title: “LAB 13”

author: “JESSICA PAOLA AGUILAR SERVIN”

date: “2023-02-23”

output: html_document

LABORATORIO - Gráficos en R con ggplot2

Instalar paqueterias necesarias cargar libreria ggplot2

```
library(ggplot2)
```

Leer Bases de datos

```
green_data <- read.csv("~/GitHub/JPAS_LAB13/INPUT/REGESIONES FINALES.csv")
```

Echando un ojo a los datos

```
names(green_data)
```

```
## [1] "STATE"     "GCI_rank"  "ICE_rank"  "GCI_index" "ICE_index" "PIBE"
## [7] "LPIBE"
```

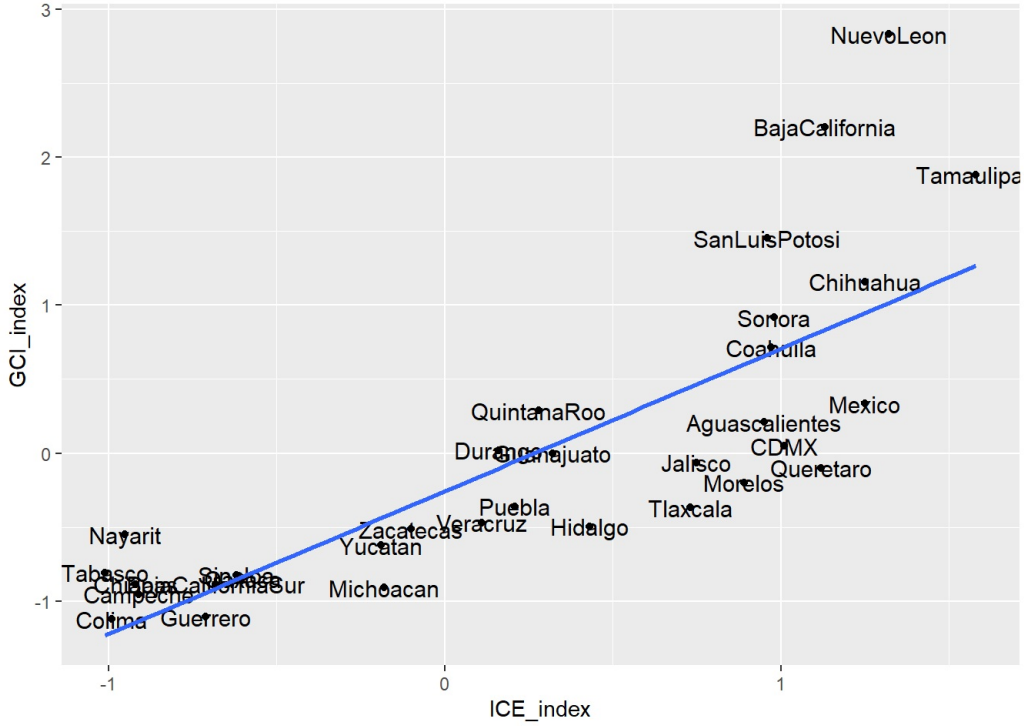
Generar primer grafico

```
p1 <- ggplot(data = green_data,
             mapping = aes(x = ICE_index,
                           y = GCI_index,)) +
geom_point()+
  geom_text(label= green_data$STATE,
            color= "black",
            size= 4)+
  geom_point() +
  geom_smooth(method = lm,
              se=FALSE,
              fullrange =T)
```

Visualizar p1

```
p1
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



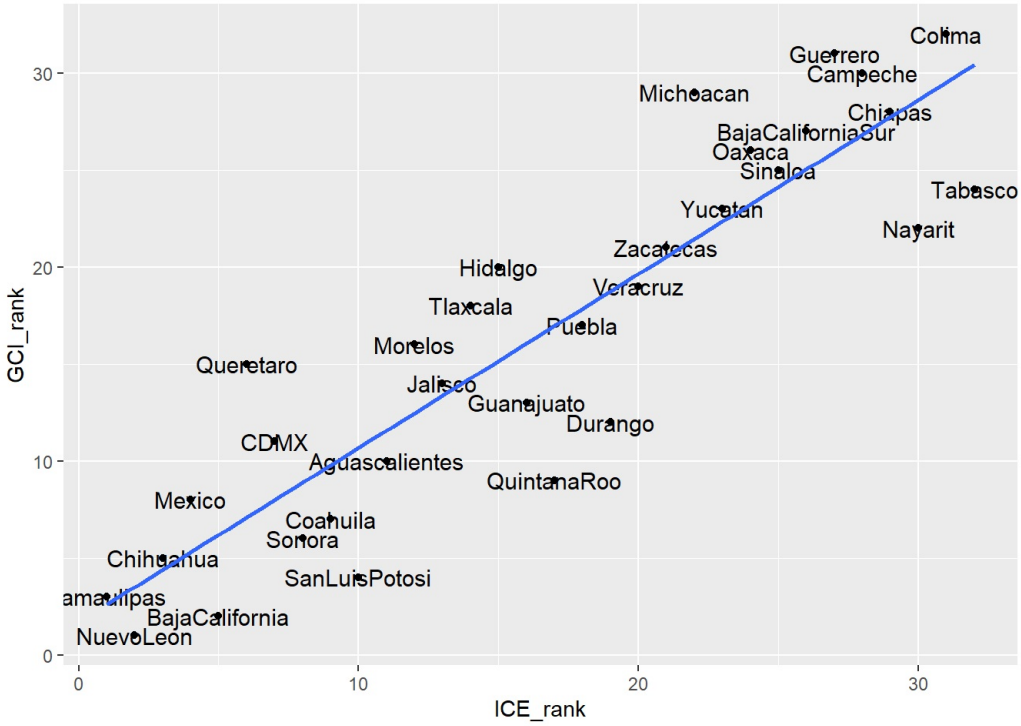
Generar segundo grafico reciclado del codigo anterior

```
p2 <- ggplot(data = green_data,
             mapping = aes(x = ICE_rank,
                           y = GCI_rank,)) +
  geom_point()+
  geom_text(label= green_data$STATE,
            color= "black",
            size= 4)+
  geom_point() +
  geom_smooth(method = lm,
              se=FALSE,
              fullrange =T)
```

Visualizar p2

```
p2
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



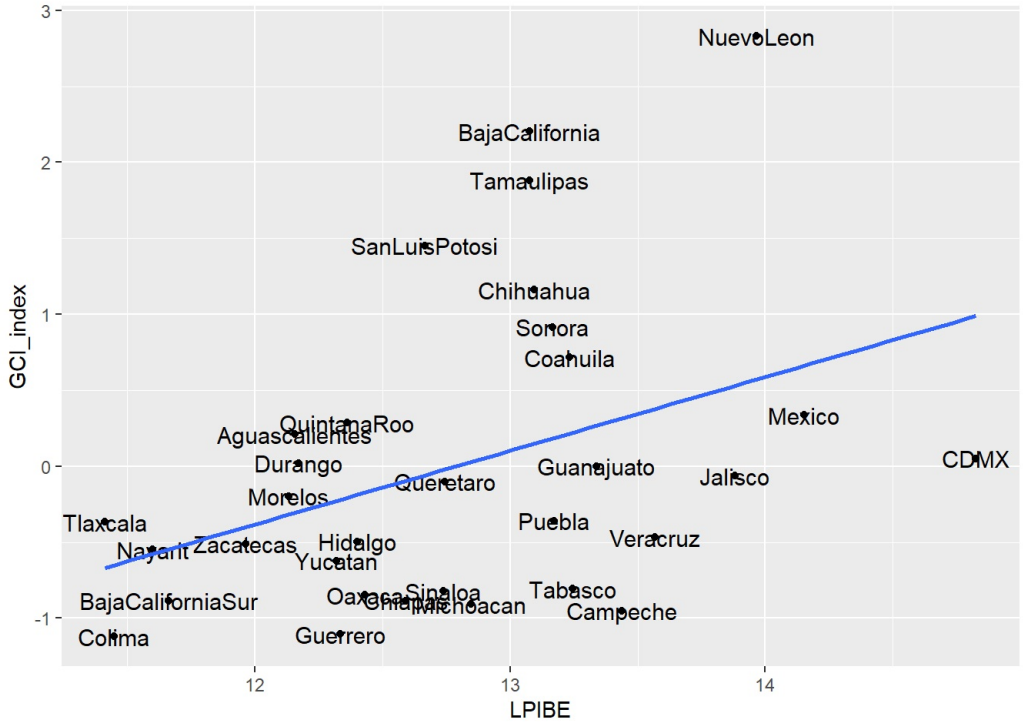
Generar tercer grafico

```
p3 <- ggplot(data = green_data,
             mapping = aes(x = LPIBE,
                           y = GCI_index,)) +
  geom_point()+
  geom_text(label= green_data$STATE,
            color= "black",
            size= 4)+
  geom_point() +
  geom_smooth(method = lm,
              se=FALSE,
              fullrange =T)
```

VIsualizar p3

```
p3
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



Correlación entre las primeras variables

```
install.packages
```

```
## function (pkgs, lib, repos = getOption("repos"), contriburl = contrib.url(repos,
##     type), method, available = NULL, destdir = NULL, dependencies = NA,
##     type = getOption("pkgType"), configure.args = getOption("configure.args"),
##     configure.vars = getOption("configure.vars"), clean = FALSE,
##     Ncpus = getOption("Ncpus", 1L), verbose = getOption("verbose"),
##     libs_only = FALSE, INSTALL_opts, quiet = FALSE, keep_outputs = FALSE,
##     ...)
## {
##     if (!is.character(type))
##         stop("invalid 'type'; must be a character string")
##     type2 <- .Platform$pkgType
##     if (type == "binary") {
##         if (type2 == "source")
##             stop("type 'binary' is not supported on this platform")
##         else type <- type2
##         if (type == "both" && (!missing(contriburl) || !is.null(available)))
##             stop("specifying 'contriburl' or 'available' requires a single type, not type = \"both\"")
##     }
##     if (is.logical(clean) && clean)
##         clean <- "--clean"
##     if (is.logical(dependencies) && is.na(dependencies))
##         dependencies <- if (!missing(lib) && length(lib) > 1L)
##             FALSE
##         else c("Depends", "Imports", "LinkingTo")
##     get_package_name <- function(pkg) {
##         gsub("_[.](zip|tar[.]gz|tar[.]bzip2|tar[.]xz)", "", gsub(.standard_regexps()$valid_package_version,
##             "", basename(pkg)))
##     }
##     getConfigureArgs <- function(pkg) {
##         if (.Platform$OS.type == "windows")
##             return(character())
##         if (length(pkgs) == 1L && length(configure.args) && length(names(configure.args)) ==
##             0L)
##             return(paste0("--configure-args=", shQuote(paste(configure.args,
##                 collapse = " "))))
##         pkg <- get_package_name(pkg)
##         if (length(configure.args) && length(names(configure.args)) &&
##             pkg %in% names(configure.args))
##             config <- paste0("--configure-args=", shQuote(paste(configure.args[[pkg]],
##                 collapse = " ")))
##         else config <- character()
##         config
##     }
##     getConfigureVars <- function(pkg) {
##         if (.Platform$OS.type == "windows")
##             return(character())
##         if (length(pkgs) == 1L && length(configure.vars) && length(names(configure.vars)) ==
```

```
##             0L)
##             return(paste0("--configure-vars=", shQuote(paste(configure.vars,
##                 collapse = " "))))
##         pkg <- get_package_name(pkg)
##         if (length(configure.vars) && length(names(configure.vars)) &&
##             pkg %in% names(configure.vars))
##             config <- paste0("--configure-vars=", shQuote(paste(configure.vars[[pkg]],
##                 collapse = " ")))
##         else config <- character()
##         config
##     }
##     get_install_opts <- function(pkg) {
##         if (!length(INSTALL_opts))
##             character()
##         else paste(INSTALL_opts[[get_package_name(pkg)]], collapse = " ")
##     }
##     if (missing(pkgs)) {
##         if (!interactive())
##             stop("no packages were specified")
##         if (.Platform$OS.type == "windows" || .Platform$GUI ==
##             "AQUA" || (capabilities("tcltk") && capabilities("X11") &&
##             suppressWarnings(tcltk::.TkUp))) {
##         }
##         else stop("no packages were specified")
##         if (is.null(available)) {
##             av <- available.packages(contriburl = contriburl,
##                 method = method, ...)
##             if (missing(repos))
##                 repos <- getOption("repos")
##             if (type != "both")
##                 available <- av
##         }
##         else av <- available
##         if (NROW(av)) {
##             pkgs <- select.list(sort(unique(rownames(av))), multiple = TRUE,
##                 title = "Packages", graphics = TRUE)
##         }
##     }
##     if (.Platform$OS.type == "windows" && length(pkgs)) {
##         pkgnames <- get_package_name(pkgs)
##         inuse <- search()
##         inuse <- sub("^package:", "", inuse[grep("^package:",
##             inuse)])
##         inuse <- pkgnames %in% inuse
##         if (any(inuse)) {
##             warning(sprintf(ngettext(sum(inuse), "package %s is in use and will not be installed",
##                 "packages %s are in use and will not be installed"),
##                 paste(sQuote(pkgnames[inuse]), collapse = ", ")),
##                 call. = FALSE, domain = NA, immediate. = TRUE)
##             pkgs <- pkgs[!inuse]
##         }
##     }
##     if (!length(pkgs))
##         return(invisible())
##     if (missing(lib) || is.null(lib)) {
##         lib <- .libPaths()[1L]
##         if (!quiet && length(.libPaths()) > 1L)
##             message(sprintf(ngettext(length(pkgs), "Installing package into %s\n(as %s is unspecified)",
##                 "Installing packages into %s\n(as %s is unspecified)"),
##                 sQuote(lib), sQuote("lib")), domain = NA)
##     }
##     ok <- dir.exists(lib) & (file.access(lib, 2) == 0L)
##     if (length(lib) > 1 && any(!ok))
##         stop(sprintf(ngettext(sum(!ok), "'lib' element %s is not a writable directory",
##             "'lib' elements %s are not writable directories"),
##             paste(sQuote(lib[!ok]), collapse = ", ")), domain = NA)
##     if (length(lib) == 1L && .Platform$OS.type == "windows") {
##         ok <- dir.exists(lib)
##         if (ok) {
##             fn <- file.path(lib, paste0("_test_dir_", Sys.getpid()))
##             unlink(fn, recursive = TRUE)
##             res <- try(dir.create(fn, showWarnings = FALSE))
##             if (inherits(res, "try-error") || !res)
##                 ok <- FALSE
##             else unlink(fn, recursive = TRUE)
##         }
##     }
##     if (length(lib) == 1L && !ok) {
##         warning(gettextf("'lib = \"%s\"' is not writable", lib),
```

```
##             domain = NA, immediate. = TRUE)
##         userdir <- unlist(strsplit(Sys.getenv("R_LIBS_USER"),
##             .Platform$path.sep))[1L]
##         if (interactive()) {
##             ans <- askYesNo(gettext("Would you like to use a personal library instead?"),
##                 default = FALSE)
##             if (!isTRUE(ans))
##                 stop("unable to install packages")
##             lib <- userdir
##             if (!file.exists(userdir)) {
##                 ans <- askYesNo(gettextf("Would you like to create a personal library\n%s\nto install packages
into?",
##                   sQuote(userdir)), default = FALSE)
##                 if (!isTRUE(ans))
##                   stop("unable to install packages")
##                 if (!dir.create(userdir, recursive = TRUE))
##                   stop(gettextf("unable to create %s", sQuote(userdir)),
##                     domain = NA)
##                 .libPaths(c(userdir, .libPaths()))
##             }
##         }
##         else stop("unable to install packages")
##     }
##     lib <- normalizePath(lib)
##     if (length(pkgs) == 1L && missing(repos) && missing(contriburl)) {
##         if ((type == "source" && any(grepl("[.]tar[.](gz|bz2|xz)$",
##             pkgs))) || (type %in% "win.binary" && endsWith(pkgs,
##             ".zip")) || (startsWith(type, "mac.binary") && endsWith(pkgs,
##             ".tgz"))) {
##             repos <- NULL
##             message("inferring 'repos = NULL' from 'pkgs'")
##         }
##         if (type == "both") {
##             if (type2 %in% "win.binary" && endsWith(pkgs, ".zip")) {
##                 repos <- NULL
##                 type <- type2
##                 message("inferring 'repos = NULL' from 'pkgs'")
##             }
##             else if (startsWith(type2, "mac.binary") && endsWith(pkgs,
##                 ".tgz")) {
##                 repos <- NULL
##                 type <- type2
##                 message("inferring 'repos = NULL' from 'pkgs'")
##             }
##             else if (grepl("[.]tar[.](gz|bz2|xz)$", pkgs)) {
##                 repos <- NULL
##                 type <- "source"
##                 message("inferring 'repos = NULL' from 'pkgs'")
##             }
##         }
##     }
##     if (length(pkgs) == 1L && is.null(repos) && type == "both") {
##         if ((type2 %in% "win.binary" && endsWith(pkgs, ".zip")) ||
##             (startsWith(type2, "mac.binary") && endsWith(pkgs,
##                 ".tgz"))) {
##             type <- type2
##         }
##         else if (grepl("[.]tar[.](gz|bz2|xz)$", pkgs)) {
##             type <- "source"
##         }
##     }
##     if (is.null(repos) && missing(contriburl)) {
##         tmpd <- destdir
##         nonlocalrepos <- any(web <- grepl("^(http|https|ftp)://",
##             pkgs))
##         if (is.null(destdir) && nonlocalrepos) {
##             tmpd <- file.path(tempdir(), "downloaded_packages")
##             if (!file.exists(tmpd) && !dir.create(tmpd))
##                 stop(gettextf("unable to create temporary directory %s",
##                   sQuote(tmpd)), domain = NA)
##         }
##         if (nonlocalrepos) {
##             df <- function(p, destfile, method, ...) download.file(p,
##                 destfile, method, mode = "wb", ...)
##             urls <- pkgs[web]
##             for (p in unique(urls)) {
##                 this <- pkgs == p
##                 destfile <- file.path(tmpd, basename(p))
##                 res <- try(df(p, destfile, method, ...))
```

```
##                 if (!inherits(res, "try-error") && res == 0L)
##                   pkgs[this] <- destfile
##                 else {
##                   pkgs[this] <- NA
##                 }
##             }
##         }
##     }
##     if (type == "both") {
##         if (type2 == "source")
##             stop("type == \"both\" can only be used on Windows or a CRAN build for macOS")
##         if (!missing(contriburl) || !is.null(available))
##             type <- type2
##     }
##     getDeps <- TRUE
##     if (type == "both") {
##         if (is.null(repos))
##             stop("type == \"both\" cannot be used with 'repos = NULL'")
##         type <- "source"
##         contriburl <- contrib.url(repos, "source")
##         if (missing(repos))
##             repos <- getOption("repos")
##         available <- available.packages(contriburl = contriburl,
##             method = method, fields = "NeedsCompilation", ...)
##         pkgs <- getDependencies(pkgs, dependencies, available,
##             lib, ...)
##         getDeps <- FALSE
##         av2 <- available.packages(contriburl = contrib.url(repos,
##             type2), method = method, ...)
##         bins <- row.names(av2)
##         bins <- pkgs[pkgs %in% bins]
##         srcOnly <- pkgs[!pkgs %in% bins]
##         binvers <- av2[bins, "Version"]
##         hasArchs <- !is.na(av2[bins, "Archs"])
##         needsCmp <- !(available[bins, "NeedsCompilation"] %in%
##             "no")
##         hasSrc <- hasArchs | needsCmp
##         srcvers <- available[bins, "Version"]
##         later <- as.numeric_version(binvers) < srcvers
##         action <- getOption("install.packages.compile.from.source",
##             "interactive")
##         if (!nzchar(Sys.which(Sys.getenv("MAKE", "make"))))
##             action <- "never"
##         if (any(later)) {
##             msg <- ngettext(sum(later), "There is a binary version available but the source version is later",
##                 "There are binary versions available but the source versions are later")
##             cat("\n", paste(strwrap(msg, indent = 2, exdent = 2),
##                 collapse = "\n"), ":\n", sep = "")
##             out <- data.frame(binary = binvers, source = srcvers,
##                 needs_compilation = hasSrc, row.names = bins,
##                 check.names = FALSE)[later, ]
##             print(out)
##             cat("\n")
##             if (any(later & hasSrc)) {
##                 if (action == "interactive" && interactive()) {
##                   msg <- ngettext(sum(later & hasSrc), "Do you want to install from sources the package which
needs compilation?",
##                     "Do you want to install from sources the packages which need compilation?")
##                   res <- askYesNo(msg)
##                   if (is.na(res))
##                     stop("Cancelled by user")
##                   if (!isTRUE(res))
##                     later <- later & !hasSrc
##                 }
##                 else if (action == "never") {
##                   cat("  Binaries will be installed\n")
##                   later <- later & !hasSrc
##                 }
##             }
##         }
##         bins <- bins[!later]
##         if (length(srcOnly)) {
##             s2 <- srcOnly[!(available[srcOnly, "NeedsCompilation"] %in%
##                 "no")]
##             if (length(s2)) {
##                 msg <- ngettext(length(s2), "Package which is only available in source form, and may need comp
ilation of C/C++/Fortran",
##                   "Packages which are only available in source form, and may need compilation of C/C++/Fortran
")
```

```
##                 msg <- c(paste0(msg, ": "), sQuote(s2))
##                 msg <- strwrap(paste(msg, collapse = " "), exdent = 2)
##                 message(paste(msg, collapse = "\n"), domain = NA)
##                 if (action == "interactive" && interactive()) {
##                   res <- askYesNo("Do you want to attempt to install these from sources?")
##                   if (is.na(res))
##                     stop("Cancelled by user")
##                   if (!isTRUE(res))
##                     pkgs <- setdiff(pkgs, s2)
##                 }
##                 else if (action == "never") {
##                   cat("  These will not be installed\n")
##                   pkgs <- setdiff(pkgs, s2)
##                 }
##             }
##         }
##         if (length(bins)) {
##             if (type2 == "win.binary")
##                 .install.winbinary(pkgs = bins, lib = lib, contriburl = contrib.url(repos,
##                   type2), method = method, available = av2, destdir = destdir,
##                   dependencies = NULL, libs_only = libs_only,
##                   quiet = quiet, ...)
##             else .install.macbinary(pkgs = bins, lib = lib, contriburl = contrib.url(repos,
##                 type2), method = method, available = av2, destdir = destdir,
##                 dependencies = NULL, quiet = quiet, ...)
##         }
##         pkgs <- setdiff(pkgs, bins)
##         if (!length(pkgs))
##             return(invisible())
##         message(sprintf(ngettext(length(pkgs), "installing the source package %s",
##             "installing the source packages %s"), paste(sQuote(pkgs),
##             collapse = ", ")), "\n", domain = NA)
##         flush.console()
##     }
##     else if (getOption("install.packages.check.source", "yes") %in%
##         "yes" && (type %in% "win.binary" || startsWith(type,
##         "mac.binary"))) {
##         if (missing(contriburl) && is.null(available) && !is.null(repos)) {
##             contriburl2 <- contrib.url(repos, "source")
##             if (missing(repos))
##                 repos <- getOption("repos")
##             av1 <- tryCatch(suppressWarnings(available.packages(contriburl = contriburl2,
##                 method = method, ...)), error = function(e) e)
##             if (inherits(av1, "error")) {
##                 message("source repository is unavailable to check versions")
##                 available <- available.packages(contriburl = contrib.url(repos,
##                   type), method = method, ...)
##             }
##             else {
##                 srcpkgs <- pkgs[pkgs %in% row.names(av1)]
##                 available <- available.packages(contriburl = contrib.url(repos,
##                   type), method = method, ...)
##                 bins <- pkgs[pkgs %in% row.names(available)]
##                 na <- srcpkgs[!srcpkgs %in% bins]
##                 if (length(na)) {
##                   msg <- sprintf(ngettext(length(na), "package %s is available as a source package but not as
a binary",
##                     "packages %s are available as source packages but not as binaries"),
##                     paste(sQuote(na), collapse = ", "))
##                   cat("\n   ", msg, "\n\n", sep = "")
##                 }
##                 binvers <- available[bins, "Version"]
##                 srcvers <- binvers
##                 OK <- bins %in% srcpkgs
##                 srcvers[OK] <- av1[bins[OK], "Version"]
##                 later <- as.numeric_version(binvers) < srcvers
##                 if (any(later)) {
##                   msg <- ngettext(sum(later), "There is a binary version available (and will be installed) but
the source version is later",
##                     "There are binary versions available (and will be installed) but the source versions are l
ater")
##                   cat("\n", paste(strwrap(msg, indent = 2, exdent = 2),
##                     collapse = "\n"), ":\n", sep = "")
##                   print(data.frame(binary = binvers, source = srcvers,
##                     row.names = bins, check.names = FALSE)[later,
##                     ])
##                   cat("\n")
##                 }
##             }
```

```
##         }
##     }
##     if (.Platform$OS.type == "windows") {
##         if (startsWith(type, "mac.binary"))
##             stop("cannot install macOS binary packages on Windows")
##         if (type %in% "win.binary") {
##             .install.winbinary(pkgs = pkgs, lib = lib, contriburl = contriburl,
##                 method = method, available = available, destdir = destdir,
##                 dependencies = dependencies, libs_only = libs_only,
##                 quiet = quiet, ...)
##             return(invisible())
##         }
##         have_spaces <- grep(" ", pkgs)
##         if (length(have_spaces)) {
##             p <- pkgs[have_spaces]
##             dirs <- shortPathName(dirname(p))
##             pkgs[have_spaces] <- file.path(dirs, basename(p))
##         }
##         pkgs <- gsub("\\", "/", pkgs, fixed = TRUE)
##     }
##     else {
##         if (startsWith(type, "mac.binary")) {
##             if (!grepl("darwin", R.version$platform))
##                 stop("cannot install macOS binary packages on this platform")
##             .install.macbinary(pkgs = pkgs, lib = lib, contriburl = contriburl,
##                 method = method, available = available, destdir = destdir,
##                 dependencies = dependencies, quiet = quiet, ...)
##             return(invisible())
##         }
##         if (type %in% "win.binary")
##             stop("cannot install Windows binary packages on this platform")
##         if (!file.exists(file.path(R.home("bin"), "INSTALL")))
##             stop("This version of R is not set up to install source packages\nIf it was installed from an RPM,
you may need the R-devel RPM")
##     }
##     cmd0 <- file.path(R.home("bin"), "R")
##     args0 <- c("CMD", "INSTALL")
##     output <- if (quiet)
##         FALSE
##     else ""
##     env <- character()
##     tlim <- Sys.getenv("_R_INSTALL_PACKAGES_ELAPSED_TIMEOUT_")
##     tlim <- if (is.na(tlim))
##         0
##     else tools:::get_timeout(tlim)
##     outdir <- getwd()
##     if (is.logical(keep_outputs)) {
##         if (is.na(keep_outputs))
##             keep_outputs <- FALSE
##     }
##     else if (is.character(keep_outputs) && (length(keep_outputs) ==
##         1L)) {
##         if (!dir.exists(keep_outputs) && !dir.create(keep_outputs,
##             recursive = TRUE))
##             stop(gettextf("unable to create %s", sQuote(keep_outputs)),
##                 domain = NA)
##         outdir <- normalizePath(keep_outputs)
##         keep_outputs <- TRUE
##     }
##     else stop(gettextf("invalid %s argument", sQuote("keep_outputs")),
##         domain = NA)
##     if (length(libpath <- .R_LIBS())) {
##         if (.Platform$OS.type == "windows") {
##             oldrlibs <- Sys.getenv("R_LIBS")
##             Sys.setenv(R_LIBS = libpath)
##             on.exit(Sys.setenv(R_LIBS = oldrlibs))
##         }
##         else env <- paste0("R_LIBS=", shQuote(libpath))
##     }
##     if (is.character(clean))
##         args0 <- c(args0, clean)
##     if (libs_only)
##         args0 <- c(args0, "--libs-only")
##     if (!missing(INSTALL_opts)) {
##         if (!is.list(INSTALL_opts)) {
##             args0 <- c(args0, paste(INSTALL_opts, collapse = " "))
##             INSTALL_opts <- list()
##         }
##     }
```

```
##     else {
##         INSTALL_opts <- list()
##     }
##     if (verbose)
##         message(gettextf("system (cmd0): %s", paste(c(cmd0, args0),
##             collapse = " ")), domain = NA)
##     if (is.null(repos) && missing(contriburl)) {
##         update <- cbind(path.expand(pkgs), lib)
##         for (i in seq_len(nrow(update))) {
##             if (is.na(update[i, 1L]))
##                 next
##             args <- c(args0, get_install_opts(update[i, 1L]),
##                 "-l", shQuote(update[i, 2L]), getConfigureArgs(update[i,
##                   1L]), getConfigureVars(update[i, 1L]), shQuote(update[i,
##                   1L]))
##             status <- system2(cmd0, args, env = env, stdout = output,
##                 stderr = output, timeout = tlim)
##             if (status > 0L)
##                 warning(gettextf("installation of package %s had non-zero exit status",
##                   sQuote(update[i, 1L])), domain = NA)
##             else if (verbose) {
##                 cmd <- paste(c(cmd0, args), collapse = " ")
##                 message(sprintf("%d): succeeded '%s'", i, cmd),
##                   domain = NA)
##             }
##         }
##         return(invisible())
##     }
##     tmpd <- destdir
##     nonlocalrepos <- !all(startsWith(contriburl, "file:"))
##     if (is.null(destdir) && nonlocalrepos) {
##         tmpd <- file.path(tempdir(), "downloaded_packages")
##         if (!file.exists(tmpd) && !dir.create(tmpd))
##             stop(gettextf("unable to create temporary directory %s",
##                 sQuote(tmpd)), domain = NA)
##     }
##     av2 <- NULL
##     if (is.null(available)) {
##         filters <- getOption("available_packages_filters")
##         if (!is.null(filters)) {
##             available <- available.packages(contriburl = contriburl,
##                 method = method, ...)
##         }
##         else {
##             f <- setdiff(available_packages_filters_default,
##                 c("R_version", "duplicates"))
##             av2 <- available.packages(contriburl = contriburl,
##                 filters = f, method = method, ...)
##             f <- available_packages_filters_db[["R_version"]]
##             f2 <- available_packages_filters_db[["duplicates"]]
##             available <- f2(f(av2))
##         }
##     }
##     if (getDeps)
##         pkgs <- getDependencies(pkgs, dependencies, available,
##             lib, ..., av2 = av2)
##     foundpkgs <- download.packages(pkgs, destdir = tmpd, available = available,
##         contriburl = contriburl, method = method, type = "source",
##         quiet = quiet, ...)
##     if (length(foundpkgs)) {
##         if (verbose)
##             message(gettextf("foundpkgs: %s", paste(foundpkgs,
##                 collapse = ", ")), domain = NA)
##         update <- unique(cbind(pkgs, lib))
##         colnames(update) <- c("Package", "LibPath")
##         found <- pkgs %in% foundpkgs[, 1L]
##         files <- foundpkgs[match(pkgs[found], foundpkgs[, 1L]),
##             2L]
##         if (verbose)
##             message(gettextf("files: %s", paste(files, collapse = ", \n\t")),
##                 domain = NA)
##         update <- cbind(update[found, , drop = FALSE], file = files)
##         if (nrow(update) > 1L) {
##             upkgs <- unique(pkgs <- update[, 1L])
##             DL <- .make_dependency_list(upkgs, available)
##             p0 <- .find_install_order(upkgs, DL)
##             update <- update[sort.list(match(pkgs, p0)), ]
##         }
##         if (Ncpus > 1L && nrow(update) > 1L) {
```

```
##             tlim_cmd <- character()
##             if (tlim > 0) {
##                 if (nzchar(timeout <- Sys.which("timeout"))) {
##                   tlim_cmd <- c(shQuote(timeout), "-s INT", tlim)
##                 }
##                 else warning("timeouts for parallel installs require the 'timeout' command")
##             }
##             args0 <- c(args0, "--pkglock")
##             tmpd2 <- file.path(tempdir(), "make_packages")
##             if (!file.exists(tmpd2) && !dir.create(tmpd2))
##                 stop(gettextf("unable to create temporary directory %s",
##                   sQuote(tmpd2)), domain = NA)
##             mfile <- file.path(tmpd2, "Makefile")
##             conn <- file(mfile, "wt")
##             deps <- paste(paste0(update[, 1L], ".ts"), collapse = " ")
##             deps <- strwrap(deps, width = 75, exdent = 2)
##             deps <- paste(deps, collapse = " \\\n")
##             cat("all: ", deps, "\n", sep = "", file = conn)
##             aDL <- .make_dependency_list(upkgs, available, recursive = TRUE)
##             for (i in seq_len(nrow(update))) {
##                 pkg <- update[i, 1L]
##                 fil <- update[i, 3L]
##                 args <- c(args0, get_install_opts(fil), "-l",
##                   shQuote(update[i, 2L]), getConfigureArgs(fil),
##                   getConfigureVars(fil), shQuote(fil), ">", paste0(pkg,
##                     ".out"), "2>&1")
##                 cmd <- paste(c("MAKEFLAGS=", tlim_cmd, shQuote(cmd0),
##                   args), collapse = " ")
##                 deps <- aDL[[pkg]]
##                 deps <- deps[deps %in% upkgs]
##                 deps <- if (length(deps))
##                   paste(paste0(deps, ".ts"), collapse = " ")
##                 else ""
##                 cat(paste0(pkg, ".ts: ", deps), paste("\t@echo begin installing package",
##                   sQuote(pkg)), paste0("\t@", cmd, " && touch ",
##                   pkg, ".ts"), paste0("\t@cat ", pkg, ".out"),
##                   "", sep = "\n", file = conn)
##             }
##             close(conn)
##             cwd <- setwd(tmpd2)
##             on.exit(setwd(cwd))
##             status <- system2(Sys.getenv("MAKE", "make"), c("-k -j",
##                 Ncpus), stdout = output, stderr = output, env = env)
##             if (status > 0L) {
##                 pkgs <- update[, 1L]
##                 tss <- sub("[.]ts$", "", dir(".", pattern = "[.]ts$"))
##                 failed <- pkgs[!pkgs %in% tss]
##                 for (pkg in failed) system(paste0("cat ", pkg,
##                   ".out"))
##                 warning(gettextf("installation of one or more packages failed,\n  probably %s",
##                   paste(sQuote(failed), collapse = ", ")), domain = NA)
##             }
##             if (keep_outputs)
##                 file.copy(paste0(update[, 1L], ".out"), outdir)
##             file.copy(Sys.glob(paste0(update[, 1L], "*.zip")),
##                 cwd)
##             file.copy(Sys.glob(paste0(update[, 1L], "*.tgz")),
##                 cwd)
##             file.copy(Sys.glob(paste0(update[, 1L], "*.tar.gz")),
##                 cwd)
##             setwd(cwd)
##             on.exit()
##             unlink(tmpd2, recursive = TRUE)
##         }
##         else {
##             tmpd2 <- tempfile()
##             if (!dir.create(tmpd2))
##                 stop(gettextf("unable to create temporary directory %s",
##                   sQuote(tmpd2)), domain = NA)
##             outfiles <- file.path(tmpd2, paste0(update[, 1L],
##                 ".out"))
##             for (i in seq_len(nrow(update))) {
##                 outfile <- if (keep_outputs)
##                   outfiles[i]
##                 else output
##                 fil <- update[i, 3L]
##                 args <- c(args0, get_install_opts(fil), "-l",
##                   shQuote(update[i, 2L]), getConfigureArgs(fil),
##                   getConfigureVars(fil), shQuote(fil))
```

```
##                 status <- system2(cmd0, args, env = env, stdout = outfile,
##                   stderr = outfile, timeout = tlim)
##                 if (!quiet && keep_outputs)
##                   writeLines(readLines(outfile))
##                 if (status > 0L)
##                   warning(gettextf("installation of package %s had non-zero exit status",
##                     sQuote(update[i, 1L])), domain = NA)
##                 else if (verbose) {
##                   cmd <- paste(c(cmd0, args), collapse = " ")
##                   message(sprintf("%d): succeeded '%s'", i, cmd),
##                     domain = NA)
##                 }
##             }
##             if (keep_outputs)
##                 file.copy(outfiles, outdir)
##             unlink(tmpd2, recursive = TRUE)
##         }
##         if (!quiet && nonlocalrepos && !is.null(tmpd) && is.null(destdir))
##             cat("\n", gettextf("The downloaded source packages are in\n\t%s",
##                 sQuote(normalizePath(tmpd, mustWork = FALSE))),
##                 "\n", sep = "", file = stderr())
##         libs_used <- unique(update[, 2L])
##         if (.Platform$OS.type == "unix" && .Library %in% libs_used) {
##             message("Updating HTML index of packages in '.Library'")
##             make.packages.html(.Library)
##         }
##     }
##     else if (!is.null(tmpd) && is.null(destdir))
##         unlink(tmpd, TRUE)
##     invisible()
## }
## <bytecode: 0x00000296782245e0>
## <environment: namespace:utils>
```

Cargar libreria

```
library(psych)
```

```
##
## Attaching package: 'psych'
```

```
## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```

Cargar la libreria y creación

```
library(readr)
correl <- read_csv("~/GitHub/JPAS_LABS24/INPUT/CUADERNOS MD/correl.csv")
```

```
## Rows: 32 Columns: 5
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## dbl (5): GCI_rank, ICE_rank, GCI_index, ICE_index, LPIBE
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

VIsualizar tabla anterior

```
head(correl)
```

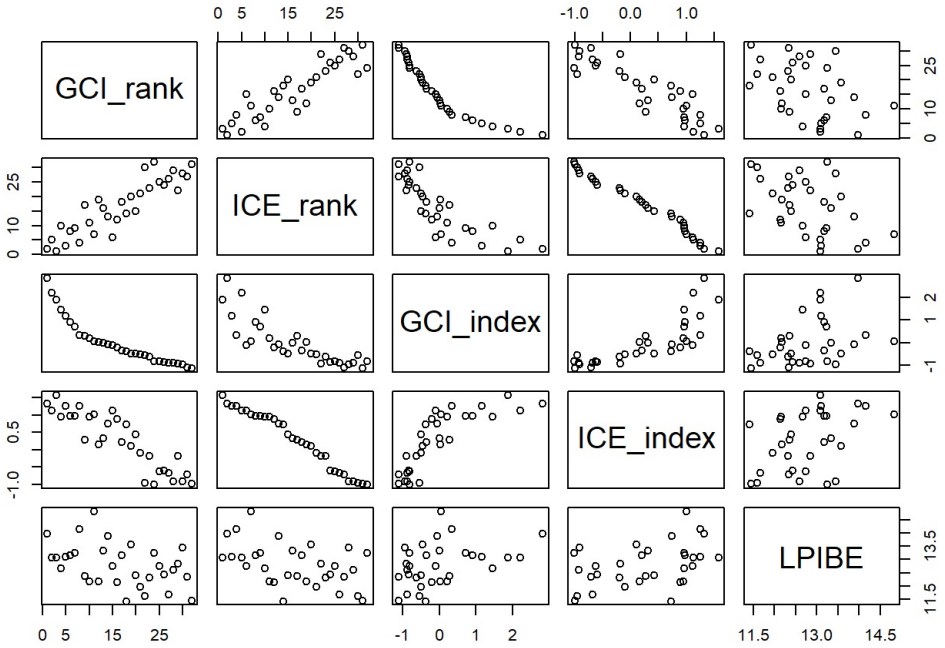
```
## # A tibble: 6 × 5
##   GCI_rank ICE_rank GCI_index ICE_index LPIBE
##      <dbl>    <dbl>     <dbl>     <dbl> <dbl>
## 1       10       11     0.210      0.95  12.2
## 2        2        5     2.20       1.13  13.1
## 3       27       26    -0.885     -0.68  11.7
## 4       30       28    -0.951     -0.91  13.4
## 5       28       29    -0.885     -0.92  12.6
## 6        5        3     1.16       1.25  13.1
```

Correlacionar de forma pariada las variables de la tabla generada

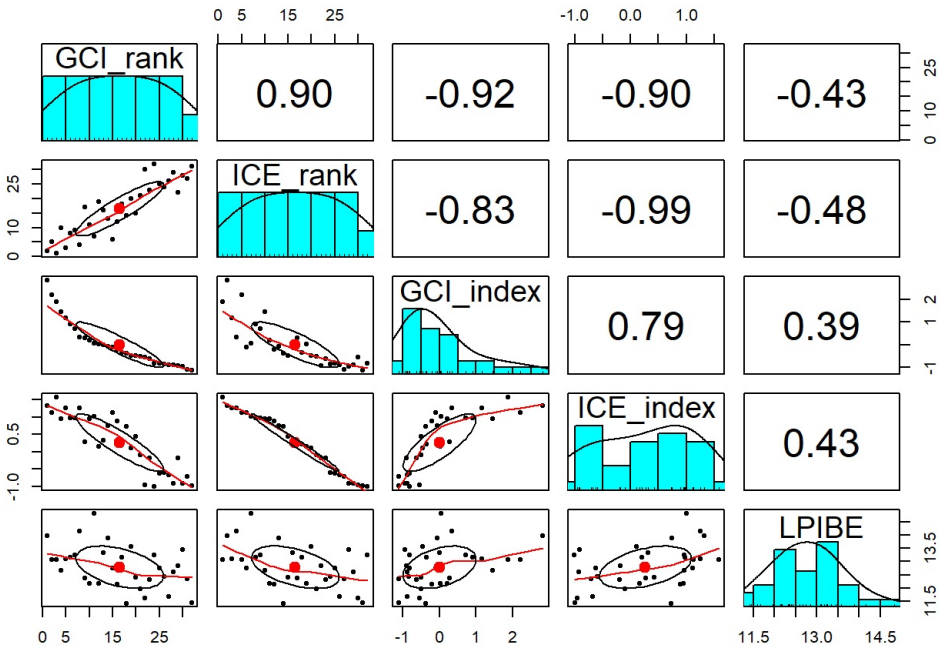
```
attach(correl)
names(correl)
```

```
## [1] "GCI_rank"  "ICE_rank"  "GCI_index" "ICE_index" "LPIBE"
```

```
pairs(correl)
```




```
pairs.panels(correl)
```



correl2 Calculamos la correlación

```
complex_corr <- cor(correl, method = "pearson")
complex_corr
```

```
##             GCI_rank   ICE_rank  GCI_index  ICE_index      LPIBE
## GCI_rank   1.0000000  0.8969941 -0.9166564 -0.8957551 -0.4288321
## ICE_rank   0.8969941  1.0000000 -0.8274473 -0.9875750 -0.4750548
## GCI_index -0.9166564 -0.8274473  1.0000000  0.7946666  0.3905008
## ICE_index -0.8957551 -0.9875750  0.7946666  1.0000000  0.4272882
## LPIBE     -0.4288321 -0.4750548  0.3905008  0.4272882  1.0000000
```

Redondeamos

```
complex_corr =round(complex_corr, digits=2)
complex_corr
```

```
##           GCI_rank ICE_rank GCI_index ICE_index LPIBE
## GCI_rank      1.00     0.90     -0.92     -0.90 -0.43
## ICE_rank      0.90     1.00     -0.83     -0.99 -0.48
## GCI_index    -0.92    -0.83      1.00      0.79  0.39
## ICE_index    -0.90    -0.99      0.79      1.00  0.43
## LPIBE        -0.43    -0.48      0.39      0.43  1.00
```

Gerar mapa de calor Llamar librerias

```
library(ggcorrplot)
library(ggplot2)
```

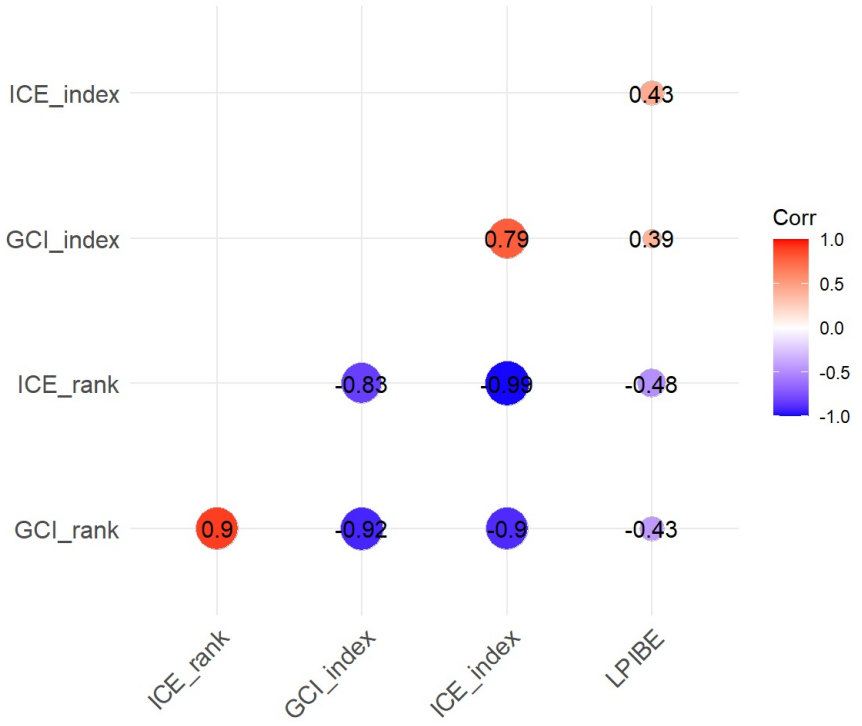
Generar grafico 4

```
p4 <- ggcorrplot(complex_corr, method = "circle", type= "lower", lab= TRUE)
ggtitle("Matriz de correlación")+
theme_minimal()
```

```
## NULL
```

Visualizar resultados

```
p4
```



Para generar un conjunto de graficos en una sola cuadricula Llamar al paquete

```
require(ggpubr)
```

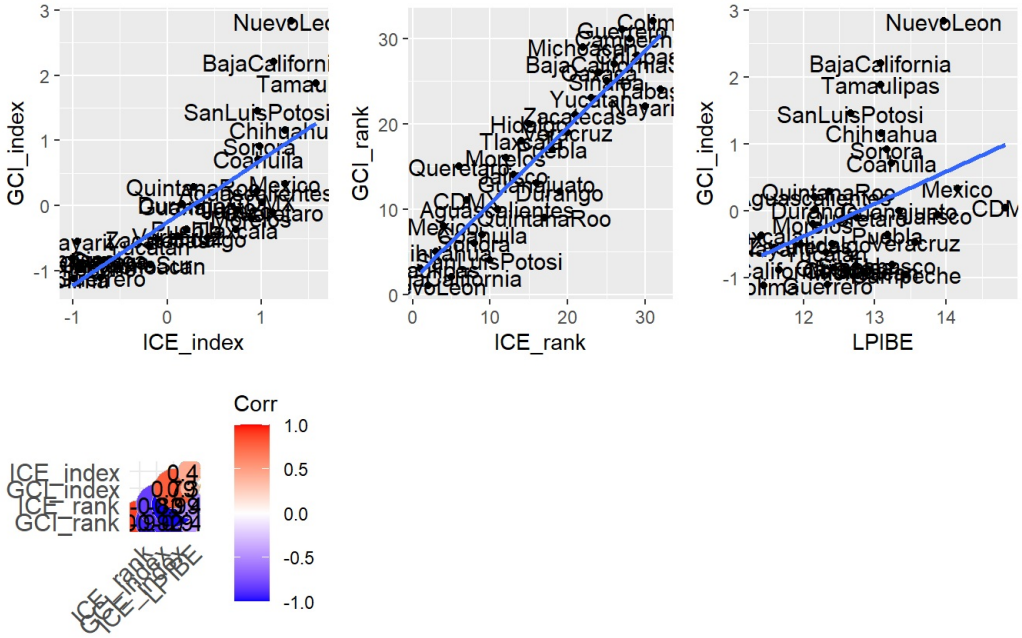
```
## Loading required package: ggpubr
```

```
ggpubr :: ggarrange (p1, p2, p3, p4, etiquetas = c ("A", "B", "C" , "D"))
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

```
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
```

```
## Warning in as_grob.default(plot): Cannot convert object of class character into
## a grob.
```



Visualizacion de un grid

```
library(ggplot2)
require (gridExtra)
```

```
## Loading required package: gridExtra
```

```
F1 <- grid.arrange (p1, p2, p3, p4, nrow = 2)
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

```
## `geom_smooth()` using formula = 'y ~ x'
## `geom_smooth()` using formula = 'y ~ x'
```