yiR Documentation

Release latest

April 20, 2016

Documentation

My collection of handy R functions.

CHAPTER 1

Data Manipulation

```
rbind.ids <- function(dataX, dataY, cols = NULL, add.id = TRUE){
    require(data.table)
    df.name <- c(deparse(substitute(dataX)), deparse(substitute(dataY)))
    dataX <- as.data.table(dataX)
    dataY <- as.data.table(dataY)
    if (is.null(cols))
        cols <- union(names(dataX), names(dataY))
    tmp <- rbind(dataX[, cols, with=F], dataY[, cols, with = F])
    if (add.id)
        tmp[, join.id := rep( df.name, c(nrow(dataX), nrow(dataY)))]
    tmp
}
```

Print the object X in org-mode format.

```
ToOrg <- function(X){
    require(ascii)
    print(ascii(X), type = "org")
}
```

CHAPTER 2

Utilise

2.1 detachAllPackages

Detach all the libraries except the official start-up libraries. Need it in development packages.

```
##' Detach all packages
##'
##'
##' @title utilities
##' @return nothing
##' @export
##' @author Yi Tang
detachAllPackages <- function() {

    basic.packages <- c("package:stats","package:graphics","package:grDevices","package:utils","packa

    package.list <- search()[ifelse(unlist(gregexpr("package:",search()))==1,TRUE,FALSE)]

    package.list <- setdiff(package.list,basic.packages)

    if (length(package.list)>0)  for (package in package.list) detach(package, character.only=TRUE)

}
```

2.2 Object Size

Show all the objects and their size, watch out your R memory usage!

```
lsos()
```

```
# improved list of objects
.ls.objects <- function (pos = 1, pattern, order.by,
                        decreasing=FALSE, head=FALSE, n=5) {
    napply <- function(names, fn) sapply(names, function(x)
                                         fn(get(x, pos = pos)))
    names <- ls(pos = pos, pattern = pattern)
    obj.class <- napply(names, function(x) as.character(class(x))[1])
    obj.mode <- napply(names, mode)
    obj.type <- ifelse(is.na(obj.class), obj.mode, obj.class)
    obj.size <- napply(names, object.size)
    obj.dim <- t(napply(names, function(x)
                        as.numeric(dim(x))[1:2]))
    vec <- is.na(obj.dim)[, 1] & (obj.type != "function")
    obj.dim[vec, 1] <- napply(names, length)[vec]
    out <- data.frame(obj.type, obj.size, obj.dim)
    names(out) <- c("Type", "Size", "Rows", "Columns")
    if (!missing(order.by))
        out <- out[order(out[[order.by]], decreasing=decreasing), ]
    if (head)
        out <- head(out, n)
    out
}
# shorthand
lsos <- function(..., n=10) {
    .ls.objects(..., order.by="Size", decreasing=TRUE, head=TRUE, n=n)
}
```

2.3 IO

```
#' save ggplot to a file which is right to be imported in World document.
my.png <- function(p, file = deparse(substitute(p))){
    png(paste(file, ".png", sep=""),  width = 12, height = 9, units = 'in', res = 300)
    print(p)
    dev.off()
}
```

estimate file size

```
##'  Estimate the space of output file
##'
##' more accurate than the object.size function.
##' @title
##' @param a.df a data.table or data.frame
##' @param n number of rows used in estimation, if n is less than 1, propoertion of rows used
##' @param ... args passed to write.csv
##' @return a estimated disk space to save a.df, in Mb unit.
##' @export
##' @author Yi Tang
SizeEst <- function(a.df, n,  ...) {
    file <- tempfile()
    if (n <= 1)
        n <- ceiling(n * nrow(a.df))
    df.subset <- a.df[seq_len(n), ]
    write.csv(df.subset, file = file, ...)
    disk.space <- file.info(file)$size
    file.remove(file)
    disk.space.est <- disk.space  / nrow(df.subset) * nrow(a.df)
    disk.space.est / (2 ^ 20) ## return as MB
}
```

2.4 Assorted

```
##'  Tests
##'
##' Test whether or not two data.table objects are identical.
##' @title
##' @param dt1
##' @param dt2
##' @param verbose
##' @return
##' @export
##' @author Yi Tang
identical.data.table <- function(dt1, dt2, verbose = TRUE) {
    if (nrow(dt1) != nrow(dt2))
        stop("different number of rows")
    if (ncol(dt1) != ncol(dt2))
        stop("different number of columns")
    if (names(dt1) != names(dt2))
        stop("different column names")
    n <- ncol(dt1)
    sapply(seq_len(n), function(i) {
        cat("\nTest columns", i)
        if(!identical(dt1[[i]], dt2[[i]]))
            stop("col ", i, " is different")
    })
    TRUE
}
```

CHAPTER 3

Visualisation

3.1 Map Layers

```
##' Map layer in ggplot2
##'
##' Create a polygon of world.
##' @title GIS
##' @return a ggplot layer
##' @export
##' @author Yi Tang
ggMapLayer <- function(){
    world <- ggplot2::map_data("world")
    gg.map <- geom_polygon(data = world, aes(x = long, y= lat, group = group))
    return(gg.map)
}


##' Add world map lay on top of a ggplot
##'
##'
##' @title Maps in gglot
##' @param p a ggplot object
##' @param alpha a number from 0 to 1. Controls the transprancy of the map
##' @param border.col colour of the boarder
##' @param xylim.no.change Logical. Whether to change xy limits or not
##' @return a ggplot layer
##' @export
##' @author Yi Tang
add_map_layer <- function (p, alpha = 0.2, border.col = "white", xylim.no.change = TRUE){
    require(maps)
    require(ggplot2)
    nworld_data <- map_data("world")
    dt <- as.data.frame(p$data)
    xy <- as.character(p$mapping)
    map <- geom_polygon(data = world_data, aes(x = long, y = lat,
                                              group = group), col = border.col, alpha = alpha)
    if (xylim.no.change)
        p + map + coord_cartesian(xlim = range(dt[, xy[1]]),
                                  ylim = range(dt[, xy[2]]))
    else p + map
}
```

```
##' save ggplot to a file which is right to be imported in World document.
##'
##' Save a ggplot object as a png file
##' @title SavePlots
##' @param p
##' @param file
##' @export
##' @author Yi Tang
my.png <- function(p, file = deparse(substitute(p))){
    png(file, width = 12, height = 9, units = 'in', res = 300)
    print(p)
    dev.off()
}
```

3.2 Title short hand

```
##' Hazard Map in ggplot2
##'
##' Short hand for plotting hazard map with rainbow color.
##' @title Hazard Map
##' @param df the dataframe
##' @param x dimension 1
##' @param y dimension 2
##' @param z values to visualise
##' @param rainbow Logical. Using rainbow or jet color scheme
##' @param minmax Logical, show minimal and max in legend
##' @param ...
##' @return a ggplot object
##' @export
##' @author Yi Tang
gg.hazard <- function(df, x = "lon", y = "lat", z = "V1", rainbow = FALSE, minmax = TRUE, ...) {
    if (rainbow == TRUE)
        this.colors <- rev(rainbow(50, start = 0, end = 4/6))
    else
        this.colors <- jet.colors(7)

    p <- ggplot(df, aes_string(x = x, y = y)) + geom_tile(aes_string(fill = z))
    if (minmax == TRUE){
        var <- df[[z]]
        ticks <- seq(min(var), max(var), len = 5)
        p <- p + scale_fill_gradientn(colours = this.colors , breaks = ticks, labels = round(ticks, (
    }
    else
        p <- p + scale_fill_gradientn(colours = this.colors)

    return(p)
}
```

3.3 others

```
##' compare multi-variables in two dataset in terms of density.
##'
##' Produce density plots for common variables in two dataset
##' @title Visualisation
##' @param dataX
##' @param dataY
##' @param cols
##' @export
##' @author Yi Tang
Compare_XY_Density <- function(dataX, dataY, cols = names(dataX)){
    require(data.table)
    require(ggplot2)
    require(reshape2)
    df.name <- c(deparse(substitute(dataX)), deparse(substitute(dataY)))
    x <- as.data.table(dataX)[, cols, with=F] ## subset
    y <- as.data.table(dataY)[, cols, with=F]
    xy <- rbind(x, y)
    xy[, dataset := rep(df.name, c(NROW(dataX), NROW(dataY)))]
    ggdf <- melt(xy, id="dataset")
    p <- ggplot(ggdf, aes(x = value, col = dataset)) + geom_density() + facet_wrap(~ variable, scale
    return(p)
}



                                        # ' save a list of ggplot
gg.save.list <- function(p.list, file.name = c("var.name", "title")){
    file.name <- match.arg(file.name)
    nm <- names(p.list)
    if (any(nm == ""))
        stop("list must have name")

    for (i in seq_along(p.list)){
        elem <- p.list[[i]]
        if (any(grepl("ggplot", class(elem)))){
            my.png(elem, nm[i])

        } else if (class(elem) == "list"){
            for (j in 1:length(elem)){
                if (any(grepl("ggplot", class(elem))))
                    my.png(elem[[j]], paste(nm[i], names(elem)[j]))
            }
        } else {
            message("\n", i, "-th element is skipped")
        }
    }
}


#' ggpot defaul color scheme
gg_color_hue <- function(n) {
    hues = seq(15, 375, length=n+1)
    hcl(h=hues, l=65, c=100)[1:n]
}

##' Step Function, replaced by geom_step()
##'
##' add stepping to points (x1, y1), (x2, y2)...
##' @title this is title
##' @param x
##' @param y
##' @return a ggplot object
##' @export
##' @author Yi Tang
plotStepFunction <- function(x, y) {
    formatLineSegDF <- function(x, y) {
        x.start <- x[-length(x)]
        x.end <- x[-1]
        y.start <- y.end <- y[-length(y)]
        data.table(x.start, x.end, y.start, y.end)
    }
    gg.df <- data.table(x, y)
    line.seg.df <- formatLineSegDF(x, y)
    ggplot(gg.df, aes(x = x, y = y)) + geom_point() +
        geom_segment(data = line.seg.df, aes(x = x.start, xend = x.end, y = y.start, yend = y.end))
}




#' compare multi-variables in two dataset in terms of density.
density_facet_ggplot <- function(dataX, dataY, cols = names(dataX)){
    require(data.table)
    require(ggplot2)
    require(reshape2)
    df.name <- c(deparse(substitute(dataX)), deparse(substitute(dataY)))
    x <- as.data.table(dataX)[, cols, with=F] ## subset
    y <- as.data.table(dataY)[, cols, with=F]
    xy <- rbind(x, y)
    xy[, dataset := rep(df.name, c(NROW(dataX), NROW(dataY)))]
    ggdf <- melt(xy, id="dataset")
    p <- ggplot(ggdf, aes(x = value, col = dataset)) + geom_density() + facet_wrap(~ variable, scale
    return(p)
}


#' ggMap
#' an world map as ggplot layer.
#' @example
#' data(hur)
#' gg.map <- ggMapLayer()
#' ggplot(hur, aes(x=long, y=lat)) + gg.map + geom_point(alpha=0.5)
ggMapLayer <- function(){
    world <- ggplot2::map_data("world")
    gg.map <- geom_polygon(data = world, aes(x = long, y= lat, group = group))
    return(gg.map)
}

#' add a world map layer on top on current ggplot,
add_map_layer <- function (p, alpha = 0.2, border.col = "white", xylim.no.change = TRUE){
    require(maps)
    require(ggplot2)
    world_data <- map_data("world")
    dt <- as.data.frame(p$data)
    xy <- as.character(p$mapping)
    map <- geom_polygon(data = world_data, aes(x = long, y = lat,
                                              group = group), col = border.col, alpha = alpha)
    if (xylim.no.change)
        p + map + coord_cartesian(xlim = range(dt[, xy[1]]),
                                         ylim = range(dt[, xy[2]]))
    else p + map
}

#' shorthand for plotting hazard map with rainbow color
gg.hazard <- function(df, x = "lon", y = "lat", z = "V1") {
    p <- ggplot(df, aes_string(x = x, y = y)) + geom_tile(aes_string(fill = z)) + scale_fill_gradient
    return(p)
}


####
##autoplot block
####

#' @reference http://librestats.com/2012/06/11/autoplot-graphical-methods-with-ggplot2/
#'


#' Check if point(x, y) cross x = a or y = b line or not.
#' @example
#' df <- structure(list(long = c(-34.2078157528796, -36.2074309699417,
#' -38.2792884737378, -40.2170974851064, -42.1780335768454, -44.1455967338515,
#' -46.4166770126002, -48.2943755413367, -50.0298281808574, -51.3114969986729,
#' -52.5441609178788, -53.7237533552569, -54.8823832918566, -55.8138670417713,
#' -56.722942918336, -57.6053794360869, -58.6228611648525, -59.5945636149565,
#' -60.591387403632, -61.5473518311293, -62.1877994852336, -62.6515082234799,
#' -63.0336967876415, -63.1144294155782, -63.3581282050358, -63.7885840015858,
#' -64.1370270967451, -64.5242218821295, -64.7156587993305, -64.5240440927702,
#' -63.6592485824048, -61.9361108628757, -59.9644036105799, -57.6850668652962,
#' -55.6269026718672), lat = c(25.220877237296, 25.0449951402138,
#' 24.2921987727861, 23.7331891023609, 22.882202932219, 21.9955804595675,
#' 21.3357658016897, 20.7665536331803, 20.4345722631771, 20.1713918488524,
#' 19.9607061210464, 19.803542676299, 19.6514829290709, 19.2946576063146,
#' 19.0185917630829, 18.6544443206195, 18.350027781835, 18.4634817877027,
#' 18.8968767697435, 19.5874567186624, 20.3871126308597, 21.1001662298256,
#' 21.6360080794085, 22.214195275253, 22.6741813775321, 23.2123449093717,
#' 24.0670696428687, 25.3393933451918, 27.2736476853658, 29.0676356902004,
#' 30.8813383987791, 32.49317150619, 33.1745910761416, 34.2677474173933,
#' 35.0450609158249)), .Names = c("long", "lat"), row.names = c(NA,
#' -35L))
#' x.grid <- seq(-70, -30, by = 10)
#' res <- CrossX(df$long, df$lat, x.grid)
#' plot(df$long, df$lat)
#' abline( v = x.grid, col = 2)
#' points(res$x, res$y, col = ifelse(res$dir == "negative", 2, 3), pch = 19)
CrossX <- function(x, y, x.grid){
    cat('\n', 'positive means', '\n from left to right or \n bottom to top\n')

    d <- c(0, diff( findInterval(x, x.grid)))
    if (sum(d != 0) == 0)
        return(NULL)
                                        # negative direction
    res1 <- res2 <- NULL
    ind <- which(d < 0)
    if(length(ind) != 0){
        xx <- x.grid[ findInterval(x[ind], x.grid) + 1]
        if (length(ind) == 1){
            ind <- c(ind - 1, ind)
        } else {
            ind[1] <- ind[1] -1  # otherwise, the first interpolated will be NA.
        }
        res1 <- approx(x[ind], y[ind], xx)
        res1$dir <- "negative"
    }

                                        # positive direction
    ind <- which(d > 0)
    if(length(ind) != 0){
        xx <- x.grid[ findInterval(x[ind], x.grid) ]
        if (length(ind) == 1){
            ind <- c(ind - 1, ind)
        } else {
            ind[1] <- ind[1] -1  # otherwise, the first interpolated will be NA.
        }
        res2 <- approx(x[ind], y[ind], xx)
        res2$dir <- "positive"
    }

    res <- rbind( as.data.table(res1), as.data.table(res2))
    res
}


                                        # ' save a list of ggplot
gg.save.list <- function(p.list, file.name = c("var.name", "title")){
    file.name <- match.arg(file.name)
    nm <- names(p.list)
    if (any(nm == ""))
        stop("list must have name")

    for (i in seq_along(p.list)){
        elem <- p.list[[i]]
        if (any(grepl("ggplot", class(elem)))){
            my.png(elem, nm[i])

        } else if (class(elem) == "list"){
            for (j in 1:length(elem)){
                if (any(grepl("ggplot", class(elem))))
                    my.png(elem[[j]], paste(nm[i], names(elem)[j]))
            }
        } else {
            message("\n", i, "-th element is skipped")
        }
    }
}


#' ggpot defaul color scheme
gg_color_hue <- function(n) {
    hues = seq(15, 375, length=n+1)
    hcl(h=hues, l=65, c=100)[1:n]
}


#' plot hazard map
```

```
gg.hazard <- function(df, x = "lon", y = "lat", z = "V1", ...) {
    var <- df[[z]]
    ticks <- seq(min(var), max(var), len = 5)
    p <- ggplot(df, aes_string(x = x, y = y)) + geom_tile(aes_string(fill = z)) + scale_fill_gradient
    return(p)
}




#### ggplot, piechart
## help function
#' check also: https://github.com/jrnold/ggthemes
#' (especially for the color schemes)

#' define style for the charts ####
#' usage: g <- g +getstyle (text_size = 20)
#' ref: https://gist.github.com/nassimhaddad/4994317
getstyle <- function(text_size = 20){
    theme_bw() +
        theme(axis.title.x = element_text(colour="black", size=text_size)) +
        theme(axis.text.x = element_text(size = text_size)) +
        theme(axis.title.y = element_text(colour="black", size=text_size)) +
        theme(axis.text.y = element_text(size = text_size)) +
        theme(legend.position="none") +
        theme(plot.title = element_text(face="bold", size = text_size+2, vjust = 2))
}

ggpie <- function(data, category = character(), value = numeric()){
    require(ggplot2)
    require(ggthemes)
    data$category <- data[, category]
    data$value <- data[, value]
    data$category <- factor(data$category,
                            levels = data$category[order(data$value, decreasing=TRUE)])

    p <- ggplot(data, aes(x = factor(1), fill = factor(category), y = (value)/sum(value),
                          order = (value)/sum(value))) +
        geom_bar(stat = "identity", width = 1) +
        labs(title = "", x = "", y= "") +
        getstyle(10) + scale_fill_tableau("colorblind10")+
        coord_polar(theta="y", direction = -1) +
        theme(legend.position="right") +
        theme(axis.ticks=element_blank(), axis.text.y = element_blank(), axis.text.x = element_blank
              legend.text=element_text(size=14), legend.title=element_text(size=14) )+
        guides(fill = guide_legend(title = category))
    return(p)
}
```

```
#' Automatically setup par mfrow
#'
#' Determine how many rows and plots to make for a certain number of
#' plots & make the appropriate call to \code{par(mfrow)}
#'
#' eg 25 plots will cause par(mfrow=c(5,5)) to be called.
#' The function fills in any unsued plotting spots, eg:
#' auto.mfrow(7, TRUE) will set up a device with 3x3 spaces, then when
#' auto.mfrow(7, FALSE) is called
#' after the plots have been made, 2 blank plots will then be 'printed'.
#' NB, you must call auto.mfrow TWICE, once before plotting, and once after
#' plotting, UNLESS
#' you know for sure that the nplots specified will fill all of the spaces.
#'
#' @param nplots an integer in [1,49]
#' @param setup if \code{TRUE}, then the graphical parameters (par) is set-up if
#'    \code{FALSE}, and nplots < the number of spaces for plots in the device, then
#'   blank plots are added to fill in the unused spaces.
#' @author Mark Cowley, 3 June 2006
#' @export
auto.mfrow <- function(nplots, setup=TRUE) {

        if(setup) {
                if(nplots <= 3) par(mfrow=c(1, nplots))
                else if(nplots <= 4)  par(mfrow=c(2,2))
                else if(nplots <= 6)  par(mfrow=c(2,3))
                else if(nplots <= 9)  par(mfrow=c(3,3))
                else if(nplots <= 12) par(mfrow=c(3,4))
                else if(nplots <= 16) par(mfrow=c(4,4))
                else if(nplots <= 20) par(mfrow=c(4,5))
                else if(nplots <= 25) par(mfrow=c(5,5))
                else if(nplots <= 30) par(mfrow=c(5,6))
                else if(nplots <= 36) par(mfrow=c(6,6))
                else if(nplots <= 42) par(mfrow=c(6,7))
                else if(nplots <= 49) par(mfrow=c(7,7))
                else if(nplots <= 56) par(mfrow=c(7,8))
                else if(nplots <= 64) par(mfrow=c(8,8))
                else {
                        stop("Too many plots")
                }
        }
        else {
                nblankplots <- par("mfrow")[1] * par("mfrow")[2] - nplots
                if(nblankplots > 0)
                        for(i in 1:nblankplots)
                                plot_blank()
        }
}
```

CHAPTER 4

GIS

4.1 Cross_X

```
#' Check if point(x, y) cross x = a or y = b line or not.
#' @example
#' df <- structure(list(long = c(-34.2078157528796, -36.2074309699417,
#' -38.2792884737378, -40.2170974851064, -42.1780335768454, -44.1455967338515,
#' -46.4166770126002, -48.2943755413367, -50.0298281808574, -51.3114969986729,
#' -52.5441609178788, -53.7237533552569, -54.8823832918566, -55.8138670417713,
#' -56.722942918336, -57.6053794360869, -58.6228611648525, -59.5945636149565,
#' -60.591387403632, -61.5473518311293, -62.1877994852336, -62.6515082234799,
#' -63.0336967876415, -63.1144294155782, -63.3581282050358, -63.7885840015858,
#' -64.1370270967451, -64.5242218821295, -64.7156587993305, -64.5240440927702,
#' -63.6592485824048, -61.9361108628757, -59.9644036105799, -57.6850668652962,
#' -55.6269026718672), lat = c(25.220877237296, 25.0449951402138,
#' 24.2921987727861, 23.7331891023609, 22.882202932219, 21.9955804595675,
#' 21.3357658016897, 20.7665536331803, 20.4345722631771, 20.1713918488524,
#' 19.9607061210464, 19.803542676299, 19.6514829290709, 19.2946576063146,
#' 19.0185917630829, 18.6544443206195, 18.350027781835, 18.4634817877027,
#' 18.8968767697435, 19.5874567186624, 20.3871126308597, 21.1001662298256,
#' 21.6360080794085, 22.214195275253, 22.6741813775321, 23.2123449093717,
#' 24.0670696428687, 25.3393933451918, 27.2736476853658, 29.0676356902004,
#' 30.8813383987791, 32.49317150619, 33.1745910761416, 34.2677474173933,
#' 35.0450609158249)), .Names = c("long", "lat"), row.names = c(NA,
#' -35L))
#' x.grid <- seq(-70, -30, by = 10)
#' res <- CrossX(df$long, df$lat, x.grid)
#' plot(df$long, df$lat)
#' abline( v = x.grid, col = 2)
#' points(res$x, res$y, col = ifelse(res$dir == "negative", 2, 3), pch = 19)
CrossX <- function(x, y, x.grid){
    cat('\n', 'positive means', '\n from left to right or \n bottom to top\n')

    d <- c(0, diff( findInterval(x, x.grid)))
    if (sum(d != 0) == 0)
        return(NULL)
                                            # negative direction
    res1 <- res2 <- NULL
    ind <- which(d < 0)
    if(length(ind) != 0){
        xx <- x.grid[ findInterval(x[ind], x.grid) + 1]
        if (length(ind) == 1){
            ind <- c(ind - 1, ind)
        } else {
            ind[1] <- ind[1] -1  # otherwise, the first interpolated will be NA.
        }
        res1 <- approx(x[ind], y[ind], xx)
        res1$dir <- "negative"
    }

                                                    # positive direction
    ind <- which(d > 0)
    if(length(ind) != 0){
        xx <- x.grid[ findInterval(x[ind], x.grid) ]
        if (length(ind) == 1){
            ind <- c(ind - 1, ind)
        } else {
            ind[1] <- ind[1] -1  # otherwise, the first interpolated will be NA.
        }
        res2 <- approx(x[ind], y[ind], xx)
        res2$dir <- "positive"
    }

    res <- rbind( as.data.table(res1), as.data.table(res2))
    res
}
```

CHAPTER 5

EVT

```
##' Transfer Laplace distirbution to original scale
##'
##' Given a upper part of x, GPD is used to extrpolate . for y <= qu, empirical transformation. for y
##' @title EVT
##' @param y numeric vector. Laplace distirbution
##' @param x numeric vector, original distirbution
##' @param qu numeric from 0 to 1.
##' @param coef numeric vector of 3 element. GPD parameters
##' @export
##' @author Yi Tang
y2x <- function(y, x, qu, coef){
    u <- rank(y, ties = "random")/(1+length(y))
    xx <- rep(NA, len = length(u))
    threshold <- quantile(x, qu)
    ind <- u <= qu
    xx[ind] <- quantile(x, u[ind])
    if (any(!ind )){
        sig <- coef[2]
        xi <- coef[3]
        xx[!ind] <- texmex::qgpd(1- (1 - u[!ind]) / (1 - qu), sigma = sig, xi = xi, u = 0) + thresho
    }
    return(xx)
}

##' Convert GPD distribution to GEV distribution
##'
##'
##' @title EVT
##' @param mu location parameter of GPD
##' @param sigma scale parameter of GPD
##' @param xi shape parameter of GPD
##' @param lambda lambda is the average clusters per year
##' @return GEV parameter
##' @export
##' @author Yi Tang
GPD2GEV <- function(mu, sigma, xi, lambda){
    mu = mu + sigma*(lambda^xi - 1) / xi
    sigma = sigma * lambda^xi
    xi = xi
    return(c(mu, sigma, xi))
}

##' Convert return level to return period
```

```
##'
##' .. content for \details{} ..
##' @title EVT
##' @param rl return level
##' @param gev.mle gev parameter
##' @return returnp eriod
##' @export
##' @author Yi Tang
RL.to.RP <- function(rl, gev.mle){ # function that turns precip level to RP (rearranged)
    rp <-  1 / ( 1 - texmex::pgev(rl, mu = gev.mle[1], sigma = gev.mle[2], xi = gev.mle[3]))
    return(rp)
}

#' automatically choose GPD threshold by coverage
optimGPDThreshold <- function(x){
    res <- gpdRangeFit(x, umin = quantile(x, 0.8), umax = quantile(x, 0.995), nint = 100)
    ## plot(res)
    phi <- data.table(th = res$th,
                      mle = res$par[, 1],
                      lb = res$lo[, 1],
                      ub = res$hi[, 1])
    xi <-  data.table(th = res$th,
                      mle = res$par[, 2],
                      lb = res$lo[, 2],
                      ub = res$hi[, 2])
    par <- data.table(rbind(phi, xi), par = rep(c("phi", "xi"), c(nrow(phi), nrow(xi))))
    xi[, n.id := 1:nrow(xi)]
    xi[, n.cover := {
        sum(lb <= xi$mle & ub >= xi$mle)
    }, by = n.id]
    opt.th <- xi[which.max(n.cover), th]
    return(opt.th)
}
```

Distribution

6.1 Laplace

```
##' Laplace Distribution
##'
##' Basic functions that relate to Laplace distribution
##' @name LaplaceDistribution
##' @param x a random variable
##' @param mu para 1
##' @param b para 2
##' @export
##' @author Yi Tang
toLaplace <- function(x, mu = 0, b = 1){
    ## u <- rank(x, ties = "random") / (1+length(x))
    u <- rank(x) / (1+length(x))
    y <-     mu - b * sign(u - 0.5) * log(1 - 2 * abs(u - 0.5))
    return(y)
}
##' @rdname LaplaceDistribution
qLaplace <- function(p, mu = 0, b = 1){
     ## mu is locaton param, b is scale para
    mu - b * sign(p - 0.5) * log(1 - 2 * abs(p - 0.5))
}
##' @rdname LaplaceDistribution
pLaplace <- function(x, mu = 0, b = 1){
    ## mu is location para, b is scale para
    1/2 + 1/2 * sign(x - mu) * (1  - exp(- abs(x - mu) / b))
}
```

CHAPTER 7

Machine Learning

```r
PrepFit <- function(df, y.name) {
    df <- as.data.frame(df)
    names(df)[names(df) == y.name] <- "y.name"
    set.seed(1)
    n <- nrow(df)
    train <- sample(n, n %/% 2)
    X <<- model.matrix(y.name ~ ., data = df[train, ])[,-1]
    Y <<- df[train, ][["y.name"]]
    X.test <<- model.matrix(as.formula(y.name ~ .), data = df[-train, ])[,-1]
    Y.test <<- df[-train, ][["y.name"]]
    train.dat <<- data.table(X, y.name = Y)
    setnames(train.dat, "y.name", y.name)
    test.dat <<- data.table(X.test, y.name = Y.test)
    setnames(test.dat, "y.name", y.name)
}
```