`comoRbidity`: an R package for the systematic analysis of disease comorbidities

Alba Gutiérrez-Sacristán   Àlex Bravo   Alexia Giannoula
Miguel A. Mayer   Ferran Sanz   Laura I. Furlong

March 22, 2018

Contents

1 Introduction

The term comorbidity refers to the coexistence or presence of multiple diseases or disorders in relation to a primary disease or disorder in a patient [1, 2]. Clinical and epidemiological studies indicate that disease comorbidities have a great impact on the evolution of health status, selection of appropriate treatments and health system costs [3, 4]. Understanding comorbidities and their etiology is key to identify new preventive and therapeutic strategies.

The goal of the `comoRbidity` R package is to provide a general overview of the disease comorbidities according to an index diseases, from both, clinical and molecular perspective. Analysing patient-level clinical data allows to identify significant comorbidities, while exploring the gene-disease associations will allow to understand the mechanisms underlying comorbidities.

The `comoRbidity` R package is a user-friendly disease comorbidity analysis software. It provides different comorbidity measures as well as visualization of comorbidity results (Figure 1). To identify statistically significant comorbidities, the `comoRbidity` R package uses clinical data, provided by the user, and gene-disease association data based on DisGeNET [12] database (www.disgenet.org)(Figure 1A, 1B). Special effort has been made in the results' visualization. Functions for analysis and visualization of the data previous to the comorbidity analysis are available (Figure 1-3). A variety of plots and heatmaps are provided to display the results of the comorbidity analysis (Figure 1-5).

The tasks that can be performed with the `comoRbidity` package are the following:

1. Age and sex analysis of the population diagnosed with the disease of interest.

2. Clinical comorbidity analysis, based on diagnosis data, in a specific sex and age interval.

3. Analysis of the comorbidity temporal directionality.

4. Molecular comorbidity analysis, based on shared genes.

5. Visualization of the results in a clear, easily interpretable manner.

The `comoRbidity` package also expedites the integration of comorbidity results with other R packages, and allows the development of complex bioinformatics workflows. In the following sections the specific functions that can be used to address each one of these tasks are presented.

1.1 Installation

The package `comoRbidity` is provided via Bitbucket. To install the `comoRbidity` R package the user must type the following commands in an R session:

```
library( "devtools" )
install_bitbucket( "ibi_group/disgenet2r" )
install_bitbucket( "ibi_group/comoRbidity" )
library( "comoRbidity" )
```
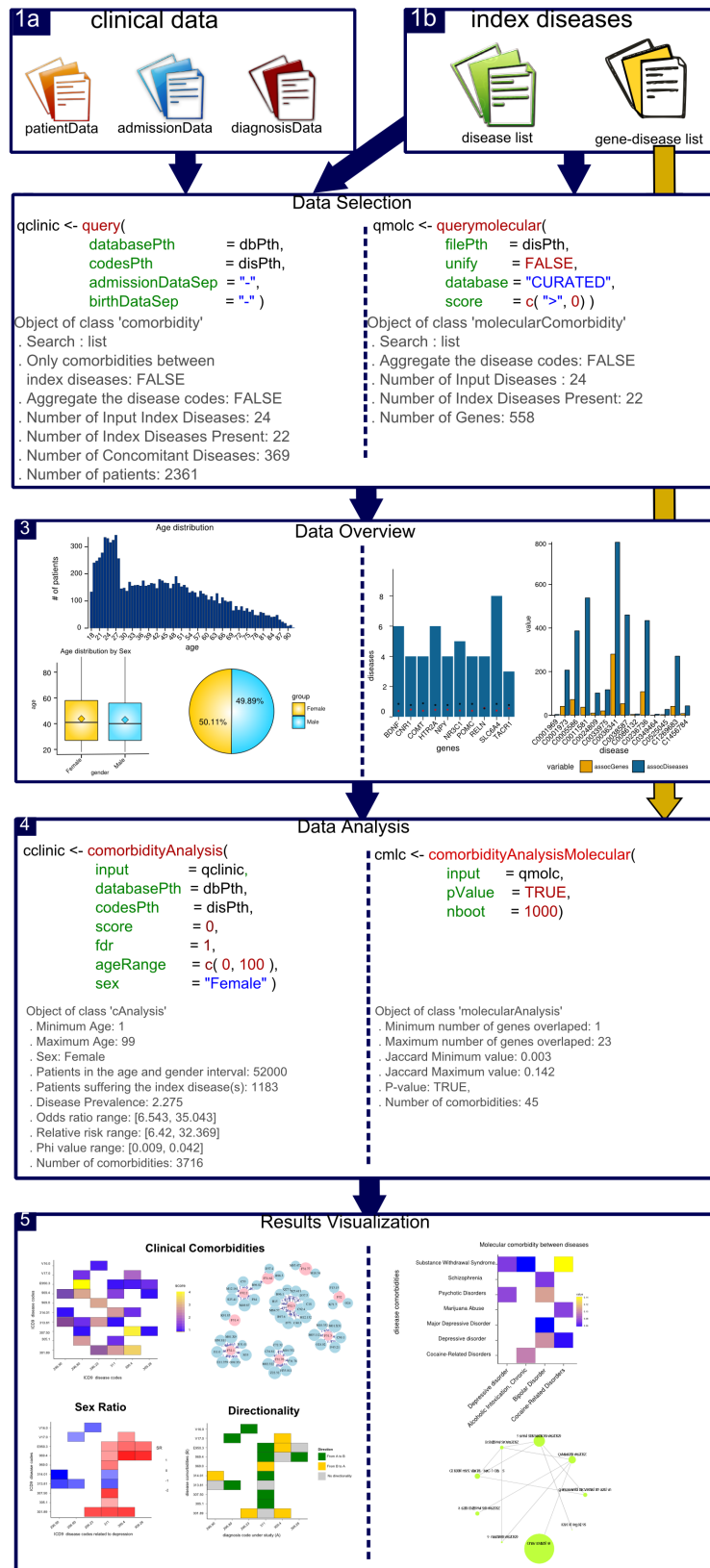
Figure 1: Overview framework of comoRbidity package

1.2 Implementation and Limitations

The `comoRbidity` R package works under:

- **Linux**

- **Mac**

- **Windows** Note that there are some limitations related to two `comoRbidity` R functions:

  - `query` function: set `python` argument to `FALSE`
  - `comorbidityAnalysis` function: set `cores` argument to 1

2 Clinical Comorbidity

The `comoRbidity` R package identifies disease comorbidities using different statistical tests and metrics. The disease comorbidity analysis can be performed stratifying the population by age and/or sex. More importantly, it allows the user to analyze his/her **own clinical record data**, and its own definition of comorbidity (e.g. regarding to the time interval considered). In addition, the `comoRbidity` package allows to perform other analysis like the Sex ratio analysis (section 2.6) and Temporal Directionality of the concomitant diseases (section 2.7).

2.1 Requirements

Four files are required for the comorbidity analysis with the `comoRbidity` package:

- patientData

- diagnosisData

- admissionData

- indexDiseaseCodes

An example of the data is shown in the next subsections. This data has been obtained from an artificial medical data set of 100000 patients with 361760 admissions (http://EMRbots.org).

2.1.1 Patient Data

The `patientData` file must contain at least three predefined columns named as stated below:

- **patient_id**: a patient identifier, that can be numeric, alphanumeric or a list of characters.

- **patient_sex**: the patient sex is required to perform a comorbidity analysis stratified by sex and for the sex ratio analysis. It can be numeric (*e.g., 0 for one sex and 1 for the other*), character (*e.g., M for male and F for female*) or a list of characters (*e.g., male and female*).

- **patient_dateBirth**: the patient birth date is required to calculate the age of the patient when he/she has been diagnosed with a particular disease, following the format year, month, day (YYYY/MM/DD), separated by any type of character.

If the `patientData` file does not contain the required columns the following notification message will appear:

```
## Check the patientData file structure.  Remember that this
## + file must contain at least three columns with the column
## + names as follows:
##  -> patient_id
##  -> patient_sex
##  -> patient_dateBirth
```

```
head( patientData )
```

```
##                          patient_id patient_sex      patient_dateBirth
## 1 F7CF0FE9-AFCD-49EF-BFB3-E42302FFA0D3    Female 1951-07-10 07:29:47.293
## 2 C3935FBC-DBBA-4844-BBE4-A175FA508454      Male 1956-01-27 22:46:39.380
## 3 1CA33F6F-2E84-4C99-AF6A-D40F7B4DB27F      Male 1972-12-22 10:11:01.867
## 4 81606388-2471-42A4-A6F1-1868AE25CFC3      Male 1984-01-17 00:49:06.903
## 5 E3120DE9-3361-40CF-A618-265C769E75A2    Female 1978-12-21 07:24:08.957
## 6 5C043111-3F94-44BC-A889-97D44ACCC7F6    Female 1974-09-25 18:38:02.440
##        PatientRace PatientMaritalStatus PatientLanguage
## 1            Asian               Single         English
## 2 African American               Single         English
## 3            White              Married         English
## 4            Asian            Separated         Spanish
## 5            White              Married         English
## 6 African American              Married         English
##   PatientPopulationPercentageBelowPoverty
## 1                                   13.70
## 2                                   15.73
## 3                                    7.09
## 4                                    2.17
## 5                                   18.67
## 6                                    2.57
```

2.1.2   Diagnosis Data

The diagnosisData file must contain at least three predefined columns named as stated below:

- **patient_id**: a patient identifier, that can be numeric, alphanumeric or a list of characters. The patient identifier must be the same that the one used in the patientData file.

- **admission_id**: an identifier related to the admission or visit, which allows to distinguish among different data entries of the same patient in the database. It can be numeric, alphanumeric or a list of characters.

- **diagnosis_code**: the disease code or codes assigned to an admission or visit (using any vocabulary or standard to identify disease codes). **Note that the index diseases in the indexDiseaseCode file must use the same standard or vocabulary as the diagnoses in diagnosisData file.**

If the diagnosisData file does not contain the required columns the following notification message will appear:

```
## Check the diagnosisData file structure.  Remember that this
##                  file must contain at least three columns with the column
##                  names as follows:
##  -> patient_id
##  -> admission_id
##  -> diagnosis_code
```

```
head( diagnosisData )
```

```
##                          patient_id admission_id diagnosis_code
## 1 54C6E968-45B3-46B1-A64F-2CE3124F2A65            3          F80.1
## 2 54C6E968-45B3-46B1-A64F-2CE3124F2A65            4          R04.81
## 3 54C6E968-45B3-46B1-A64F-2CE3124F2A65            5            I36
## 4 9DD23357-9BEB-43E4-802D-1AB7ACDD4A3A            1          H16.43
```

```
## 5 9DD23357-9BEB-43E4-802D-1AB7ACDD4A3A                  2        M05.161
## 6 9DD23357-9BEB-43E4-802D-1AB7ACDD4A3A                  3        M06.011
##                                              diagnosis_description
## 1                                   Expressive language disorder
## 2                 Acute idiopathic pulmonary hemorrhage in infants
## 3                           Nonrheumatic tricuspid valve disorders
## 4                              Localized vascularization of cornea
## 5 Rheumatoid lung disease with rheumatoid arthritis of right knee
## 6  Rheumatoid arthritis without rheumatoid factor, right shoulder
```

2.1.3 Admission Data

The `admissionData` file must contain at least three predefined columns named as stated below:

- **patient_id**: a patient identifier, that can be numeric, alphanumeric or a list of characters. The patient identifier must be the same one that is used in the patientData file.

- **admission_id**: an identifier related to the admission or visit in which a disease diagnosis was made, that allows to distinguish between different entries of the patient in the database. It can be numeric, alphanumeric or a list of characters.

- **admissionStartDate**: the date of the admission or visit in which the patient was diagnosed with the diseases under study. This information is needed for the directionality analysis. The date has to follow the format year, month, day (YYYY/MM/DD). Any separator charachter is allowed.

If the `admissionData` file does not contain the required columns the following notification message will appear:

```
## Check the admissionData file structure.  Remember that this
##                  file must contain at least three columns with the column
##                  names as follows:
##  -> patient_id
##  -> admission_id
##  -> admissionStartDate
```

```
head( admissionData )

##                              patient_id admission_id admissionStartDate
## 1 9380F9E3-1927-42F3-9731-03A74D4E4C6B            5         2011-03-23
## 2 0A89658C-C739-45CA-9BF1-CBDDDFB922C0            1         1974-02-10
## 3 0A89658C-C739-45CA-9BF1-CBDDDFB922C0            2         1991-05-22
## 4 0A89658C-C739-45CA-9BF1-CBDDDFB922C0            3         1995-02-26
## 5 0A89658C-C739-45CA-9BF1-CBDDDFB922C0            4         2005-03-17
## 6 0A89658C-C739-45CA-9BF1-CBDDDFB922C0            5         2008-04-12
##   admissionEndDate
## 1       2011-03-28
## 2       1974-02-16
## 3       1991-05-29
## 4       1995-02-28
## 5       2005-04-04
## 6       2008-04-16
```

2.1.4 Index Disease Codes

The `indexDiseaseCode` file contains the diseases in which you are interested for the comorbidity analysis. As explained before, the index diseases in the `indexDiseaseCode` file must be in the same

format as the diagnoses in `diagnosisData` file.

`indexDiseaseCode` file must contain at least one predefined columns named as stated below:

- **Code**: the disease code (in any format).

- **Agg**: this column is not compulsory for performing the comorbidity analysis. It must be included if the user wants to group the index disease codes in a higher category or class (e.g., the example below shows all index diseases being grouped in two disease categories, depression (Dep) and bipolar disorder (BD)).

```
head( indexDiseaseCode )

##     Code
## 1    F32
## 2  F30.9
## 3  F32.0
## 4  F32.1
## 5  F32.2
## 6  F32.3
##                                                           Description
## 1                              Major depressive disorder, single episode
## 2                                          Manic episode, unspecified
## 3                        Major depressive disorder, single episode, mild
## 4                    Major depressive disorder, single episode, moderate
## 5 Major depressive disorder, single episode, severe without psychotic features
## 6   Major depressive disorder, single episode, severe with psychotic features
##    Agg
## 1 Dep
## 2 Dep
## 3 Dep
## 4 Dep
## 5 Dep
## 6 Dep
```

> **While the aforementioned columns are required, the files may contain other additional information, as shown in the previous examples. The extra information will no be used by the `comoRbidity` R package.**

2.2 comoRbidity objects

2.2.1 comorbidity object

The `comorbidity` object is obtained when the `query` function is applied. This object is used as input for other functions in the package that enable the user to have an overview of the population under study, including age, sex and diagnosis. `comorbidity` object is also used as input in the function that performs the comorbidity analysis (`comorbidityAnalysis`).

In summary, `comorbidity` object is the input for the following functions:

- comorbidityAnalysis

- summaryDB

- populationAge

- diagnosisUse

The `comorbidity` object contains the query information as well as a summary about the results. It shows:

- the type of search that has been done (`Search`)

- if the comorbidities will be estimated only between the index diseases (`Only comorbidities between index diseases:  TRUE`), or if they will be estimated among the index disease and all the disorders (Only comorbidities between index diseases: FALSE)

- if the individual index disease codes are used for the comorbidity study (`Aggregate the disease codes:  FALSE`) or if they are grouped into a higher category (`Aggregate the disease codes: TRUE`)

- the number of index disorders used as input (`N. Input Index Diseases`)

- the total number of disorders present in this subset (`N. Index Diseases Present`)

- the number of concomitant disorders present in the results (`N. Concomitant Disorders`)

- the patients that suffer at least one of the input disorders (`N. Patients`)

All the `comoRbidity` R package objects come with a function called `extract`. The `extract` function allows the user to retrieve data stored in the object. The `extract` function returns a formatted `data.frame` with the complete set of information obtained from the required data.

```
comor_obj

## Object of class 'comorbidity'
##   . Search:                          list
##   . Only comorbidities between index diseases: FALSE
##   . Aggregate the disease codes: FALSE
##   . Number of Input Index Diseases:   18
##   . Number of Index Diseases Present: 17
##   . Number of Concomintant Diseases:  2498
##   . Number of Patients:               2361
```

2.2.2   cAnalysis object

The `cAnalysis` object is obtained when `comorbidityAnalysis` function is applied. This object is used as input for other functions in the package that enable the user to visualize the results in different graphical ways. Moreover, `cAnalysis` object is used as input for further analysis in the comorbidity results, like the sex ratio analysis and the directionality analysis.

`cAnalysis` object is the input for the following functions:

- `network`

- `heatmapPlot`

- `sexRatio`

- `directionality`

The `cAnalysis` object contains the results of the comorbidity analysis and other relevant information for the user. `cAnalysis` object shows the age interval that has been applied for the analysis (`Age Min` and `Age Max`), the sex, the number of patients that belong to this group from the total data (`Patients in the age and sex interval`) as well as the number of them that suffer the disease of interest (`Patients diagnosed with the index disease(s)`). Other data such as the disease prevalence and the range values obtained for each parameter estimated to measure the comorbidity are also contained in the `cAnalysis` object. Finally, the number of comorbidities that pass the `cutOff` determined by the user are also shown (`Number of comorbidities`).

All the `comoRbidity` R package objects come with a function called `extract`. The `extract` function allows the user to retrieve data stored in the object. The `extract` function

returns a formatted `data.frame` with the complete set of information obtained from the required data.

```
comorMale

## Object of class 'cAnalysis'
##   . Minimum age: 0
##   . Maximum age: 100
##   . Sex   : Male
##   . Patients in the age and sex interval: 48000
##   . Patients suffering the index disease(s): 1178
##   . Disease Prevalence: 2.454
##   . Odds ratio range: [6.183 , 34.553]
##   . Relative risk range: [6.067 , 32.107]
##   . Phi value range: [0.009 , 0.043]
##   . Number of comorbidities: 3694

class(comorMale)

## [1] "cAnalysis"
## attr(,"package")
## [1] "comoRbidity"
```

```
comorbidityData <- extract ( comorMale )
head( comorbidityData )

##       disAcode disBcode disA disB AB AnotB BnotA notAnotB fisher oddsRatio
## 817    F31.11   I25.41   59   78  3    56    75    47866  0.000    34.190
## 108    F31.62      D11   64   75  3    61    72    47864  0.000    32.694
## 1639    F32.4  M05.561   77   64  3    74    61    47862  0.000    31.809
## 1643    F32.1   O10.11   65   46  2    63    44    47891  0.002    34.553
## 620     F32.5    B57.3   53   57  2    51    55    47892  0.002    34.148
## 1608    F32.3   E10.63   51   62  2    49    60    47889  0.002    32.578
##       95%confidenceInterval relativeRisk   phi expect score correctedPvalue
## 817       ( 6.694 , 108.74 )       31.291 0.043  0.096 1.868           0.127
## 108      ( 6.411 , 103.732 )       30.000 0.042  0.100 1.862           0.127
## 1639      ( 6.24 , 100.844 )       29.221 0.041  0.103 1.859           0.127
## 1643      ( 3.97 , 137.665 )       32.107 0.035  0.062 1.498           0.127
## 620      ( 3.929 , 135.706 )       31.778 0.035  0.063 1.497           0.127
## 1608     ( 3.751 , 129.403 )       30.361 0.034  0.066 1.493           0.127
##       sumRank
## 817         1
## 108         2
## 1639        3
## 1643        4
## 620         5
## 1608        6
```

2.3 Data extraction

The first step in order to perform the comorbidity analysis is extracting the data related to the patients diagnosed with the index diseases. These index diseases are determined by the `indexDiseaseCode` file.

The `query` function allows the user to extract the data and store it in a `comorbidity` class object. As input the `query` function requires:

- `databasePth`: determines the path where the three required input files (`patientData`, `diagnosisData`, `admissionData`) are located.

- `codesPth`: determines the path where the file with the index diseases is located (`indexDiseaseCode`).

- `admissionDateSep`: determines what separator symbol is used in the admission date.

- `birthDateSep`: determines what separator symbol is used in the birth date.

Table 1: Optional arguments for data extraction

aggregatedDis argument	intraCodes argument	Description
FALSE	FALSE	Data extraction is done using the Codes column from the index disease file. The comorbidities will be estimated between the index diseases and the rest of diseases that the patient has suffered (*e.g.*, *ff*).
TRUE	FALSE	Data extraction is done using the Agg column from the index disease file, that groups the diseases in a higher class. The comorbidities will be estimated between the index diseases and the rest of diseases that the patient has suffered (*e.g.*, *aggQuery*).
FALSE	TRUE	Data extraction is done using the Codes column from the index disease file. Comorbidities will be estimated only between the index diseases (*e.g.*, *queryIntra*).
TRUE	TRUE	Data extraction is done using the Agg column from the index disease file, that groups the diseases in a higher class. Comorbidities will be estimated only between the index diseases (*e.g.*, *aggQueryIntra*).

As a result, a `comorbidity` object is obtained. This object will contain those patients that have been diagnosed with at least one of the index diseases presented in the `indexDiseaseCode` file, and the data related to them, according to the options selected in the `query` function.

> **Note that the `query` function has an optional argument, `python`, that by default is `FALSE`, but that can be changed to `TRUE` to run the `query` function faster by using python script. In order to use this option it is necesary to have python installed in your computer. Any python release can be used (import sys and import datetime are required).**

In the following we illustrate the different options for data extraction shown in Table 1.
A. aggregatedDis = FALSE intraCodes = FALSE (Default option)
The `databasePth` argument should contain the location of the folder that contains the input files. As an example, it will be used the path where example data is located.

```
databasePth <- system.file("extdata", package="comoRbidity")
diagnosticCodes <- system.file("extdata", package="comoRbidity")
```

The user should indicate his own path following the next structure:

```
databasePthEx <- "/home/user/.../..."
diagnosticCodesEx <- "/home/user/.../..."
```

```
ff <- query( databasePth      = databasePth,
             codesPth         = diagnosticCodes,
             admissionDataSep = "-",
             birthDataSep     = "-"
             )

## Starting querying the index diseases in the dataset
## Loading the input datasets
## Checking the patientData file structure
## Checking the diagnosisData file structure
## Checking the admissionData file structure
```

```
## Checking the patients
## Starting querying for your index diseases
## Generating the resulting objects

ff

## Object of class 'comorbidity'
##   . Search:                         list
##   . Only comorbidities between index diseases: FALSE
##   . Aggregate the disease codes: FALSE
##   . Number of Input Index Diseases:    18
##   . Number of Index Diseases Present: 17
##   . Number of Concomitant Diseases:  2498
##   . Number of Patients:               2361
```

B. aggregatedDis = TRUE intraCodes = FALSE

```
aggQuery <- query( databasePth      = databasePth,
             codesPth        = diagnosticCodes,
             admissionDataSep = "-",
             birthDataSep     = "-",
             aggregatedDis    = TRUE
             )
aggQuery

## Object of class 'comorbidity'
##   . Search:                         list
##   . Only comorbidities between index diseases: FALSE
##   . Aggregate the disease codes: TRUE
##   . Number of Input Index Diseases:    2
##   . Number of Index Diseases Present: 2
##   . Number of Concomitant Diseases:  2483
##   . Number of Patients:               2361
```

C. aggregatedDis = FALSE intraCodes = TRUE

```
queryIntra <- query( databasePth      = databasePth,
             codesPth        = diagnosticCodes,
             admissionDataSep = "-",
             birthDataSep     = "-",
             intraCodes       = TRUE,
             aggregatedDis    = FALSE
             )
queryIntra

## Object of class 'comorbidity'
##   . Search:                         list
##   . Only comorbidities between index diseases: TRUE
##   . Aggregate the disease codes: FALSE
##   . Number of Input Index Diseases:    18
##   . Number of Index Diseases Present: 17
##   . Number of Concomitant Diseases:  17
##   . Number of Patients:               2361
```

D. aggregatedDis = TRUE intraCodes = TRUE

```
aggQueryIntra <- query( databasePth      = databasePth,
             codesPth        = diagnosticCodes,
             admissionDataSep = "-",
```

```
            birthDataSep    = "-",
            intraCodes      = TRUE,
            aggregatedDis   = TRUE
            )
aggQueryIntra

## Object of class 'comorbidity'
##   . Search:                         list
##   . Only comorbidities between index diseases: TRUE
##   . Aggregate the disease codes: TRUE
##   . Number of Input Index Diseases:    2
##   . Number of Index Diseases Present: 2
##   . Number of Concomintant Diseases:   2
##   . Number of Patients:              2361
```

2.4 Overview of the clinical data

2.4.1 Summary DB

The `comoRbidity` R package allows the user to analyze and characterize the population under study, by using the `summaryDB` function.

As an input, the `summaryDB` function requires:

- `input`: a `comorbidity` object, obtained after applying the `query` function.

- `maleCode`: the symbol which denotes males in users' database (e.g., 0, M, Male...etc)

- `femaleCode`: the symbol which denotes females in users' database (*e.g., 1, F, Female...etc*)

The output of the `summaryDB` function is a plot with three different graphics (Figure 2):

- A barplot with the age distribution of the patients diagnosed with the disease of interest.

- A boxplot showing the age distribution by sex.

- A pie chart representing the sex distribution.

```
summaryDB(input = ff,
          maleCode = "Male",
          femaleCode ="Female")
                        ## Checking the input object
```
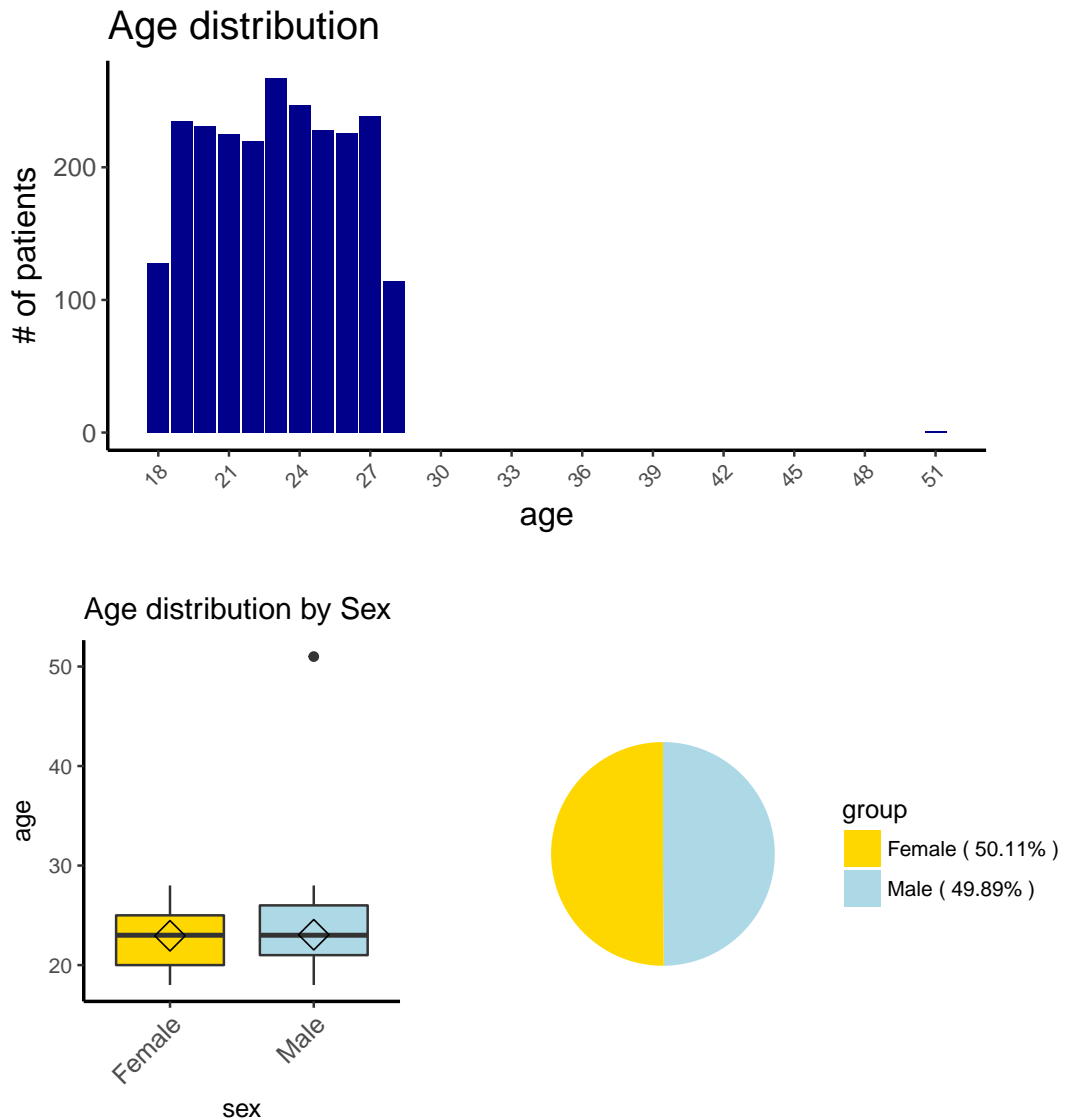

Figure 2: Summary plot containing: age distribution, age distribution by sex and sex distribution.

Note that the color of the barplot, the boxplot and the pie chart can be changed adding the next arguments to the `summaryDB` function:

- `barColor`: By default `"darkblue"`. It defines the bar color in the age distribution bar-plot.

- `femaleColor`: By default `"gold"`. It defines the heatmap color for the female value.

- `maleColor`: By default `"lightblue"`. It defines the color for male values.

2.4.2 Population analysis based on the disease under study

Expanding on the age analysis, the `comoRbidity` R package also allows to analyze the age distribution of patients diagnosed with the index disease(s) compared with all the patients of the database. The

age of diagnosis with the index disease is the age of the patient when he/she is diagnosed by the first time with that disease. For the rest of patients, the age considered is the one at the first admission of the patient in the database.

As an input, `populationAge` function requires:

- `input`: a `comorbidity` object, obtained with the `query` function.

- `codesPth`: determines the path where the file with the index diseases is located (`indexDiseaseCode`).

- `databasePth`: determines the path where the three required input files (`patientData, diagnosisData, admissionData`) are located.

The `populationAge` function has four additional arguments, that are optional for the user:

- `type` argument allows the user to select the output barplot. By default the type is `"together"` (Figure 3), but it can be set to `"separate"` (Figure 4).

- `interactive` argument allows to create an interactive barplot, that show you the specific information of each bar in the barplot in an interactive way.

- `disorderColor` argument allows to define the bar color representing the disease patients. By default gold.

- `allColor` argument allows to define the bar color representing the whole population. By default blue.

The results of the population analysis can be visualized together (Figure 3) or separately. **Note that the patient age in the whole population is estimated taking into account the first admission date of the patient to the database while the age for index disorder patients is estimated taking into account the first admission date in which the patient has been diagnosed with the index disease.**

```
populationAge ( input      = ff,
                codesPth   = diagnosticCodes,
                databasePth = databasePth,
                type       = "together",
                interactive = FALSE)
                           ## Checking the input object
```
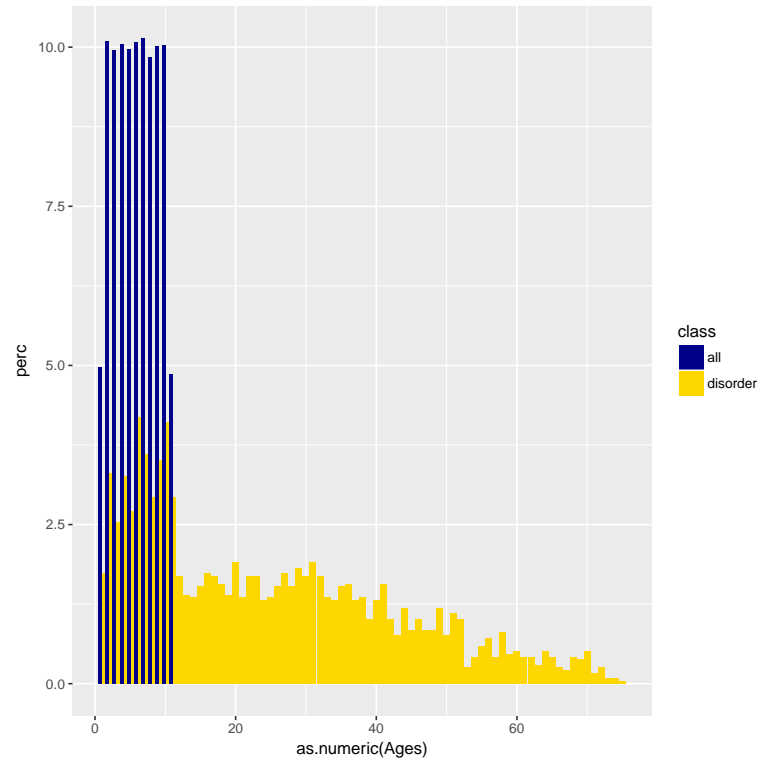

Figure 3: Barplot representing the age-distribution of the patients suffering the disorder of interest vs all population age.

```
populationAge ( input      = ff,
                codesPth    = diagnosticCodes,
                databasePth = databasePth,
                type        = "separate",
                interactive = FALSE)
                            ## Checking the input object
```
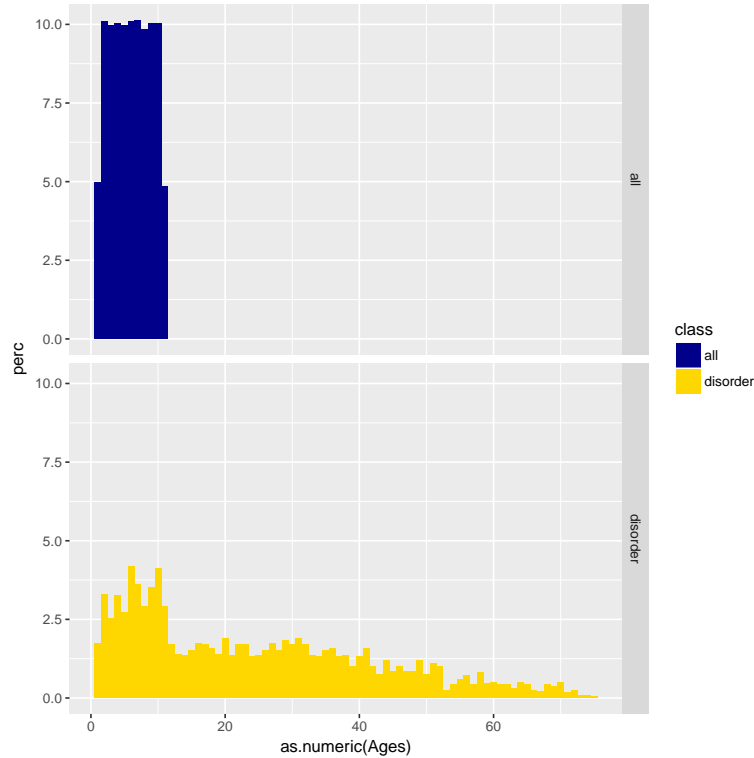

Figure 4: Separate barplot comparing the age-distribution of the patients diagnosed with the disorder of interest vs all population age.

2.4.3 Disease Prevalence

The `comoRbidity` R package allows the user to calculate the disease prevalence. In order to obtain this information, the user can apply the `diseasePrevalence` function.

As a input, the `diseasePrevalence` function requires:

- `input`: a `comorbidity` object, obtained after applying the `query` function.

- `maleCode`: the symbol which denotes males in users' database (e.g., 0, M, Male...etc)

- `femaleCode`: the symbol which denotes females in users' database (*e.g., 1, F, Female...etc*)

- `databasePth`: determines the path where the three required input files (`patientData, diagnosisData, admissionData`) are located.

The output of the `diseasePrevalence` function is a barplot showing the disease prevalence in the entire population and the disease prevalence according to sex (Figure 5):

```
diseasePrevalence(input = ff,
          maleCode = "Male",
          femaleCode ="Female",
          databasePth = databasePth)
                    ## Checking the input object
```
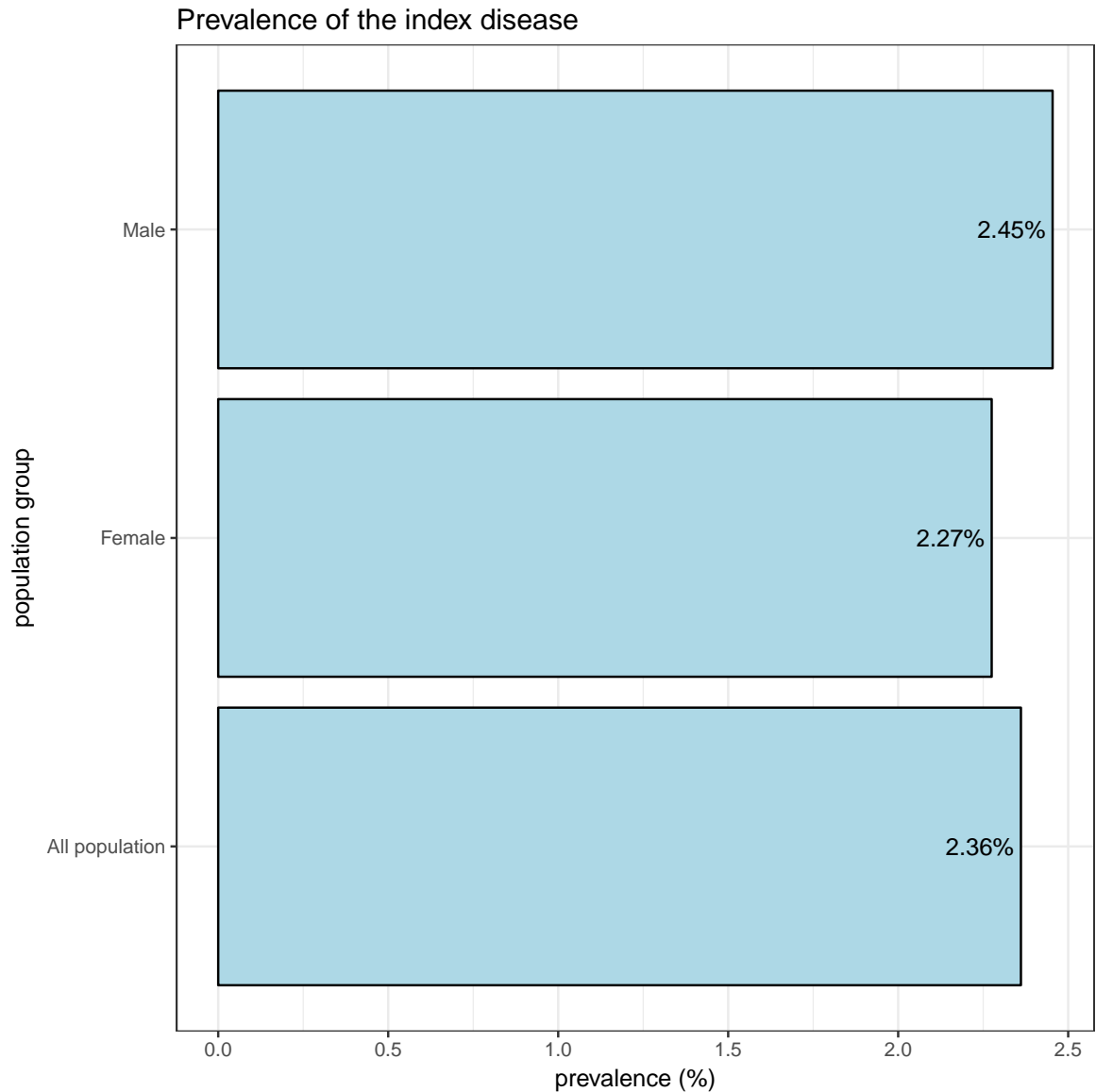
Prevalence of the index disease

Figure 5: Barplot showing the disease prevalence in all the population and also, according to the sex.

Note that the color of the barplot can be changed adding the `barColor` argument to the `diseasePrevalence` function. By default `"blue"`.

2.4.4   Code Use

When studying a disorder that is defined by more than one diagnosis code, the `comoRbidity` R package allows to analyze the percentage of use of each one of the index disease codes (Figure 6).
As an input, `diagnosticUse` function requires:

- `input`: a `comorbidity` object, obtained with the `query` function.

- `codesPth`: determines the path where the file with the index diseases is located (`indexDiseaseCode`).

The `diagnosticUse` function has two more arguments that are optional for the user:

- **cutOff** argument allows the user to select those index disease codes that will be represented in the output barplot. By default the **cutOff** is set as 0, but it can be set to any other percentage value.

- **interactive** argument allowsthe user to create an interactive barplot, which shows the specific information of each bar in the barplot in an interactive way. By default the **interactive** argument is set as **FALSE**.

```
diagnosticUse( codesPth    = diagnosticCodes,
               input       = ff,
               cutOff      = 0,
               interactive = FALSE
               )
                            ## Checking the input object
```
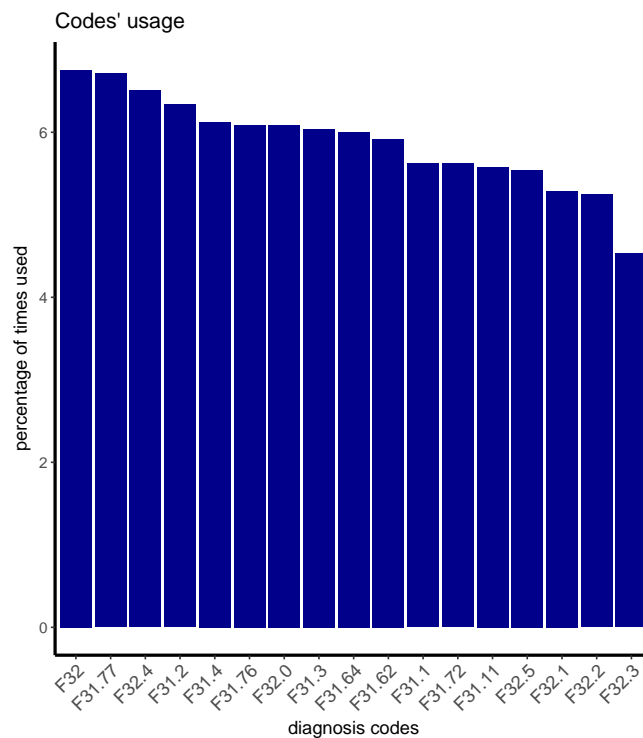

Figure 6: Index disease codes usage in percentage

Note that the color of the barplot can be changed adding the `barColor` argument to the `diagnosticUse` function. By default `"darkblue"`.

If `aggregatedDis` argument has been set to `TRUE`, the graphic will show the percentage of use of each disease category, in this case bipolar disorder (BD) and depression (Dep) (Figure 7).

```
diagnosticUse( codesPth     = diagnosticCodes,
               input        = aggQueryIntra,
               cutOff       = 0,
               interactive  = FALSE
               )
                          ## Checking the input object
```
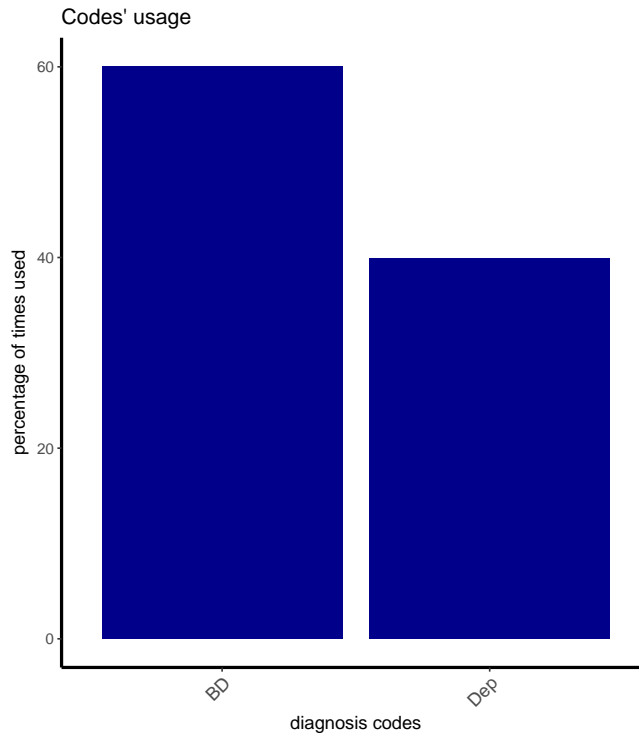

Figure 7: Index disease codes usage in percentage when aggregation has been performed

2.5   Clinical Comorbidity Analysis

Once we have a general overview of the patients diagnosed with the index diseases in the database and the main characteristics of this population, the next step is to perform the comorbidity analysis.

The user can estimate the statistically significant comorbidities by applying the `comorbidityAnalysis` function to the `comorbidity` object previously generated with the `query` function.

2.5.1   Comorbidity Measurements

The `comoRbidity` R package estimates several measures to determine if two diseases are comorbid in the population under study: Risk Ratio (RR), Odds Ratio (OR), Comorbidity score, Phi value. In addition, a Fisher exact test can be applied to determine the significance of the associations.

Relative risk or Risk Ratio (RR) The Relative Risk or Risk Ratio (RR) expresses the relationship between disease A and disease B as a ratio measure of effect (or risk) on the disease prevalence. If we consider that disease A is the outcome variable and disease B the exposure variable, we can ask if a diagnosis of disease B increases the risk of having a diagnosis of disease A. In other words, if disease A co-occurs with disease B more frequently than expected as if they were independent in this population.

The RR is estimated as the fraction between the number of patients diagnosed with both diseases and random expectation based on disease prevalence, as described in [2]:

$$RR_{AB} = \frac{C_{AB}N}{P_A P_B} \qquad (1)$$

where C_{AB} is the number of patients diagnosed with both diseases, N is the total number of patients in the population and P_A and P_B are the prevalences of diseases A and B.

The RR can take the following values:

- RR = 1: Disease A and disease B are independent regarding prevalence.

- RR > 1: Disease A and disease B co-occur more often than expected by chance.

- RR < 1: Disease A and disease B co-occur less often than expected by chance.

Odds ratio The odds ratio represents the increased chance that someone suffering disease A will have the comorbid disorder B. It shows the extent to which suffering a disorder increases the risk of developing another illness or disorder. The odds ratio is derived from a comparison of rates of the illness among individuals who do and do not exhibit the factor of interest. A statistically significant odds ratio (significantly different from 1.00 at the .05 level) indicates an appreciable risk associated with a particular factor. For example, an odds ratio of 2.00 indicates a doubled risk of the appearance of the disorder. The 95% confidence interval is also shown in the output resultant table.

$$OR_{AB} = \frac{C_{AB}H}{C_A C_B} \qquad (2)$$

where C_{AB} is the number of patients diagnosed with both diseases, H is the number of patients without any of the diseases and C_A and C_B are the number of patients diagnosed with diseases A and B respectively.

Comorbidity score This score is defined in Roque et al. as follows [14]:

$$comorbidity score = log_2 \left(\frac{observed + 1}{expected + 1} \right), expected = \frac{P_A P_B}{N} \qquad (3)$$

where *observed* stands for the number of patients diagnosed with both diseases (disease A and disease B), and *expected* is estimated based on the prevalence of each disease (prevalence of disease A, P_A, multiplied by the prevalence of disease B, P_B, and divided by the total number of patients, N). A pseudocount of 1 is added to correct bias of the Comorbidity score towards low prevalent diseases. Since the logarithm is applied, a comorbidity score of 1.0 means that the observed comorbidities are higher than two fold (approximately) than expected.

Phi value (Pearsons correlation for binary variables) measures the robustness of the comorbidity association. It can be expressed mathematically as:

$$\phi_{AB} = \frac{C_{AB}N - P_A P_B}{\sqrt{P_A P_B (N - P_A)(N - P_A)}} \qquad (4)$$

where N is the total number of patients in the population, P_A i and P_B are prevalences of diseases A and B respectively. C_{AB} is the number of patients that have been diagnosed with both diseases A and B, and P_A P_B is the random expectation based on disease prevalence. The Pearson correlation coefficient, can take a range of values from +1 to -1:

- $\phi_{AB} = 0$: indicates that there is no correlation between the two diseases.

- $\phi_{AB} > 0$: indicates a positive correlation between the two diseases.

- $\phi_{AB} < 0$: indicates a negative correlation.

Fisher test A Fisher exact test for each pair of diseases is performed to assess the null hypothesis of independence between the two diseases. Four groups of patients are defined in order to perform the statistical testing: patients diagnosed with disease A and disease B, patients diagnosed with disease A but not disease B, patients diagnosed with disease B but not disease A and patients not diagnosed with disease A nor disease B. The Fisher exact test is then applied to estimate the

p-value for each pair of diseases. The Benjamini-Hochberg [13] false discovery rate method ("fdr") is applied to correct for multiple testing by default. However user can select the best correction method for the analysis changing the `correctionMethod` argument. The adjustment methods include the Bonferroni correction ("bonferroni"), Holm correction ("holm"), Hochberg correction ("hochberg"), Hommel ("hommel") and Benjamini Yekutieli ("BY").

These measures allow the user to quantify the co-occurrence of disease pairs compared with the random expectation. The user can select the measure and the cut-off value in order to asses disease comorbidity.

> **Note that the output data frame containing all the comoribidity measurements is sorted by a combined rank of all the different values as described in [16]. In order to estimate the combined rank, and similarly to what is used in many ranking procedures, first we rank each different comorbidity measurement, and subsequently we sum the different rank positions for each measurement.**

`comorbidityAnalysis` function allows the user to perform the comorbidity analysis and store it in a `cAnalysis` class object. As input the function requires:

- `input`: a `comorbidity` object obtained after applying the `query` function.

- `codesPth`: determines the path where the file with the index diseases is located (`indexDiseaseCode`).

- `databasePth`: determines the path where the three required input files (`patientData`, `diagnosisData`, `admissionData`) are located.

- `ageRange`: determines what is the age range of interest for performing the comorbidity analysis. By default it is set from 0 to 100 years old.

- `sex`: determine what is the sex of interest for performing the comorbidity analysis.

Moreover, the `comorbidityAnalysis` function allows to restrict the results according to the comorbidity measurements values:

- score

- correctedPval

- oddsRatio

- rr

- phi

The user can filter the results by applying all the comorbidity measurements that are considered necesary. The cut-off value for these measurements must be numeric. The example below shows a query in which two of the five comorbidity measurements are applied, the `score` and the `correctedPval`.

```
comorFemale <- comorbidityAnalysis( input         = ff,
                                    codesPth         = diagnosticCodes,
                                    databasePth      = databasePth,
                                    score            = 0,
                                    correctionMethod = "fdr",
                                    correctedPval    = 1,
                                    ageRange         = c( 0, 100 ),
                                    sex              = "Female",
                                    verbose          = FALSE
                                    )
save(comorFemale, file=paste0(databasePth, "comorFemale.RData"))
```

As a result, a `cAnalysis` object is obtained. This object contains a summary of the comorbidities that have been found.

```
load(system.file("extdata", "comorFemale.RData", package="comoRbidity"))
comorFemale

## Object of class 'cAnalysis'
##   . Minimum age: 0
##   . Maximum age: 100
##   . Sex   : Female
##   . Patients in the age and sex interval: 52000
##   . Patients suffering the index disease(s): 1183
##   . Disease Prevalence: 2.275
##   . Odds ratio range: [6.543 , 35.043]
##   . Relative risk range: [6.42 , 32.369]
##   . Phi value range: [0.009 , 0.042]
##   . Number of comorbidities: 3716
```

All the `comoRbidity` **R package objects come with a function called** `extract`. **The** `extract` **function allows the user to retrieve data stored in the object. The** `extract` **function returns a formatted** `data.frame` **with the complete set of information obtained from the comorbidity analysis results.**

```
comorbidityDataFem <- extract ( comorFemale )
head( comorbidityDataFem )

##      disAcode disBcode disA disB AB AnotB BnotA notAnotB fisher oddsRatio
## 328   F31.62    D37.4   77   63  3    74    60    51863  0.000    35.043
## 1017  F31.72      Z11   63   51  2    61    49    51888  0.002    34.719
## 1521   F32.4   C40.81   78   72  3    75    69    51853  0.000    30.060
## 913    F32.3    C40.3   57   59  2    55    57    51886  0.002    33.101
## 1191  F31.11   E10.52   74   78  3    71    75    51851  0.000    29.212
## 490    F32.1   D31.01   61   56  2    59    54    51885  0.002    32.571
##      95%confidenceInterval relativeRisk   phi expect score correctedPvalue
## 328     ( 6.872 , 111.155 )      32.158 0.042  0.093 1.871           0.124
## 1017    ( 3.999 , 137.715 )      32.369 0.034  0.062 1.498           0.124
## 1521     ( 5.918 , 95.039 )      27.778 0.039  0.108 1.852           0.124
## 913     ( 3.818 , 131.219 )      30.925 0.033  0.065 1.495           0.124
## 1191     ( 5.755 , 92.201 )      27.027 0.038  0.111 1.848           0.124
## 490     ( 3.758 , 128.809 )      30.445 0.033  0.066 1.493           0.124
##      sumRank
## 328      1.0
## 1017     2.0
## 1521     3.0
## 913      4.0
## 1191     5.5
## 490      5.5
```

2.5.2 Clinical Comorbidity Visualization

In order to visualize the comorbidity analysis results, `comoRbidity` package provides two different options:

- Network: obtained by applying `network` function.

- Heatmap: obtained by applying `heatmapPlot` function.

Comorbidity Network

`network` function allows the user to visualize the data contained in the `cAnalysis` object obtained after applying the `comorbidityAnalysis` function.

As input the `network` function requires:

- input: a `cAnalysis` object obtained after applying the `comorbidityAnalysis` function.

- `databasePth`: determines the path where the three required input files (`patientData`, `diagnosisData`, `admissionData`) are located.

- `layout`: by deafult `"layout.fruchterman.reingold"`. It can be set to other of the possible igraph layouts.

- `selectValue`: By default `"score"` variable will be selected. It can be set to any of the other possible variables ('correctedPval', 'odds ratio', 'phi', 'rr').

- `cutOff`: By default '0.05'. The value of the argument can be changed to any other numeric variable, according to the range of the selected value.

- `npairs`: By default '0'. The value of the argument can be changed to any other numeric variable to show in the network only those comorbidities suffered by at least `npairs` of patients.

- `prop`: Determines the node size proportionality. By default it is set to 1. The value of the argument can be changed to any other numeric variable.

- `title`: Determines the title of the network figure. By default 'Comorbidity network'.

- `interactive` Determines if the output network is interactive or not. By default the `interactive` argument is set to `FALSE`. The value of the argument can be changed to `TRUE`, as a result an interactive network will be obtained.

```
load(system.file("extdata", "comorFemale.RData", package="comoRbidity"))
network ( input = comorFemale,
         databasePth = databasePth,
         layout = "layout.fruchterman.reingold",
         selectValue = "score",
         cutOff = 1.45,
         npairs = 2,
         prop  = 1,
         title = "Female comorbidity network",
         interactive = FALSE
       )
```

As a result, a network is obtained (Figure 8). The nodes in pink belong to the disorder of interest, while the blue nodes correspond to the comorbidity disorders. Note that the color of the nodes can be changed by adding the following arguments to the `network` function:

- `diseaseColor`: determines the node color for the disorder of interest. By default it is set to `"pink"`.

- `comorColor`: determines the node color for the comorbid disorders. By default it is set to `"lightblue"`.
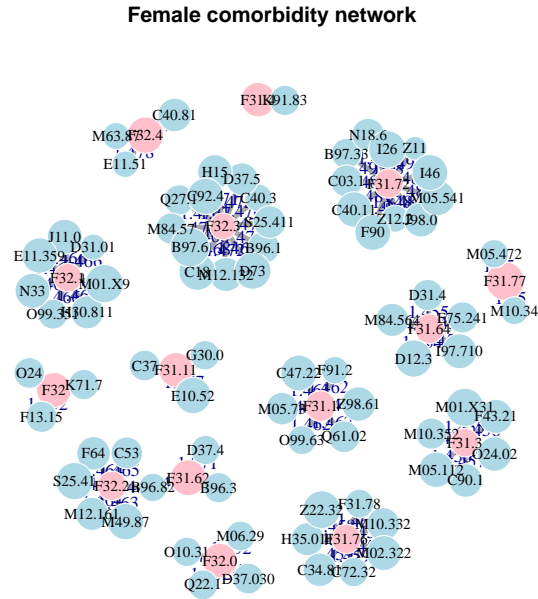
Female comorbidity network

Figure 8: Comorbidity network in female population

The `comoRbidity` package also allows to visualize `cAnalysis` object in a heatmap (Figure 9). The required input is the same as for the `network` function.

```
heatmapPlot( input       = comorFemale,
             selectValue = "score",
             npairs      = 2,
             cutOff      = 1.45,
             verbose     = FALSE,
             interactive = FALSE)
```

Note that the color of the heatmap can be changed adding the next arguments to the `heatmaPlot` function:

- `lowColor`: By default `"0000FF"`. It defines the heatmap color for the lowest value.

- `highColor`: By default `"yellow"`. It defines the heatmap color for the highest value.
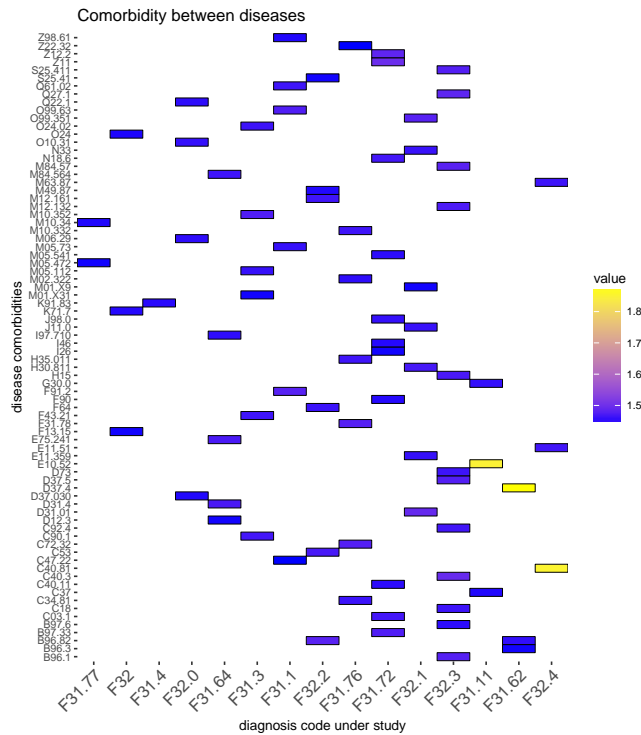
Figure 9: Comorbidity heatmap in female population

2.6 Sex ratio analysis

The `comoRbidity` package also estimates the sex ratio (SR) parameter. The sex ratio (SR) parameter allows to see if a comorbidity suffered in both, men and women, is equally likely in both sex or if it is more likely in one sex than in another. For a comorbidity A and age group t, SR (2.6) is defined by Klimek et al. [15] as follows:

$$SR(A, B) = log\left(\frac{1 + \dfrac{D_f(B)}{D_f(A, B)}}{1 + \dfrac{D_m(B)}{D_m(A, B)}}\right)$$

where Dm(f)(B) stands for the number of patients (men or women) suffering disease B in age group t, and Dm(f)(A,B) denotes those patients suffering disease B who also have been diagnosed with a disease A. SR values close to 0 mean that the comorbidity is equally likely for men and women. Positive SR values indicate that the comorbidity is more likely for women, while negative SR values indicate that the comorbidity is more likely in men.

To obtain the sex ratio heatmap, two steps should be followed:

1. Apply the `sexRatio` function using as input the `cAnalysis` object obtained for both sex.

2. Apply the `heatmapSexRatio` to the previous results. `interactive` argument can be set to `TRUE` if an interactive heatmap is required.

In the next example, a filter to the SR results has been applied to show only extreme SR values.

26

```
srAnalysis <- sexRatio(female    = comorFemale,
                       male      = comorMale,
                       fisherTest = 0
                       )


srAnalysis <- srAnalysis[as.numeric(srAnalysis$SR) <= -0.5
                         | as.numeric(srAnalysis$SR) >= 0.5,]
```

```
heatmapSexRatio(srAnalysis,
                interactive = FALSE
                )
```


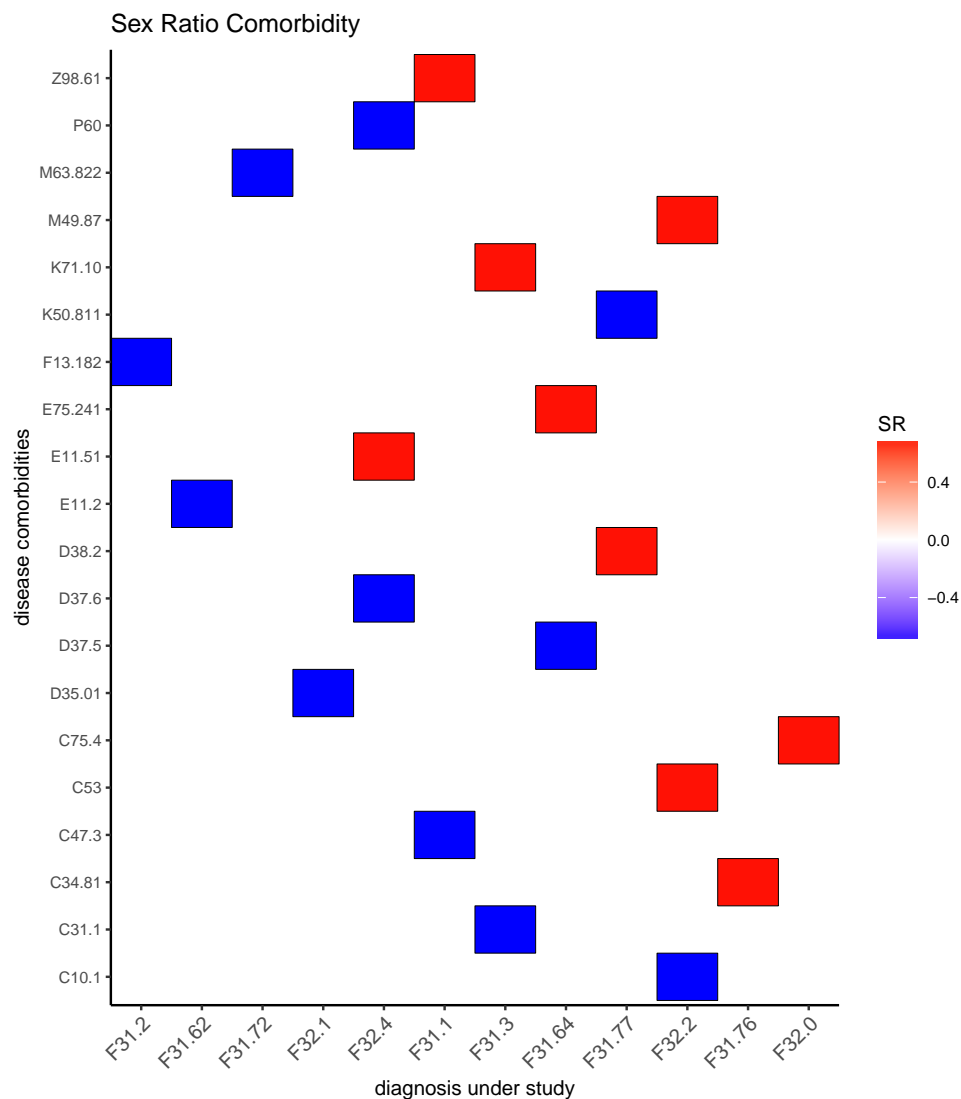Figure 10: Sex ratio (SR) comorbidities heatmap

As a result a heatmap (Figure 10) is obtained. In red (positive values) we show these comorbidities that are more likely for women, while in blue (negative values) we show those that are more likely for men. These colors can be changed adding the following arguments:

- maleColor Determines the heatmap color for those comorbidities that ' are more likely in men

than women. By default `"blue"`.

- `femaleColor` Determines the heatmap color for those comorbidities that ' are more likely in women than men. By default `"red"`.

2.7 Directionality analysis

Additionally, the `comoRbidity` package allows to analyze the temporal directionality of the co-occurring diseases (Figure 11).

The temporal direction of disease association ($d_A \rightarrow d_B$ and $d_B \rightarrow d_A$) is assessed for the diagnosis pairs with a significant corrected p-value. Specifically, the number of patients for whom diagnosis dB follows diagnosis dA or vice versa, is calculated and an exact binomial test is, subsequently, used with a probability of success equal to 0.5. A preferred (significant) direction is determined for those diagnosis pairs that result in binomial tests with p-values $<$ 0.05 and according to the pair that appears more often.

The `directionality` function allows the user to perform the directionality analysis. As input the function requires:

- `input`: an object of class `cAnalysis`.

- `databasePth`: determines the path where the three required input files (`patientData, diagnosisData, admissionData`) are located.

- `minPairs`: determines the minimum number of patients that must suffer the comorbidity to take them into account for the directionality analysis. By default the minPairs value is set to 1.

- `sex`: determines what is the sex of interest for performing the directionality analysis.

- `ageRange`: determines what is the age range of interest for performing the directionality analysis. By default it is set from 0 to 100 years old.

- `days`: determines the number of days of difference needed for considering two diseases as comorbid for the directionality analysis.

- `dataSep`: determines the separator symbol used in the admission date.

- `correctionMethod`: determines the correction method applied by the user. The Bonferroni correction (`"bonferroni"`) is applied to correct for multiple testing by default. However user can select the best correction method for the analysis. The adjustment methods include the Benjamini-Hochberg false discovery rate method (`"fdr"`), Holm correction (`"holm"`), Hochberg correction (`"hochberg"`), Hommel (`"hommel"`) and Benjamini Yekutieli (`"BY"`).

```
comorbidityDirection <- directionality( input           = comorFemale,
                                        databasePth     = databasePth,
                                        sex             = "Female",
                                        ageRange        = c(0,100),
                                        days            = 0,
                                        minPairs        = 1,
                                        dataSep         = "-",
                                        correctionMethod = "bonferroni")

## Checking the input objects

summary(as.factor(comorbidityDirection$result))

## No directionality
##               3716
```

As a result a `data.frame` is obtained. This `data.frame` contains 7 columns, with the comorbidity disorders and the directionality results in numeric and character format.

```
##          disAcode  disBcode  AtoB  BtoA  test  correctedPvalue            result
## 328       F31.62     D37.4     2     1    1.0                1 No directionality
## 1017      F31.72       Z11     1     1    1.0                1 No directionality
## 1521       F32.4    C40.81     1     2    1.0                1 No directionality
## 913        F32.3     C40.3     2     0    0.5                1 No directionality
## 1191      F31.11    E10.52     2     1    1.0                1 No directionality
## 490        F32.1    D31.01     1     1    1.0                1 No directionality
```

These results can be visualized in a heatmap by applying the `heatmapDirection` function. As input this function requires the `data.frame` obtained after applying the `directionality` function. The `interactive` argument is also available. By default it is set to `FALSE`.

In the previous example none of the comorbidities show a preferred directionality. For this reason and to illustrate the `heatmapDirection` output we will apply the function to a `test` object containing all the possible results.

```
heatmapDirection(test,
                 fromAtoBColor    = "darkgreen",
                 fromBtoAColor    = "orange",
                 noDirectionColor = "grey",
                 interactive = FALSE)
                          ## Checking the input object
```


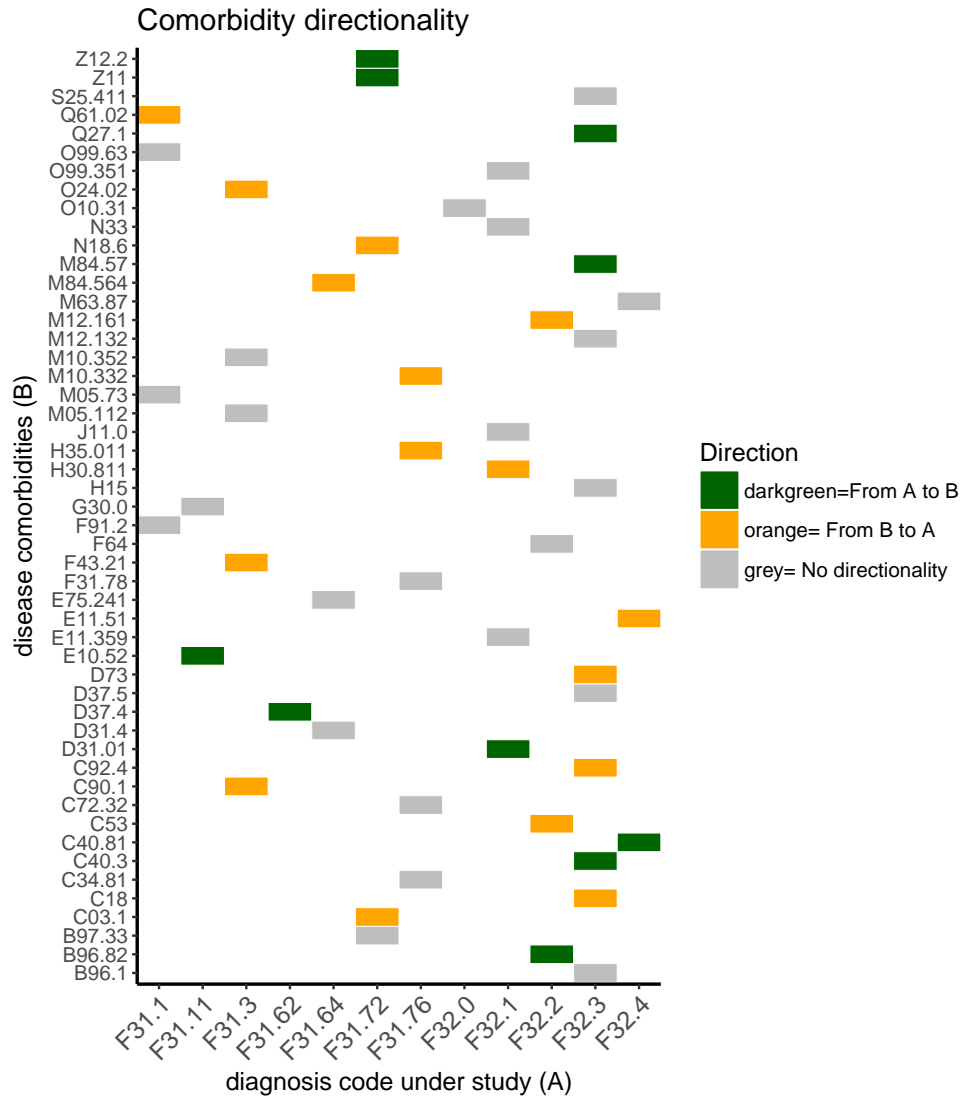Figure 11: Comorbidities directionality heatmap

As a result, a heatmap is obtained (Figure 11). Note that three possible results can be shown in the heatmap in three different colors:

- From A to B (by default: green).

- From B to A (by default: yellow).

- No directionality (by default: grey).

Note that the color of the heatmap can be changed adding the next arguments to the `heatmaDirection` function:

- **fromAtoBColor**: By default **"darkgreen"**. It determines the heatmap color when the preferred direction is from A to B.

- **fromBtoAColor**: By default **"orange"**. It determines the heatmap color when the preferred direction is from B to A.

- **noDirectionColor**: By default **"grey"**. It determines the heatmap color when there is no preferred direction. ' By default it is set to "grey"

3 Molecular Comorbidity

3.1 Requirements

The `cuiDiseaseList` file is required in order to perform the molecular comorbidity analysis with `comoRbidity` package. Alternatively, the user can provide as input a file containing gene disease associations. An example of the data is shown below.

3.1.1 Index Disease Codes

The `cuiDiseaseList` file must contain at least two predefined columns named as stated below:

- **identifier_id**: the UMLS identifier of the disease/s of interest.

- **name**: this column is not compulsory for performing the comorbidity analysis. It must be included if the user want to collapse the index disease codes in a higher category. The example below shows all index diseases collapsed in 7 different categories.

If the cuiDiseaseList file does not contain the required columns the next message will appear:

```
## Check the input file structure.  Remember that this
## + file must contain at least two columns with the column
## + names as follows:
##  -> identifier
##  -> name
```

```
filePth <- system.file("extdata", package="comoRbidity")
cuiDiseaseList <- read.delim( paste0(filePth, "/cuiDiseaseList.txt"), header = TRUE, sep = "\t" )
head(cuiDiseaseList)

##   identifier                           name
## 1   C0001969            Alcohol use disorders
## 2   C0001973            Alcohol use disorders
## 3   C0005586 Bipolar disorders and related disorders
## 4   C0006870            Cannabis use disorders
## 5   C0011570              Depressive disorders
## 6   C0011581              Depressive disorders
```

> While the aforementioned columns are required, the files may contain other additional information. The extra information will no be used by the `comoRbidity` R package for the molecular comorbidity analysis.

3.1.2 External file containing gene disease association

The alternative input file must contain at least two columns with the column names as follows:

- gene

- diseaseName

```
filePth <- system.file("extdata", package="comoRbidity")
genediseaseData <- read.delim( file    = paste0(filePth, "/genediseaseTable.txt"),
                               header = TRUE,
                               sep    = "\t" )
head(genediseaseData)
```

```
##   geneId   gene  disease          diseaseName
## 1    217  ALDH2 C0001969 Alcohol use disorders
## 2    125  ADH1B C0001969 Alcohol use disorders
## 3  10327 AKR1A1 C0001969 Alcohol use disorders
## 4    126  ADH1C C0001969 Alcohol use disorders
## 5   2908   NR3C1 C0001969 Alcohol use disorders
## 6   4524   MTHFR C0001969 Alcohol use disorders
```

> **Note that if you use this format data for the molecular comorbidity analysis, the overview of the gene-disease data performed applying the `summaryDiseases` will not be possible**

3.2   molecular comoRbidity objects

3.2.1   molecularComorbidity object

`molecularComorbidity` object is obtained when `querymolecular` function is applied. This object is used as input for other functions in the package that enable the user to have an overview about their index diseases as well as the genes associated with them. `molecularComorbidity` object is used as input in the function that performs the molecular comorbidity analysis (`comorbidityAnalysisMolecular`).

In summary `molecularComorbidity` object is the input for the functions:

- `summaryDiseases`

- `comorbidityAnalysisMolecular`

The `molecularComorbidity` object contains the query information as well as the summary of the results. It shows:

- The type of search that has been done (`Search`).

- If the `identifier` column is used for the comorbidity study (`Aggregate the disease codes: FALSE`), or if they are collapsed into a higher category and the `name` column is used for the comorbidity study (`Aggregate the disease codes:  TRUE`).

- `N. Input Diseases`: number of diseases that the user gives as input.

- `N. Index Diseases Present`: number of diseases that present associated genes in DisGeNET.

- `N. Genes`: number of genes associated to the diseases.

```
class(mc)

## [1] "molecularComorbidity"
## attr(,"package")
## [1] "comoRbidity"

mc

## Object of class 'molecularComorbidity'
##   . Search:                       list
##   . Aggregate the disease codes: TRUE
##   . Number of Input Diseases:         7
##   . Number of Index Diseases Present:   22
##   . Number of Genes   :               1759
```

All the `comoRbidity` R package objects come with a function called `extract`. The `extract` function allows the user to retrieve data stored in the object. The `extract` function

33

returns a formatted `data.frame` with the complete set of information obtained from the required data.

```
head(extract(mc))

##    geneId geneSymbol diseaseId          diseaseName
## 1    217      ALDH2  C0001969 Alcoholic Intoxication
## 2    125      ADH1B  C0001969 Alcoholic Intoxication
## 3  10327      AKR1A1 C0001969 Alcoholic Intoxication
## 4    126      ADH1C  C0001969 Alcoholic Intoxication
## 5   2908      NR3C1  C0001969 Alcoholic Intoxication
## 6   4524      MTHFR  C0001969 Alcoholic Intoxication
```

3.2.2   molecularcAnalysis object

`molecularcAnalysis` object is obtained when `comorbidityAnalysisMolecular` or `dataframe2molecularcomorbidit`
functions are applied. This object is used as input for other functions in the package that enable the
user to visualize the results in different graphical ways.

`molecularcAnalysis` object is the input for the following functions:

  • `network`

  • `heatmapPlot`

`molecularcAnalysis` object contains the results of the comorbidity analysis and other relevant
information for the user. `molecularcAnalysis` object shows the gene overlap interval (minimum
and maximum value of the gene overlap for the disease comorbidities, Overlap Min and Overlap Max)
as well as the Jaccard Index interval (`Jaccard Min` and `Jaccard Max`). Other data such as if the
p-value has been estimated or not is also shown (`P-value`). Finally, the number of comorbidities
found based on the genes shared between diseases are also shown (`Number of comorbidities`).

```
class(mcAnalysis)

## [1] "molecularcAnalysis"
## attr(,"package")
## [1] "comoRbidity"

mcAnalysis

## Object of class 'molecularcAnalysis'
##   . Minimum number of genes overlaped: 1
##   . Maximum number of genes overlaped: 23
##   . Jaccard Minimum value:            0.003
##   . Jaccard Maximum value:            0.142
##   . P-value      :                    TRUE
##   . Number of comorbidities:          45
```

All the `comoRbidity` **R package objects come with a function called** `extract`. **The** `extract`
function allows the user to retrieve data stored in the object. The `extract` **function
returns a formatted** `data.frame` **with the complete set of information obtained from the
required data.**

```
head(extract(mcAnalysis))

##                                 V1                          V2 geneV1
## 59           Alcoholic Intoxication Substance Withdrawal Syndrome      1
```

34

```
## 61 Alcoholic Intoxication, Chronic              Bipolar Disorder     41
## 64 Alcoholic Intoxication, Chronic     Cocaine-Related Disorders     41
## 65 Alcoholic Intoxication, Chronic             Depressive disorder    41
## 67 Alcoholic Intoxication, Chronic     Major Depressive Disorder     41
## 70 Alcoholic Intoxication, Chronic                 Mood Disorders     41
##    geneV2 overlap jaccard  pval
## 59     53       1   0.019 0.000
## 61     72       5   0.046 0.000
## 64    108      13   0.096 0.000
## 65     37       2   0.026 0.011
## 67     41       2   0.025 0.000
## 70      3       1   0.023 0.013
```

3.3 Data extraction

3.3.1 Using as input a list of diseases

The first step to perform the molecular comorbidity analysis is to extract the gene-disease association data for the index diseases. This information is extracted from DisGeNET database (http://disgenet.org).

`querymolecular` function allows the user to extract the genes associated to the index diseases, based on DisGeNET data, and store it in a `molecularComorbidity` class object. As input the function requires:

- `filePth`: determines the file name with the complete path where the file with disorders of interest is located.

- `unify`: the default value is set to `FALSE`. If the argument is set to `TRUE`, the `name` colum from the cui disease file will be selected for performing the comorbidity analysis.

- `database`: the default value is set to `'CURATED'`. User can select any of the databases available in DisGeNET (Table 2).

- `score`: by default it is set to `(">", 0)`. It means that all the data available in DisGeNET will be used for the comorbidity analysis. For detailed information about DisGeNET score: *http://disgenet.org/web/DisGeNET/menu/dbinfoscore*.

Table 2: Source databases included in DisGeNET

Name	Description
CTD_{human}	The Comparative Toxicogenomics Database, human data
UNIPROT	The Universal Protein Resource
CLINVAR	ClinVar, public archive of relationships among sequence variation and human phenotype
ORPHANET	The portal for rare diseases and orphan drugs
GWASCAT	The NHGRI-EBI GWAS Catalog
PSYGENET	Psychiatric disorders Gene association NETwork
HPO	Human Phenotype Ontology
GAD	The Genetic Association Database
CTD_{mouse}	The Comparative Toxicogenomics Database, *Mus musculus* data
CTD_{rat}	The Comparative Toxicogenomics Database, *Rattus norvergicus* data
MGD	The Mouse Genome Database
RGD	The Rat Genome Database
LHGDN	Literature-derived human gene-disease network generated by text mining Entrez's GeneRIFs
BEFREE	Data from text mining medline abstracts using the BeFree System
CURATED	Human curated sources:(CTD_{human}), ClinVar, GWASCAT, UniProt, Orphanet, PsyGeNET, and HPO
PREDICTED	All data from animal models: CTD_{rat}, RGD, CTD_{mouse}, MGD
ALL	All previous data sources

As a result, a `molecularComorbidity` object is obtained. This object contains all the gene-disease associations for the index diseases available in DisGeNET according to the user database and score selection.

```
diseaseCodes <- paste0(filePth, "/cuiDiseaseList.txt")
mc <- querymolecular ( filePth = diseaseCodes,
                       unify = FALSE,
                       database = "CURATED",
                       score = c(">", 0))

## Error:  Proxy Error

mc

## Object of class 'molecularComorbidity'
##  . Search:                         list
##  . Aggregate the disease codes: TRUE
##  . Number of Input Diseases:          7
##  . Number of Index Diseases Present:  22
##  . Number of Genes   :                1759
```

3.3.2 Using as input a list of gene-disease associations

Alternatively the user can provide her/his own gene-disease association file. In order to transform that input file in a `molecularComorbidity` class object the `dataframe2molecularcomorbidity` function. The input file must contain at least two predefined columns named as stated below:

- **gene**: a gene identifier, that can be numeric, alphanumeric or a list of characters.

- **diseaseName**: a disease identifier, that can be numeric, alphanumeric or a list of characters.

  If the input file does not contain the required columns the next message will appear:

  ```
  ## Check the input file structure.  Remember that this
  ##              file must contain at least two columns with the column
  ##                 names as follows:
  ##  -> gene
  ##  -> diseaseName
  ```

  ```
  geneDiseaseFile <- read.delim(system.file("extdata", "genediseaseTable.txt", package="comoRbi
  head( geneDiseaseFile )

  ##   geneId   gene  disease                 diseaseName
  ## 1    217   ALDH2 C0001969 Alcohol use disorders
  ## 2    125   ADH1B C0001969 Alcohol use disorders
  ## 3  10327 AKR1A1 C0001969 Alcohol use disorders
  ## 4    126   ADH1C C0001969 Alcohol use disorders
  ## 5   2908   NR3C1 C0001969 Alcohol use disorders
  ## 6   4524   MTHFR C0001969 Alcohol use disorders
  ```

  As input the function only requires one argument, `filePth`, that determines the file name with the complete path where the file with gene and diseases of interest is located. There is not restriction about the gene and/or disease identifier class. However, the user should take into account that the `summaryDiseases` function cannot be applied. Nevertheless, the molecular comorbidity analysis can be applied.

  ```
  genediseaseFile <- paste0(filePth, "/genediseaseTable.txt")
  manualmc <- dataframe2molecularcomorbidity( filePth = genediseaseFile)
  manualmc

  ## Object of class 'molecularComorbidity'
  ```

```
##   . Search:                          list
##   . Aggregate the disease codes: FALSE
##   . Number of Input Diseases:        7
##   . Number of Index Diseases Present:  7
##   . Number of Genes    :             1759
```

3.4   Overview of the gene-disease data

> **Note that if you apply this function to a `molecularComorbidity` object obtained applying the `dataframe2molecularcomorbidity` function, the next error message will appear: `Sorry, this function is only available for object obtained after applying the queryMolecular function.`**

3.4.1   Genes summary

The `comoRbidity` R package allows the user to analyze and characterize the genes associated to the index diseases. In order to have a general overview about the genes, the `summaryDiseases` function can be applied, by setting the `type` argument to `gene_barplot`.

As input, the `summaryDiseases` function requires:

- `input`: a `molecularComorbidity` object, obtained after applying the `querymolecular` function.
- `type`: 'gene_barplot' is selected to perform the gene analysis.
- `database`: the DisGeNET database from where the gene information will be retrieved (by default: 'CURATED').
- `interactive`: Determines if the output barplot is interactive or not. By default the `interactive` argument is set to `FALSE`. The value of the argument can be changed to `TRUE`, as a result an interactive barplot generated with Shiny will be obtained.

As a result, a barplot showing the number of diseases associated to each gene will be displayed. Moreover, some gene attributes like the Disease Pleiotropy Index (DPI, by default the black dot) and Disease Specificity Index (DSI, by default the red dot) are displayed. Note that the default colors of the DPI and DSI can be changed using the arguments dsiColor and dpiColor. These indexes provide an indication on how specific is a gene with respect to diseases. For example, a DSI close to one means that a gene is annotated to a small number of diseases and therefore is more disease-specific, while a low DSI value (e.g. 0.25) indicates that the gene is associated to a large number of diseases and therefore is less disease-specific. For more information about these indexes:
http://disgenet.org/web/DisGeNET/menu/dbinfospecificity.

```
summaryDiseases( input   = mc,
                 type    = "gene_barplot",
                 database = "CURATED",
                 interactive = FALSE
        )


## Checking the input object

## The top 10 genes with more diseases associated are shown
```
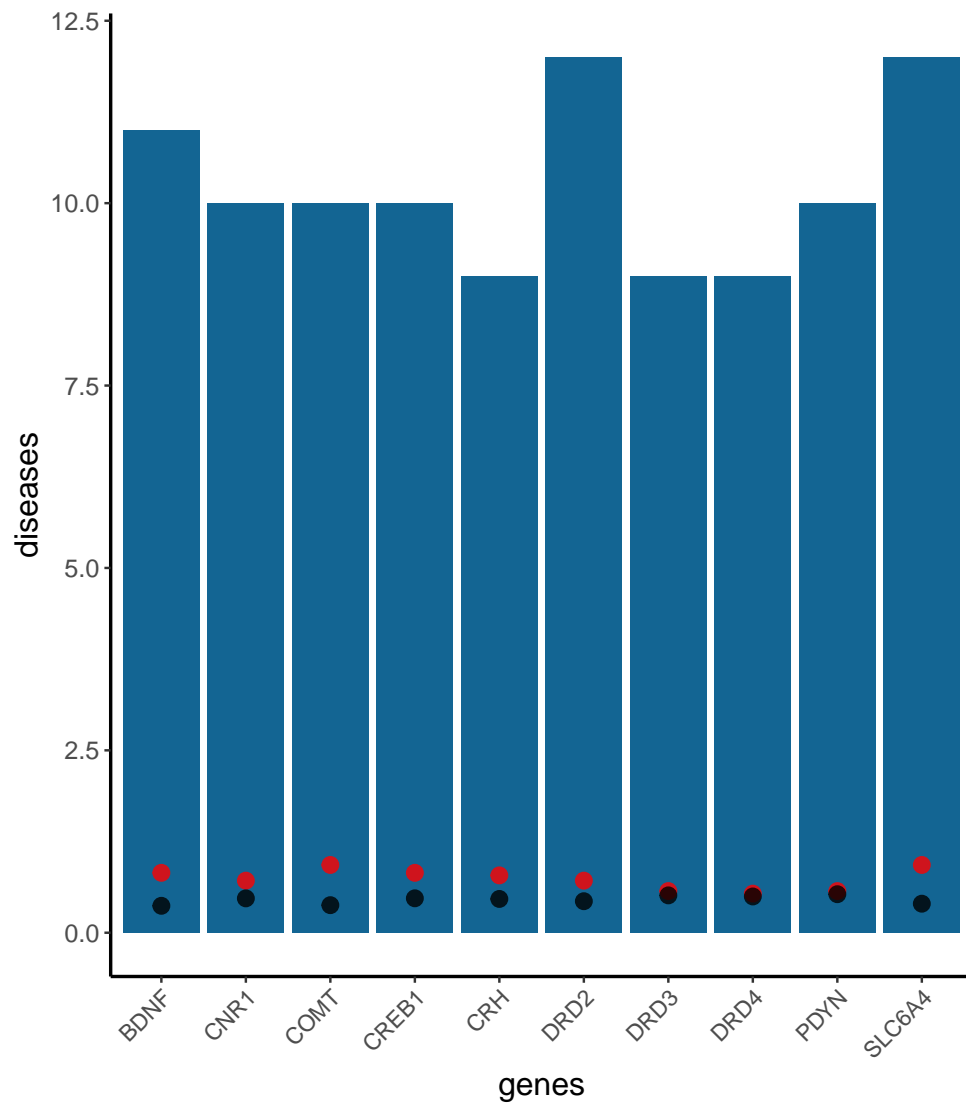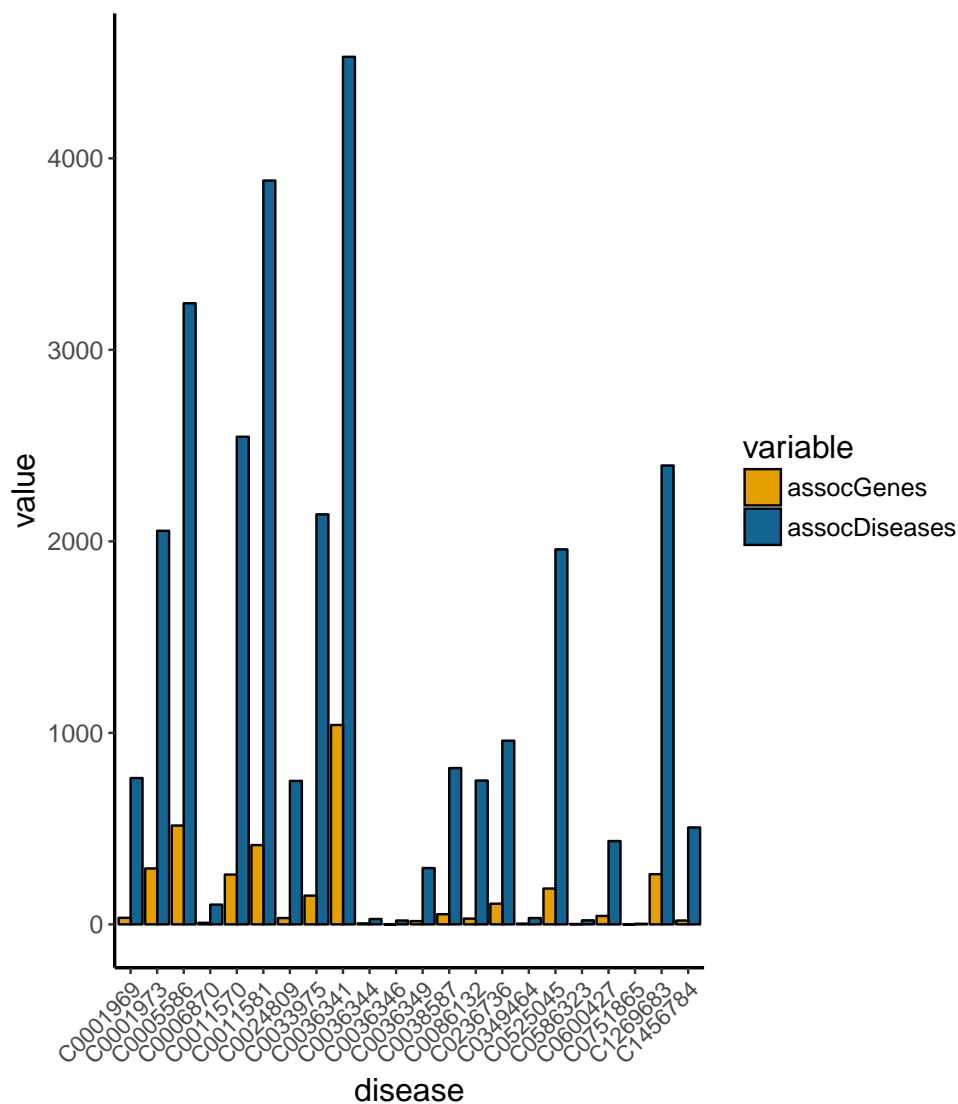
3.4.2 Diseases summary

The `comoRbidity` R package allows the user to analyze and characterize the diseases. In order to have a general overview about the number of genes associated to each one of the index diseases as well as the number of disorders that share some genes with them, the `summaryDiseases` function can be applied, setting the `type` argument to `dis_barplot`.

```
summaryDiseases( input    = mc,
                 type     = "dis_barplot",
                 database = "CURATED")

## Checking the input object
```

Note that the bar colors can be changed adding the next arguments to the `summaryDiseases` function:

- `assocGeneColor`: By default `"E69F00"`. It defines the bar color that represents the number of associated genes
- `assocDiseaseColor`: By default `"136593"`. It defines the bar color that represents the number of associated diseases.

3.5 Molecular comorbidity analysis

Having a general overview about the index diseases and the genes associated with them, the next step is to perform the molecular comorbidity analysis.

The user can estimate the molecular comorbidities by applying the `comorbidityAnalysisMolecular` function to the `molecularComorbidity` object previously generated with the `queryMolecular` function.

3.5.1 Molecular Comorbidity Measurements

The comorbidity is estimated from the molecular perspective taking into account the number of genes shared between the diseases according to DisGeNET data. The `comoRbidity` R package estimates the Jaccard index and optionally a p-value associated to it.

Jaccard Index The Jaccard Index, also known as the Jaccard similarity coefficient, is a statistic measurement used for comparing the similarity of two sets, and is defined as the size of the intersection divided by the size of the union of the sample sets:

$$J(A, B) = \frac{\mid A \cap B \mid}{\mid A \cup B \mid}$$

`comorbidityAnalysisMolecular` function computes the Jaccard Index as an estimation of the similarity of two diseases based on the number of genes shared between the diseases according to DisGeNET data.

P-Value To determine if the association between two diseases as estimated by the Jaccard Index is statistically significant, a bootstrap procedure can be applied. `comorbidityAnalysisMolecular` function has two optional arguments, `pValue` and `nboot`. By applying this function, random gene sets of size n and p (being n, p the number of genes associated to disease 1 and 2, respectively) are sampled from a population of human disease genes obtained from DisGeNET CURATED. These random gene sets (n and p) are then used to compute the Jaccard Index for diseases 1 and 2. This procedure is repeated `nboot` times. Then the number of times that it has been obtained a Jaccard Index for the random gene sets larger than the observed value of the Jaccard Index is computed and represents its p-value.

The `comorbidityAnalysisMolecular` function allows the user to perform the molecular comorbidity analysis and store it in a `molecularcAnalysis` object. As input this function requires:

- `input`: a `molecularComorbidity` object obtained after applying the `querymolecular` function.
- `pValue`: determines if the p-value is estimated or not. By default it is set to `'FALSE'`. The `pValue` argument can be set to `'TRUE'` in order to estimate the P-value associated to each Jaccard Index.
- `nboot`: determines the number of random times that the Jaccard Index is computed using random sets. By default it is set to `100`. The value of the argument can be changed to any other numeric variable.

```
mcAnalysis <- comorbidityAnalysisMolecular(input  = mc,
                                           pValue = TRUE,
                                           nboot  = 1000
                                           )

## Checking the input object
## Estimating the overlap and jaccard
## Estimating the p-value
## A total of 8948 genes obtained from DisGeNET CURATED database are
##                 being used for the bootstrap process
## Pvalue estimation for each comorbidity
```

As a result, a `molecularcAnalysis` object is obtained. The `molecularcAnalysis` object contains the results of the molecular comorbidity analysis and other relevant information for the user. `molecularcAnalysis` object shows the gene overlap interval for the disease comorbidities (`Overlap Min` and `Overlap Max`) as well as the Jaccard Index interval for the disease comorbidities

(`Jaccard Min` and `Jaccard Max`). Other data such as if the p-value has been estimated or not is shown (`P-value`). Finally the number of comorbidities that have gene overlap are also shown (`Number of comorbidities`).

```
## Object of class 'molecularcAnalysis'
##  . Minimum number of genes overlaped: 1
##  . Maximum number of genes overlaped: 302
##  . Jaccard Minimum value:           0.001
##  . Jaccard Maximum value:           0.601
##  . P-value     :                   TRUE
##  . Number of comorbidities:        130
##                                 V1                          V2 geneV1
## 23 Alcohol Withdrawal Seizures Alcoholic Intoxication, Chronic      2
## 42        Alcoholic Intoxication Alcoholic Intoxication, Chronic     34
## 43        Alcoholic Intoxication              Bipolar Disorder      34
## 45        Alcoholic Intoxication       Cocaine-Related Disorders     34
## 46        Alcoholic Intoxication            Cocaine Dependence      34
## 47        Alcoholic Intoxication           Depressive disorder      34
##    geneV2 overlap jaccard pval
## 23    292       2   0.007    0
## 42    292      17   0.055    0
## 43    516      19   0.036    0
## 45    108       5   0.036    0
## 46     44       3   0.040    0
## 47    414      16   0.037    0
```

3.5.2   Molecular comorbidity visualization

In order to visualize the molecular comorbidity analysis results, the `comoRbidity` R package provides two different options:

- Network: obtained by applying the `network` function.
- Heatmap: obtained by applying the `heatmapPlot` function.

Comorbidity Network
`network` function allows the user to visualize the data contained in the `molecularcAnalysis` object obtained after applying the `comorbidityAnalysisMolecular` function.

As input the `network` function requires:

- `input`: a `molecularcAnalysis` object obtained after applying the `comorbidityAnalysisMolecular` function.
- `layout`: by deafult `"layout.circle"`. It can be set to other of the possible igraph layouts.
- `selectValue`: By default `"jaccard"` variable will be selected. It can be set to to the p-value variable ('pval').
- `cutOff`: By default '0.05'. The value of the argument can be changed to any other numeric variable, according to the range of the selected value.
- `npairs`: By default '0'. The value of the argument can be changed to any other numeric variable to show in the network only those comorbidities in which the gene overlap is equal or greather than the `npairs` value.
- `prop`: Determines the node size proportionality. By default it is set to 1. The value of the argument can be changed to any other numeric variable.
- `title`: Determines the title of the network figure. By default 'Comorbidity network'.

- interactive Determines if the output network is interactive or not. By default the interactive argument is set to FALSE. The value of the argument can be changed to TRUE, as a result an interactive network will be obtained.

```
mcNetwork <- network( input = mcAnalysis ,
                      layout = "layout.circle",
                      selectValue = "jaccard",
                      title = "Molecular comorbidity network",
                      cutOff = 0.05,
                      prop   = 0.2,
                      diseaseColor = "olivedrab1",
                      interactive = FALSE,
)
                      ## Checking the input object
```
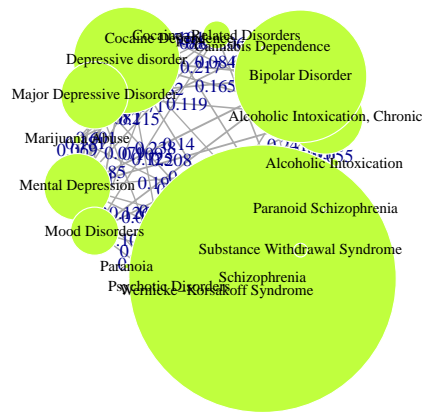
Molecular comorbidity network

Figure 12: Molecular comorbidity network

As a result, a network is obtained (Figure 12). Note that the color of the nodes can be changed by adding the diseaseColor argument to the network function. The diseaseColor argument is set to "pink" color by default.

Comorbidity heatmap The comoRbidity package also allows to visualize molecularcAnalysis object in a heatmap (Figure 13). The required input is the same as for the network function.

```
mcHeatmap <- heatmapPlot( input = mcAnalysis ,
           selectValue = "jaccard",
           cutOff = 0.05,
           interactive = FALSE,
           lowColor = "#0000FF",
           highColor = "yellow" )
                    ## Checking the input object

mcHeatmap
```
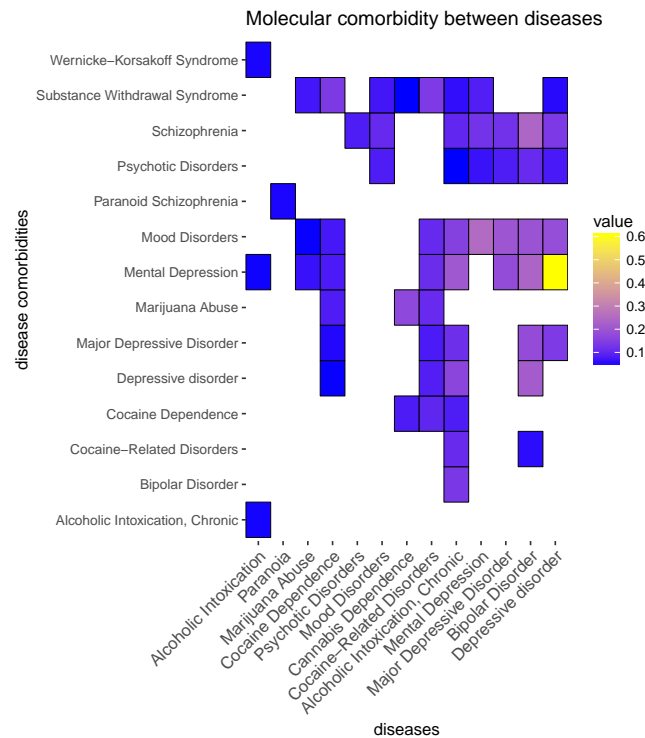

Figure 13: Molecular Comorbidity heatmap

Note that the color of the heatmap can be changed adding the next arguments to the `heatmaPlot` function:

- `lowColor`: By default `"0000FF"`. It defines the heatmap color for the lowest value.
- `highColor`: By default `"yellow"`. It defines the heatmap color for the highest value.

4   Warnings

All the functions in the `comoRbidity` R package first check that the input object belongs to the correct class. If the input is not correct the following message will be shown:

```
## Check the input object.  Remember that this
##                 object must be obtained after applying the query
##                 function to your input file.  The input object class must
##                 be:"cAnalysis" or "molecularcAnalysis"
```

5 Bibliography

References

[1] Capobianco E, Lio' P **Comorbidity: a multidimensional approach.** Trends in molecular medicine 2013 vol: 19 (9) pp: 515-21

[2] Hidalgo C, Blumm N, Barabasi A, Christakis N **A Dynamic Network Approach for the Study of Human Phenotypes** PLoS Computational Biology 2009 vol: 5 (4) pp: e1000353

[3] Vogeli C, Shields A, Lee T, Gibson T, Marder W, Weiss K, Blumenthal D. **Multiple chronic conditions: prevalence, health consequences, and implications for quality, care management, and costs.** Journal of general internal medicine 2007: 39139

[4] Hwang W, W. Weller, H. Ireys, Anderson G. **Outofpocket medical spending for care of chronic conditions** Health Affairs 2001, Vol. 20, No. 6

[5] Park J, Lee D, Christakis NA, Barabasi A. **The impact of cellular networks on disease comorbidity** Mol Syst Biol 2009;5:262

[6] Park S, Yang JS, Shin YE, Park J, Jang SK, Kim S. **Protein localization as a principal feature of the etiology and comorbidity of genetic disease.** Mol Syst Biol 2011 May 24;7:494.

[7] Lee D, Park J, Kay K, Christakis N, Oltvai Z, Barabasi A. **The implications of human metabolic network topology for disease comorbidity.** Proceedings of the National Academy of Sciences 2008;105(29):98809885.

[8] Le D, Kwon Y. **The effect of feedback loops on disease comorbidity in human signaling networks.** Bioinformatics 2011 Apr;27(8):11131120.

[9] Barabasi, A. L. **Network medicinefrom obesity to the "diseasome** N Engl J Med 2007, 357:404407

[10] Barabasi, A. L., N. Gulbahce, and J. Loscalzo. **Network medicine: a networkbased approach to human diseasome** Nat Rev Genet 2011, 12:56687

[11] Jakovljevic, Miro and Ostojic, Ljerka. **Comorbidity and multimorbidity in medicine today:challenges and opportunities for bringing separated branches of medicine closer to each other.me** Mostariensia, 2013. 18287

[12] Pinero J, Queralt-Rosinach N, Bravo A, Deu-Pons J, Bauer-Mehren A et. al. **DisGeNET: a discovery platform for the dynamical exploration of human diseases and their genes.** Database, 2015 vol: 2015 pp: bav028

[13] Yoav Benjaminia; Dan Draib; Greg Elmerc; Neri Kafkafid; Ilan Golanib **Controlling the false discovery rate in behavior genetics research** Behavioural Brain Research 2001 doi:10.1016/S0166-4328(01)00297-2

[14] Roque, Francisco S and Jensen, Peter B and Schmock, Henriette and Dalgaard, Marlene and Andreatta, Massimo and Hansen, Thomas and Søeby, Karen and Bredkjær, Søren and Juul, Anders and Werge, Thomas and others **Using Electronic Patient Records to Discover Disease Correlations and Stratify Patient Cohorts** PLOS Computational Biology 2011 doi:10.1371/journal.pcbi.1002141

[15] Klimek, Peter and Kautzky-Willer, Alexandra and Chmiel, Anna and Schiller-Frühwirth, Irmgard and Thurner, Stefan **Quantification of diabetes comorbidity risks across life using nation-wide big claims data.** PLoS Comput. Biol. 2015 doi: 0.1371/journal.pcbi.1004125

[16] Gomez-Cabrero, David and Menche, Jörg and Vargas, Claudia and Cano, Isaac and Maier, Dieter and Barabási, Albert-László and Tegnér, Jesper and Roca, Josep **From comorbidities of chronic obstructive pulmonary disease to identification of shared molecular mechanisms by data integration** BMC bioinformatics 2016