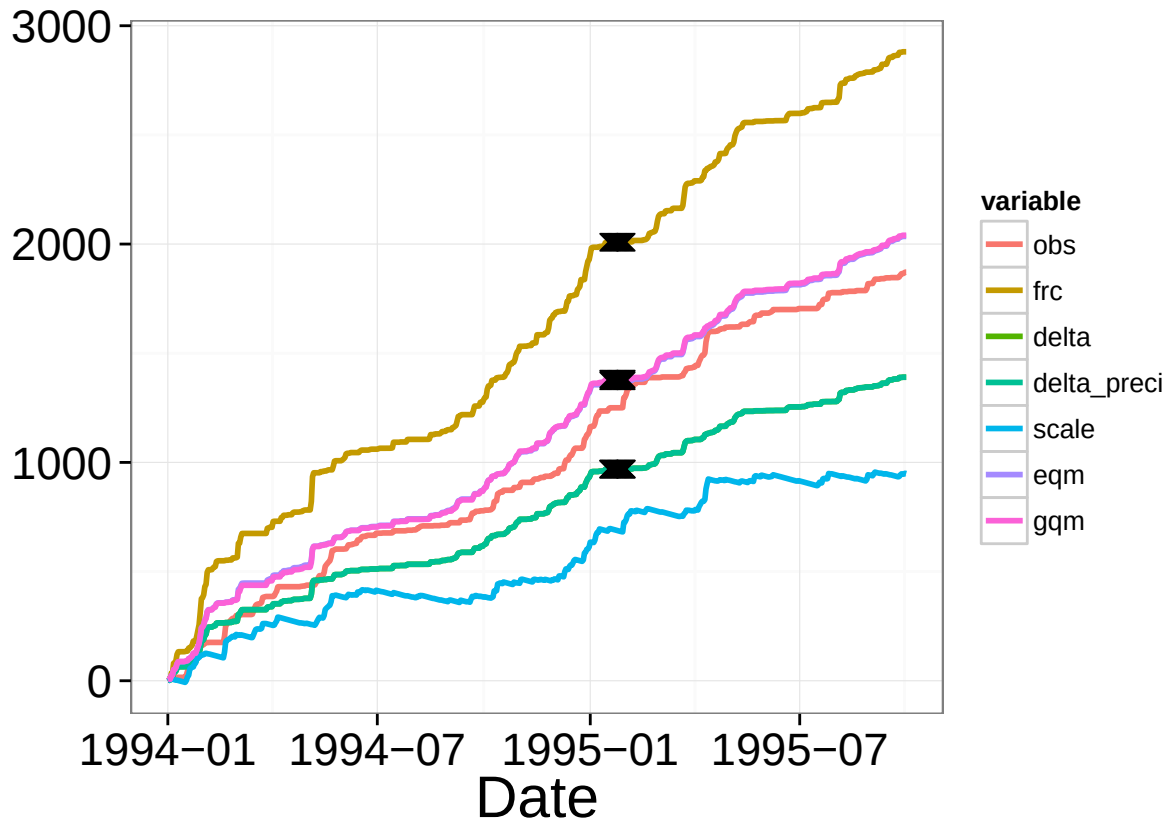
For time series real time bias correction.

```
######## Time series biascorrection
########

# Use the time series from testdl as an example, we take frc, hindcast and obs from #testdl.
data(testdl)

# common period has to be extracted in order to better train the forecast.
datalist <- extractPeriod(testdl, startDate = '1994-1-1', endDate = '1995-10-1')

frc <- datalist[[1]]
hindcast <- datalist[[2]]
obs <- datalist[[3]]

# The data used here is just for example, so there could be negative data.
```

```r
# default method is scaling
biasFactor <- getBiasFactor(hindcast, obs)
frc_new <- applyBiasFactor(frc, biasFactor)

# for precipitation data, extra process needs to be executed, so you have to tell
# the program to it is a precipitation data.

biasFactor <- getBiasFactor(hindcast, obs, preci = TRUE)
frc_new1 <- applyBiasFactor(frc, biasFactor)

# You can use other methods to biascorrect, e.g. delta method.
biasFactor <- getBiasFactor(hindcast, obs, method = 'delta')
# delta method needs obs input.
frc_new2 <- applyBiasFactor(frc, biasFactor, obs = obs)

biasFactor <- getBiasFactor(hindcast, obs, method = 'eqm', preci = TRUE)
# eqm needs obs input
frc_new3 <- applyBiasFactor(frc, biasFactor, obs = obs)

biasFactor <- getBiasFactor(hindcast, obs, method = 'gqm', preci = TRUE)
frc_new4 <- applyBiasFactor(frc, biasFactor)

plotTS(obs, frc, frc_new, frc_new1, frc_new2, frc_new3, frc_new4, plot = 'cum')
```



```
# You can also give name to this input list.
TSlist <- list(obs, frc, frc_new, frc_new1, frc_new2, frc_new3, frc_new4)
names(TSlist) <- c('obs', 'frc', 'delta', 'delta_preci', 'scale', 'eqm', 'gqm')
plotTS(list = TSlist, plot = 'cum')
```



If the forecasts you extracted only has incontinuous data for certain months and years, e.g., for seasonal forecasting, forecasts only provide 3-6 months data, so the case can be for example Dec, Jan and Feb of every year from year 1999-2005. In such case, you need to extract certain months and years from observed time series , then you can use the extract tool `extractPeriod()` to do the job.

2.6 Analysis and Comparison

For some cases, analysis and comparison are necesssary, which are also provided by `hyfo`.

There are three different kinds of output from `getSpatialMap` and `getPreciBar`, respectively, `output = 'data'`, `output = 'ggplot'` and `output = 'plot'`.

`output = 'data'` is default in the function and do not need to be declare when input. It is mainly used in analyzing and replot the results.

`output = 'ggplot'` is used when combining different plots.

`output = 'plot'` is used when a layer output is needed. the output can be directly printed, and can be mannually combined by the plot arrange functions, e.g., `grid.arrange()`

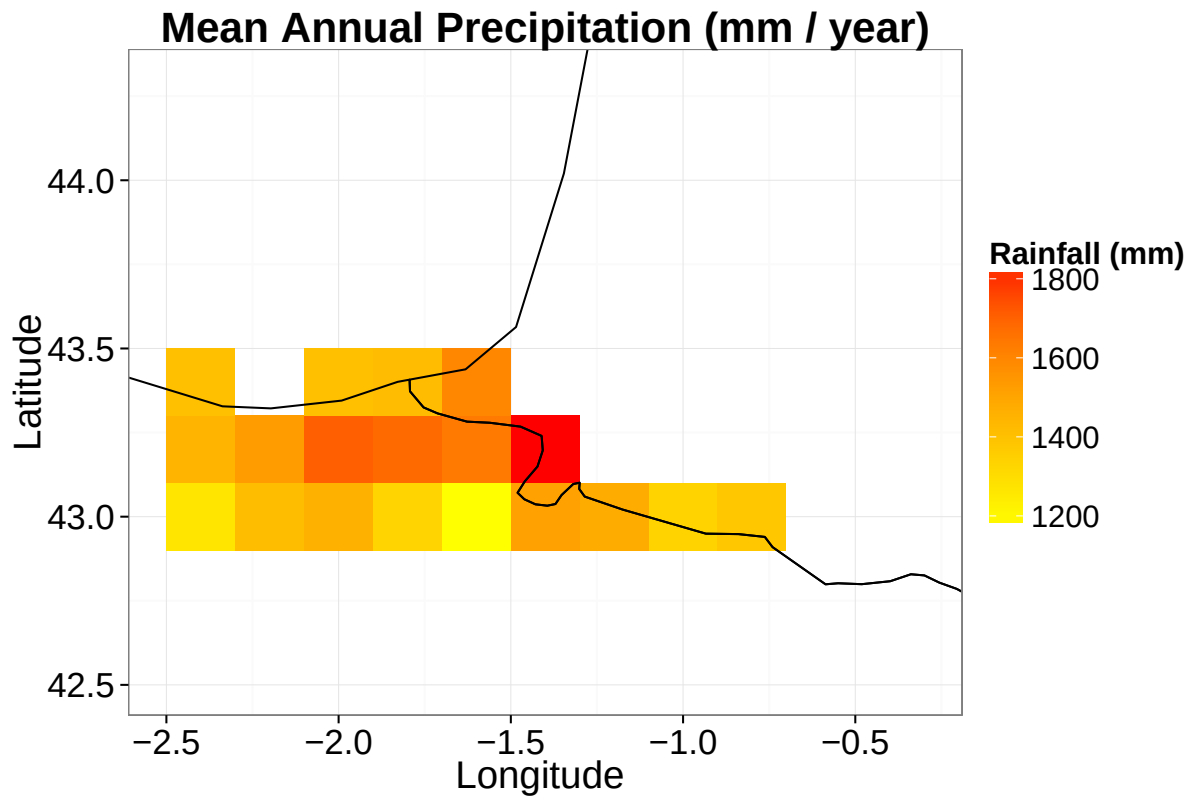
Note: All the comparisons must be comparable, e.g.,

- For `getSpatialMap_comb`, the maps to be compared should be with same size and resolution, in other words, they should be fully overlapped by each other. Check `?getSpatialMap_comb` for details.
- For `getPreciBar_comb`, the bar plots to be compared should belong to the same kind, e.g., spring and winter, January and December, and couldn't be spring and annual. Details can be found by `?getPreciBar_comb`

2.6.1 Spatial Map

The default "data" output provides a matrix, representing the raster information of the spatial map.

```
a <- getSpatialMap(tgridData, method = 'meanAnnual')
```



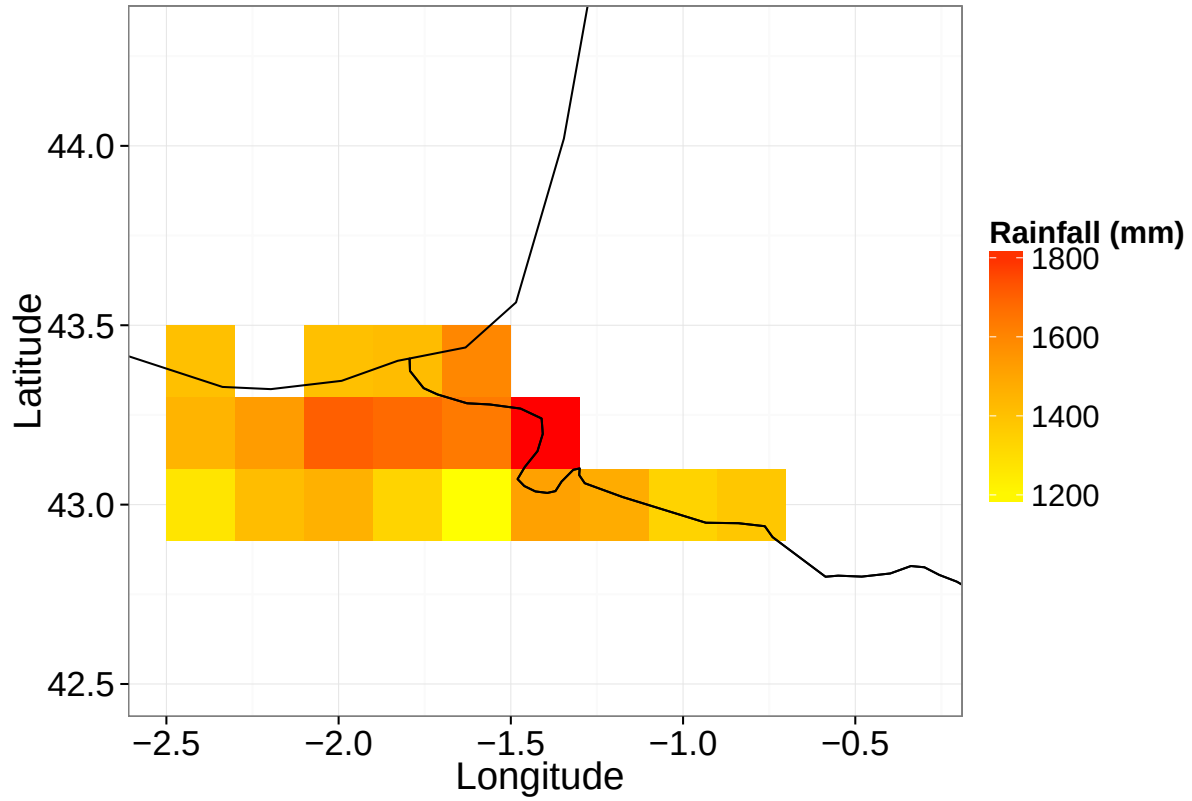

```
a
```

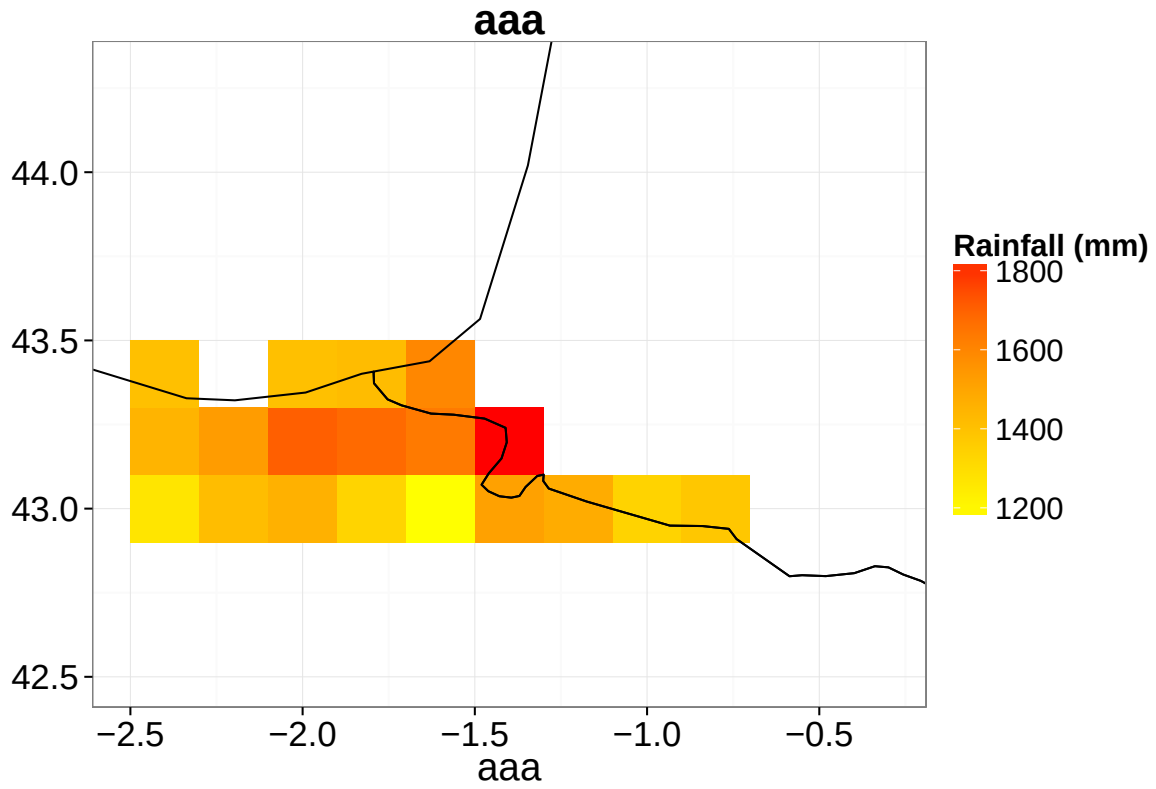
```
##          -2.4     -2.2       -2     -1.8     -1.6     -1.4     -1.2
## 44.2       NA       NA       NA       NA       NA       NA       NA
## 44         NA       NA       NA       NA       NA       NA       NA
## 43   1265.663 1414.792 1459.179 1331.803 1167.537 1515.733 1476.697
## 43.2 1449.765 1533.385 1711.032 1683.085 1638.575 1840.649       NA
## 43.4 1406.753       NA 1404.264 1420.504 1601.129       NA       NA
## 43.6       NA       NA       NA       NA       NA       NA       NA
## 43.8       NA       NA       NA       NA       NA       NA       NA
## 42.8       NA       NA       NA       NA       NA       NA       NA
## 42.6       NA       NA       NA       NA       NA       NA       NA
##            -1     -0.8 -0.6 -0.4
```

```
## 44.2       NA       NA   NA   NA
## 44         NA       NA   NA   NA
## 43   1334.274 1377.036   NA   NA
## 43.2       NA       NA   NA   NA
## 43.4       NA       NA   NA   NA
## 43.6       NA       NA   NA   NA
## 43.8       NA       NA   NA   NA
## 42.8       NA       NA   NA   NA
## 42.6       NA       NA   NA   NA
```

This matrix is upside down from what you can see from the plot. **DO NOT try to change this matrix.** `hyfo` can deal with it.

```
# For re-plot the matrix, input the matrix, and then the map can be replot.
b <- getSpatialMap_mat(a)

# Without title and x and y, also you can assign yourself.
b <- getSpatialMap_mat(a, title = 'aaa', x = 'aaa', y = '')
```

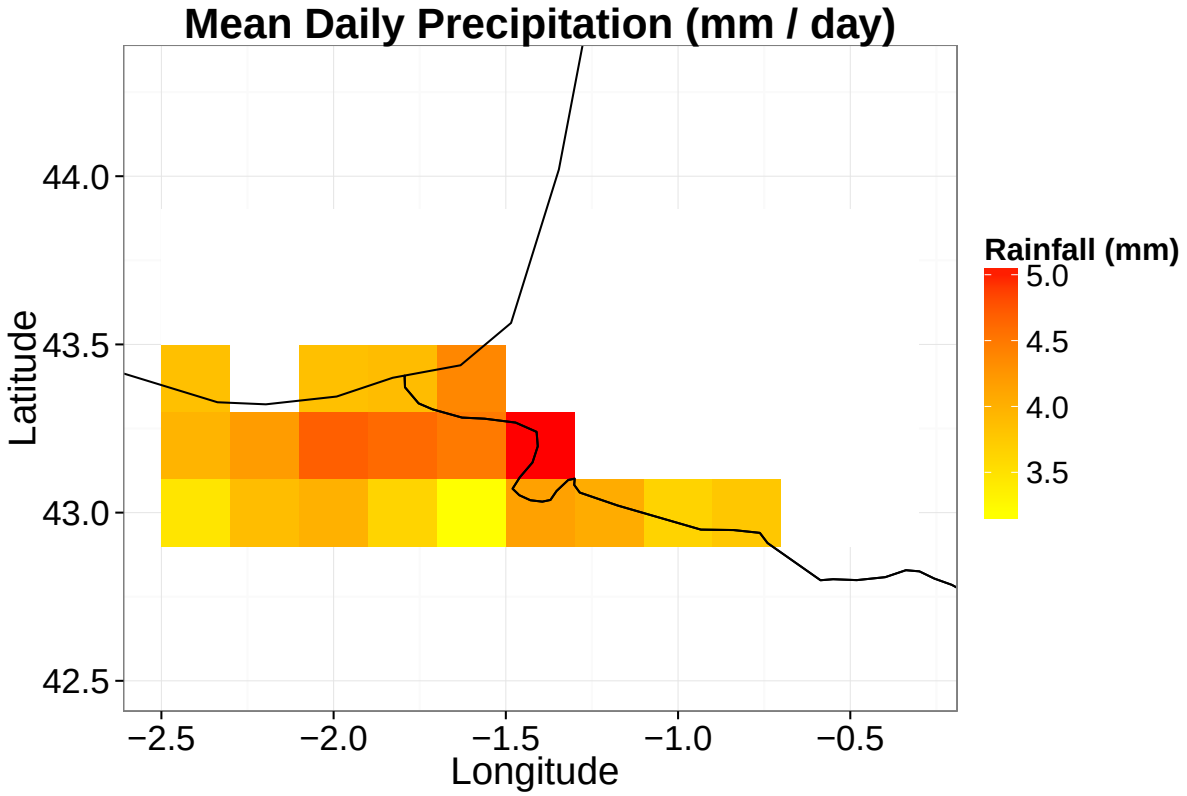


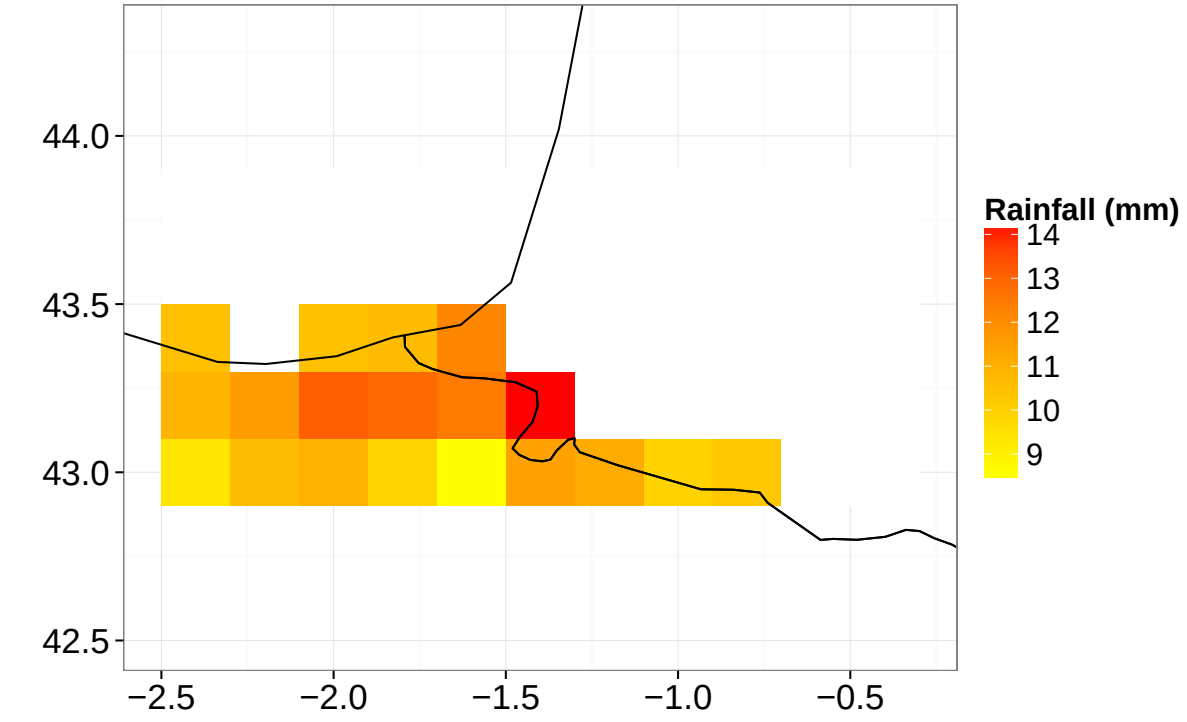


The matrix can be used to make different analysis and plot again.

Note If the matrix doesn't come from `getSpatialMap`, dimension name of longitude and latitude needs to be provided to the matrix, in order to be plotted.

```
a1 <- getSpatialMap(tgridData, method = 'mean')

# To make some changes to mean value.
b <- a1 * 3 -1
getSpatialMap_mat(b, title = '', x = '', y = '')

# Bias, variation and other analysis can also be processed
# the same way.
# Just apply the analysis to the matrix and
# use getSpatialMap_mat to plot.
```





If multi-plot is needed, `hyfo` can also combine different plots together. Use `output = ggplot`, which gives

back the a special format that can be easily used by `ggplot2`

```
a1 <- getSpatialMap(tgridData, method = 'spring', output = 'ggplot', name = 'spring')
```

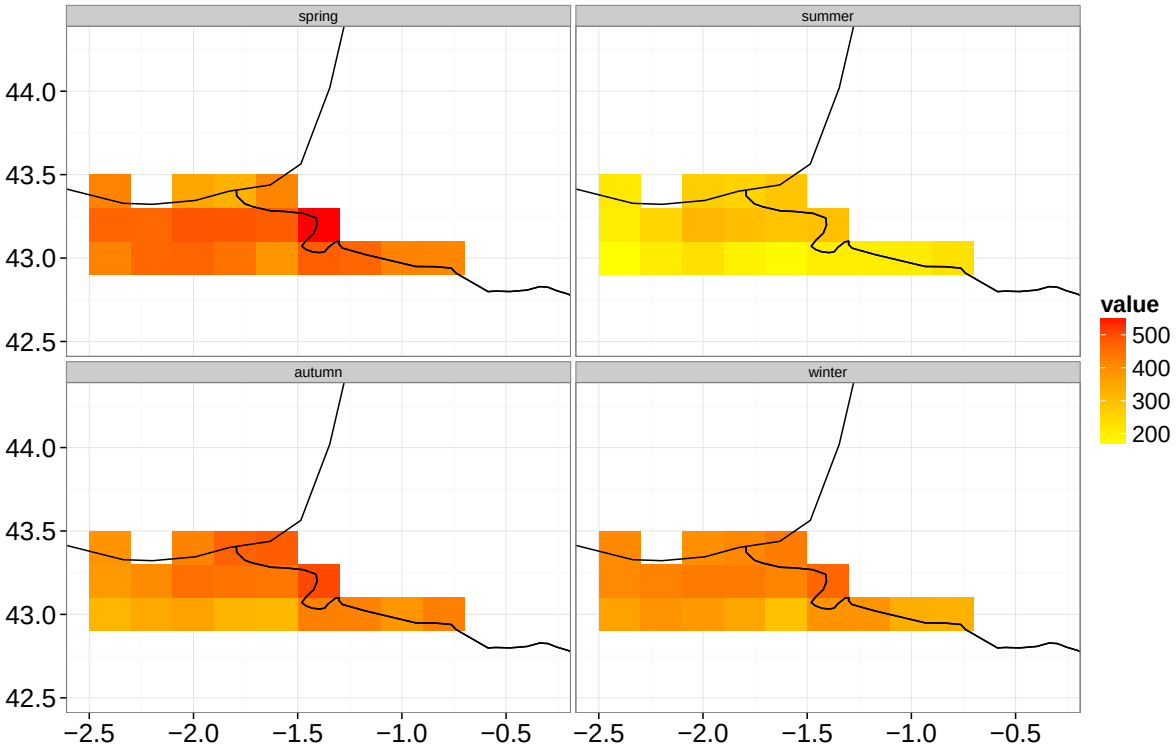
```
a2 <- getSpatialMap(tgridData, method = 'summer', output = 'ggplot', name = 'summer')
```

```
a3 <- getSpatialMap(tgridData, method = 'autumn', output = 'ggplot', name = 'autumn')
```

```
a4 <- getSpatialMap(tgridData, method = 'winter', output = 'ggplot', name = 'winter')
```

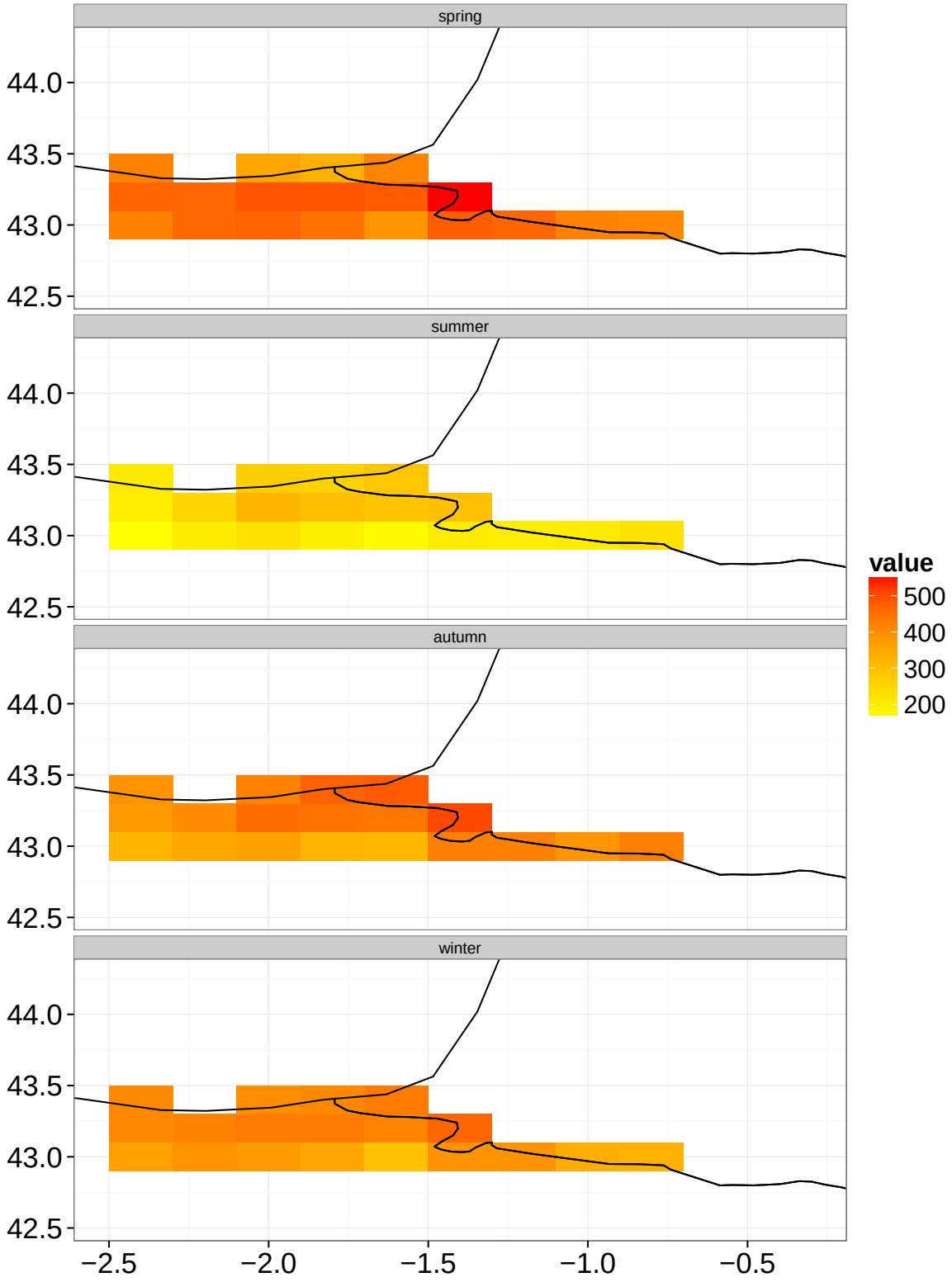
```
getSpatialMap_comb(a1, a2, a3, a4, nrow = 2)# you cannot assign title
```

```
## Check if the data list is available for rbind or cbind...
##
## Data list is OK
```




```
getSpatialMap_comb(a1, a2, a3, a4, nrow = 4)
```

```
## Check if the data list is available for rbind or cbind...
##
## Data list is OK
```



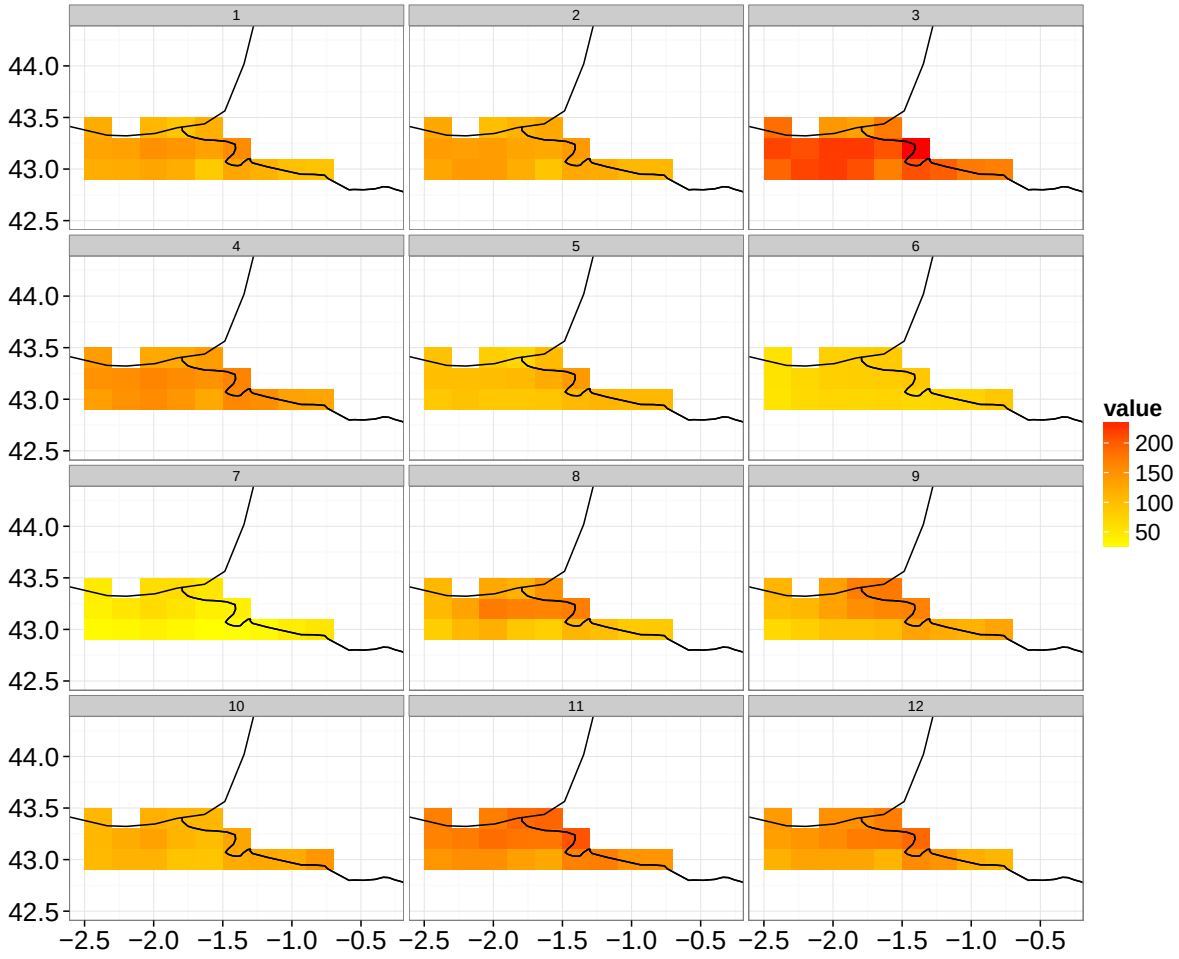
`getSpatialMap_comb` accepts list (using `list =`) object too, which is easier for multi-plot. First list of 12

months are got. **NOTE: If input is a list, the argument should be `list = yourlist`, not directly put the list in the argument.**

```
c <- lapply(1:12, function(x) getSpatialMap(tgridData, method = x, output = 'ggplot',
                                            name = x))
```

Then they are combined.

```
getSpatialMap_comb(list = c, nrow = 4)
```



2.6.2 Bar Plot

Basically, bar plot follows the same rule as part 2.4.1 spatial map, only a few cases that needs to pay attention.

```
b1 <- getPreciBar(tgridData, method = 'spring', output = 'ggplot', name = 'spring')
```

```
b2 <- getPreciBar(tgridData, method = 'summer', output = 'ggplot', name = 'summer')
```

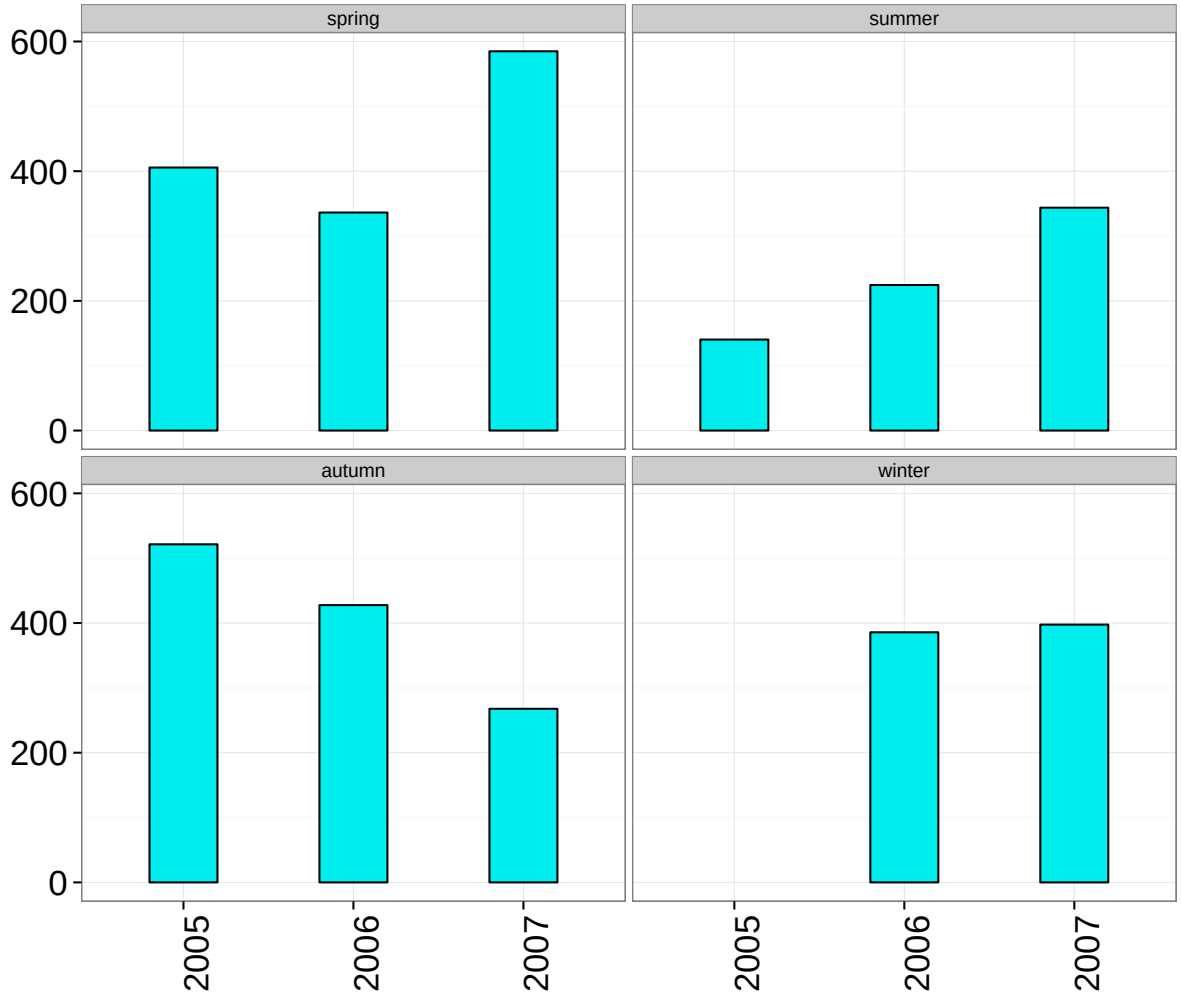
```
b3 <- getPreciBar(tgridData, method = 'autumn', output = 'ggplot', name = 'autumn')
```

```
b4 <- getPreciBar(tgridData, method = 'winter', output = 'ggplot', name = 'winter')
```

```
## Warning: Removed 1 rows containing missing values (position_stack).
```

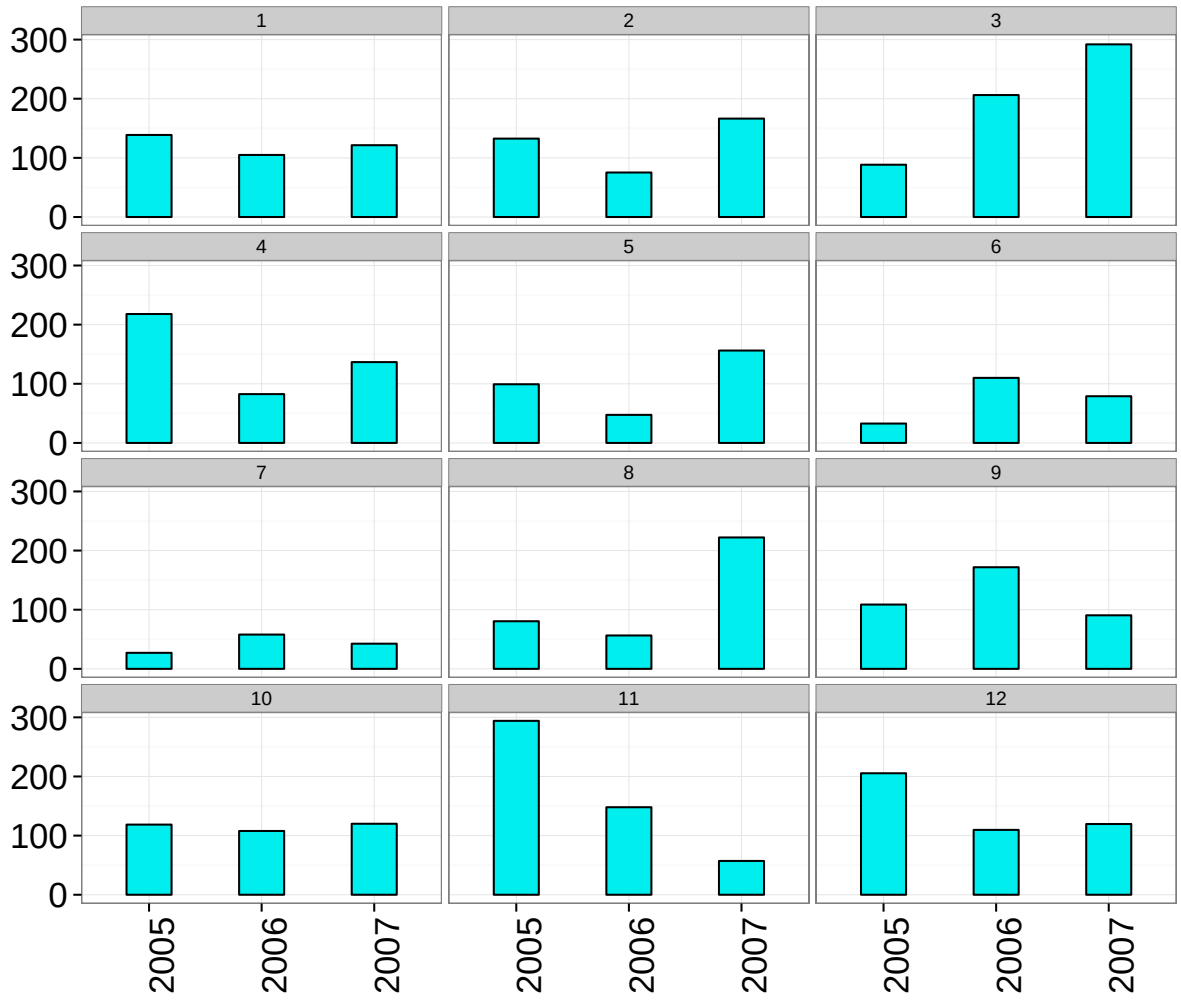
```
getPreciBar_comb(b1, b2, b3, b4, nrow = 2)
```

```
## Check if the data list is available for rbind or cbind...
##
## Data list is OK
```




```
c <- lapply(1:12, function(x) getPreciBar(tgridData, method = x, output = 'ggplot',
                                          name = x))
```

```
getPreciBar_comb(list = c, nrow = 4)
```



2.7 Model Input

2.7.1 Extract time series from Forecasting Dataset

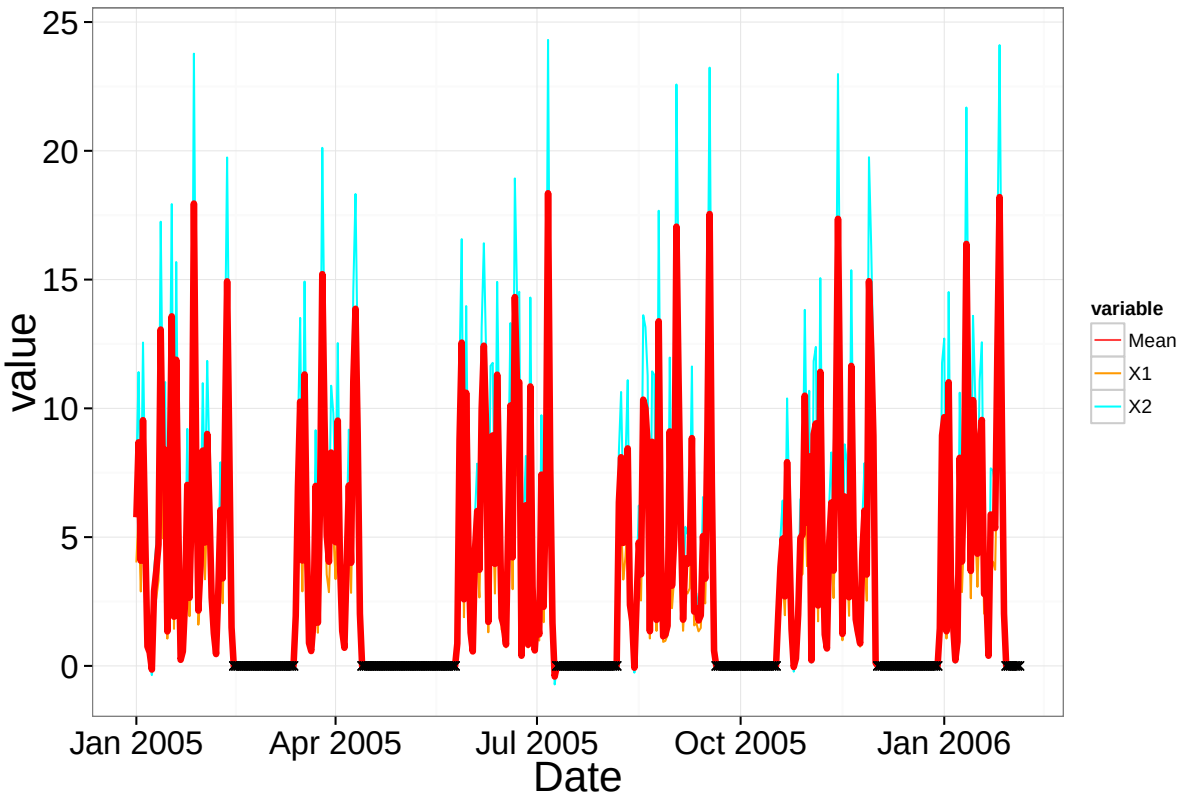
If there are different members existing in the dataset, `hyfo` can extract them and generate a dataframe for the easy input to the model. If the dataset doesn't have a member part, then `hyfo` will extract only one single time seres.

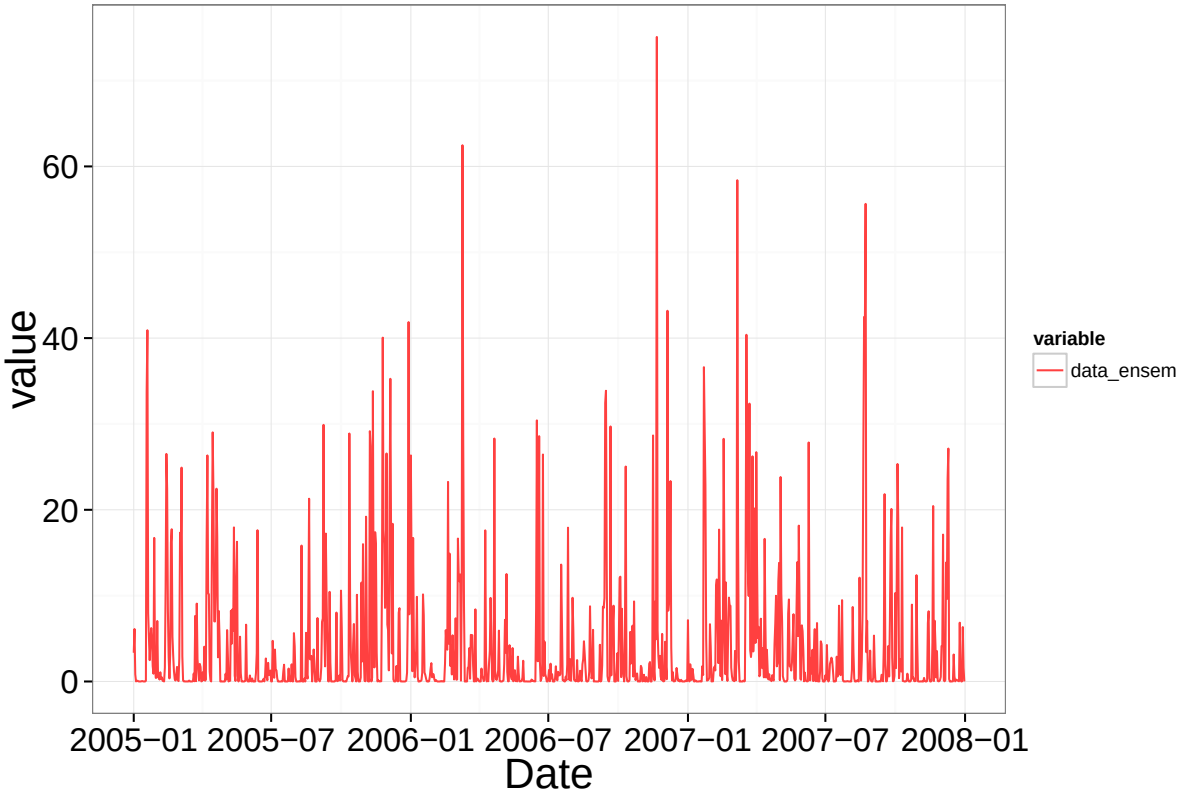
```
filePath <- system.file("extdata", "tnc.nc", package = "hyfo")

# Then if you don't know the variable name, you can use \code{getNcdfVar} to
# get variable name
varname <- getNcdfVar(filePath)

nc <- loadNcdf(filePath, varname)
```

```
## Loading data...
## Processing...
```

```
a <- getFrcEnsem(nc)


# If there is no member session in the dataset, a single time sereis will be
# extracted.

a1 <- getFrcEnsem(tgridData)
```

```
## There is no member part in the dataset, there will be only one column of value
##              returned.
```

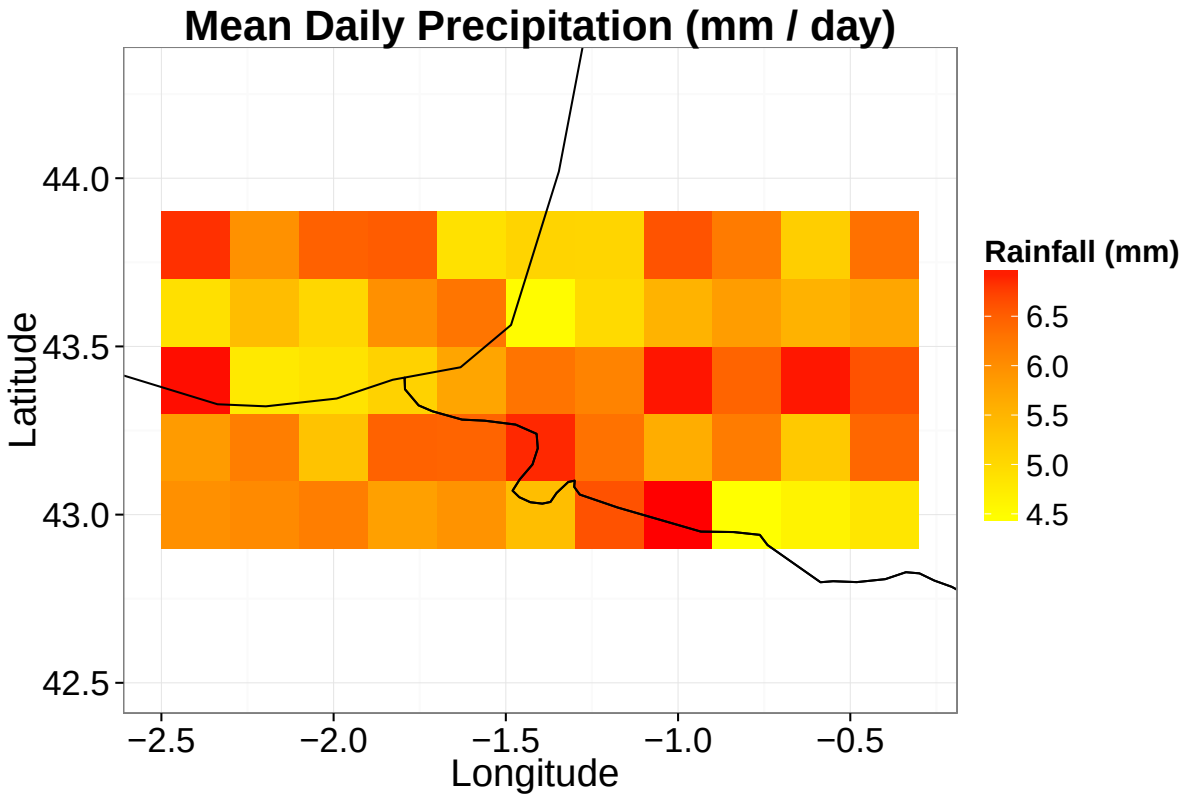




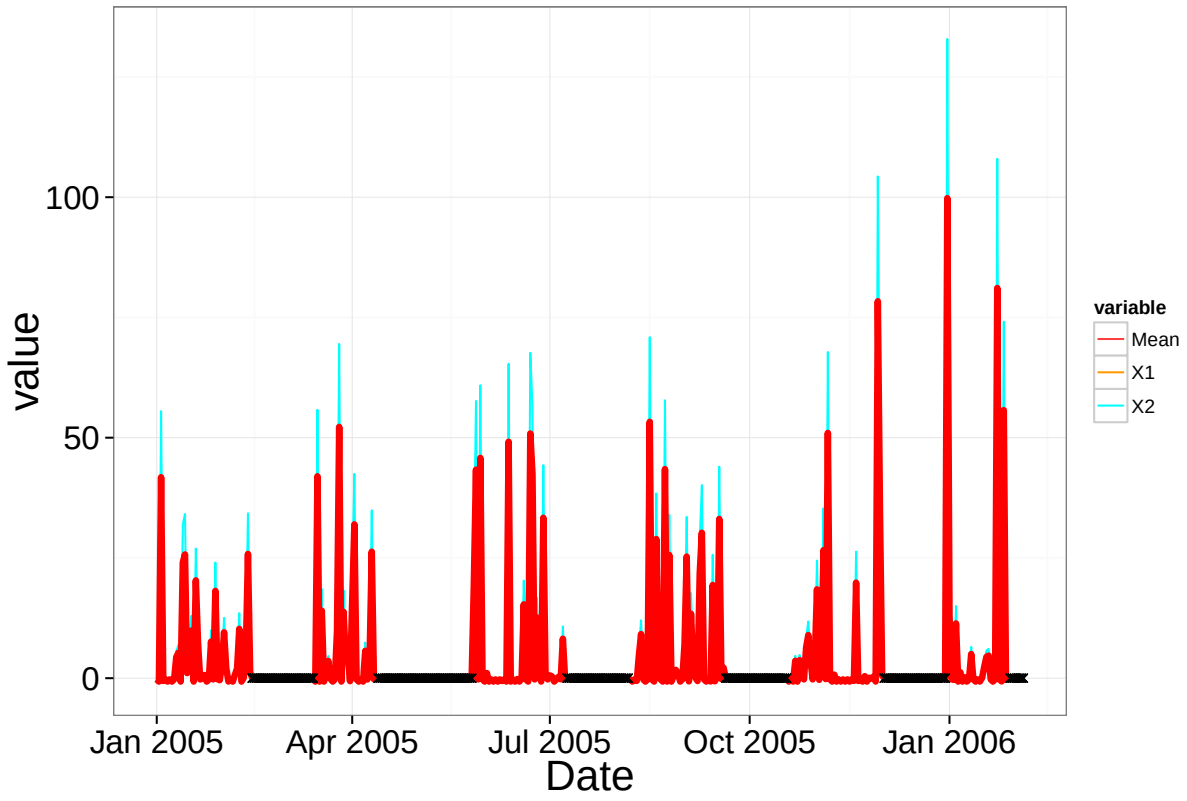
The default output is spatially averaged, if there are more than one cells in the dataset, the mean value of the cells will be calculated. While if you are interested in special cell, you can assign the cell value, for how to assign, please check the details in `?getFrcEnsem`. You can also directly use longitude and latitude to extract time series, using `coord =`

```
getSpatialMap(nc, 'mean')
```

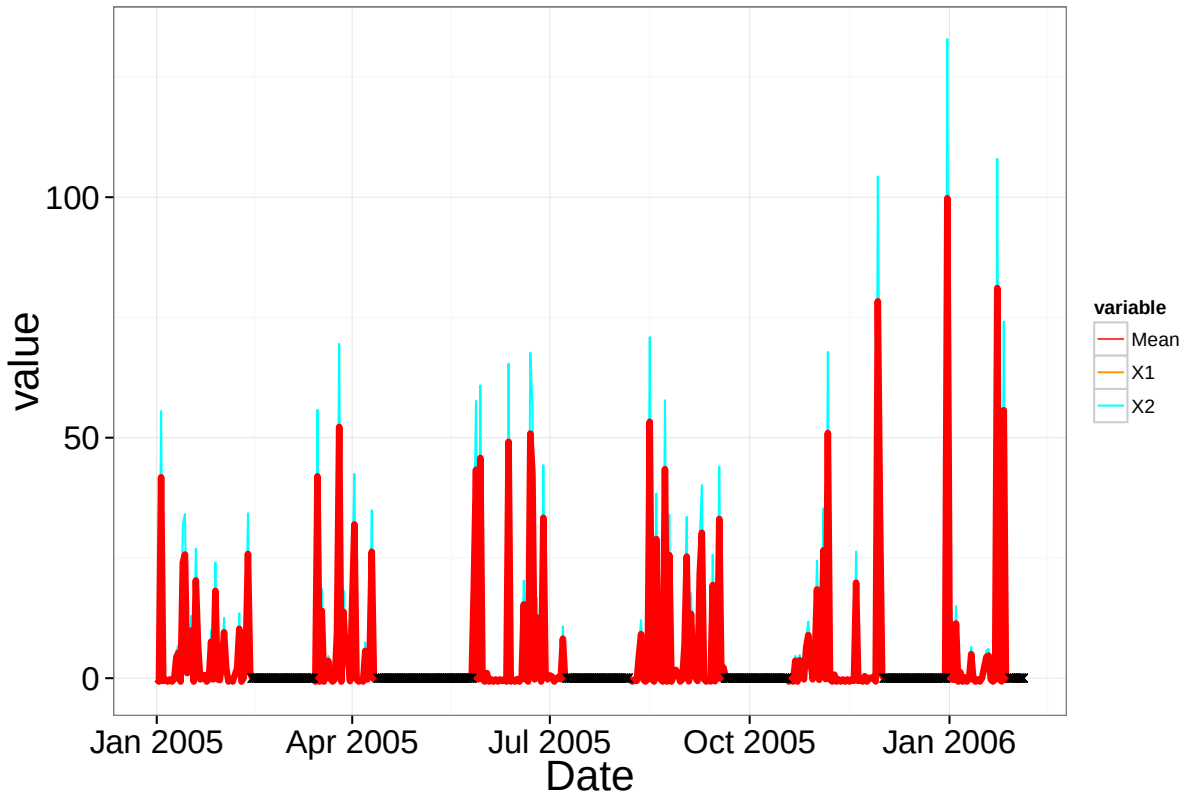
```
## Mean value of the members are returned.
```




```
a <- getFrcEnsem(nc, cell = c(6,2))
```




```
# From the map, cell = c(6, 2) means lon = -1.4, lat = 43.2, so you can use
# corrd to locate your research area and extract time series.
b <- getFrcEnsem(nc, coord = c(-1.4, 43.2))
```



If you want to combine different plots together, you can use `_comb` function to combine plots.

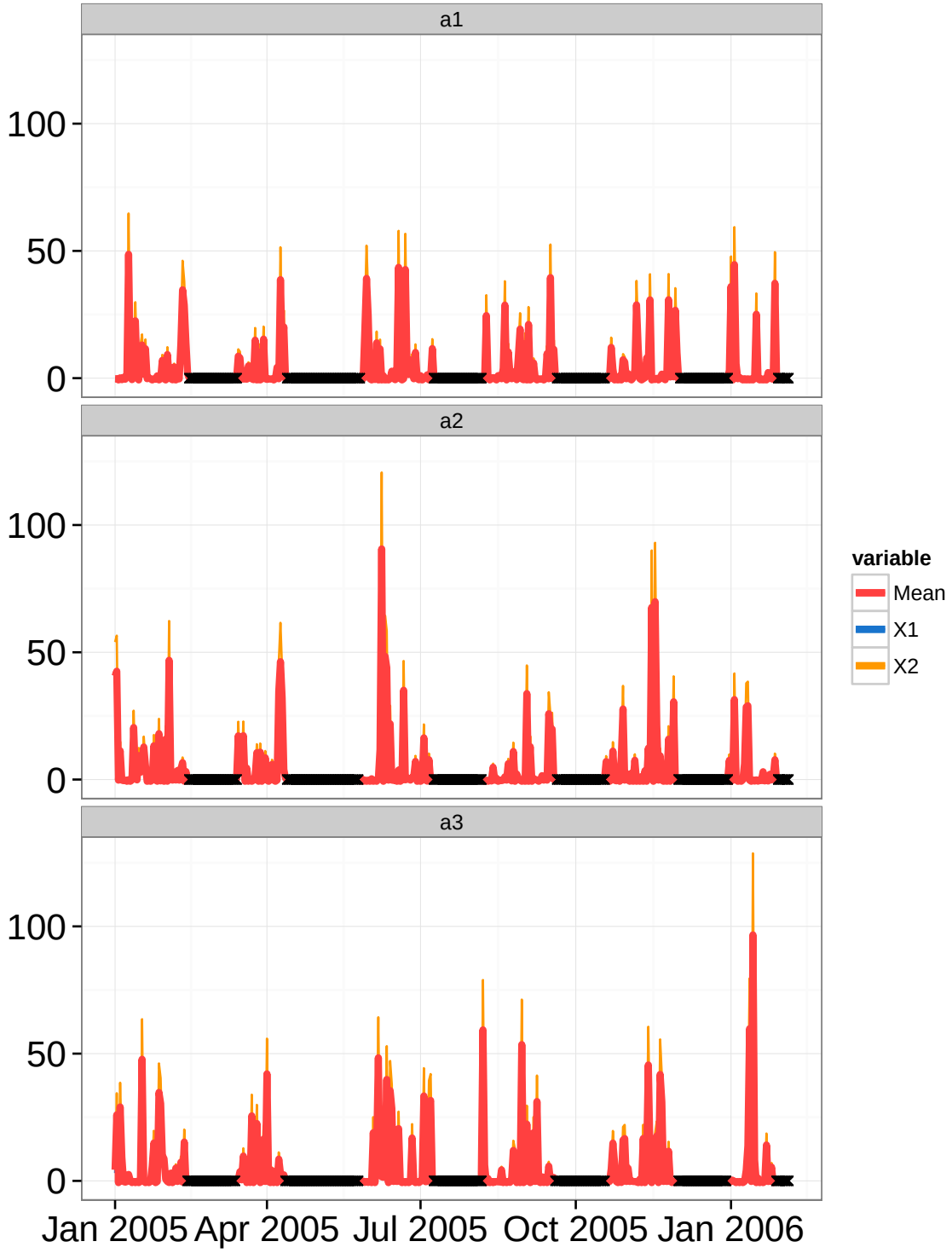
```
a1 <- getFrcEnsem(nc, cell = c(2,3), output = 'ggplot', name = 'a1')
```

```
a2 <- getFrcEnsem(nc, cell = c(2,4), output = 'ggplot', name = 'a2')
```

```
a3 <- getFrcEnsem(nc, cell = c(2,5), output = 'ggplot', name = 'a3')
```

```
getEnsem_comb(a1, a2, a3, nrow = 3)
```

```
## Check if the data list is available for rbind or cbind...
##
## Data list is OK
```



Plot rules are just the same as described in 1.3.3, please check if needed.

2.7.2 Get Bias-corrected Data.

Usually time series is needed in the model. As introduced in section 2.5, in `biasCorrect()`, you can get bias corrected output.

2.7.3 Resample Data

As described in section 1.3.4, `resample` also works for hyfo grid data.

3. Anarbe Case

The functions with anarbe case end with `_anarbe`, all of them are used to collect different available published data in anarbe catchment in Spain. The data comes from two website: here and here, there are precipitation or discharge data on those website, and can be downloaded directly.

Since the available files on those website are arranged by a year or five years, for long term data collection, a tools is necessary for collecting data from different files.

Note: For excel files, if you have access to the dam regulation excel file of the dam anarbe, you can use `collectData_excel_anarbe` in the package, but this function is commented in the original code, cannot be used directly. Go to original file in the library or go to github here, copy the original code.

There are two csv files and txt files included in the package, which can be used as examples.

```
file <- system.file("extdata", "1999.csv", package = "hyfo")
folder <- strsplit(file, '1999')[[1]][1]

a <- collectData_csv_anarbe(folder, output = TRUE)
str(a)
b <- collectData_txt_anarbe(folder, output = TRUE)
str(b)
```