Introduction to Data Mining with R

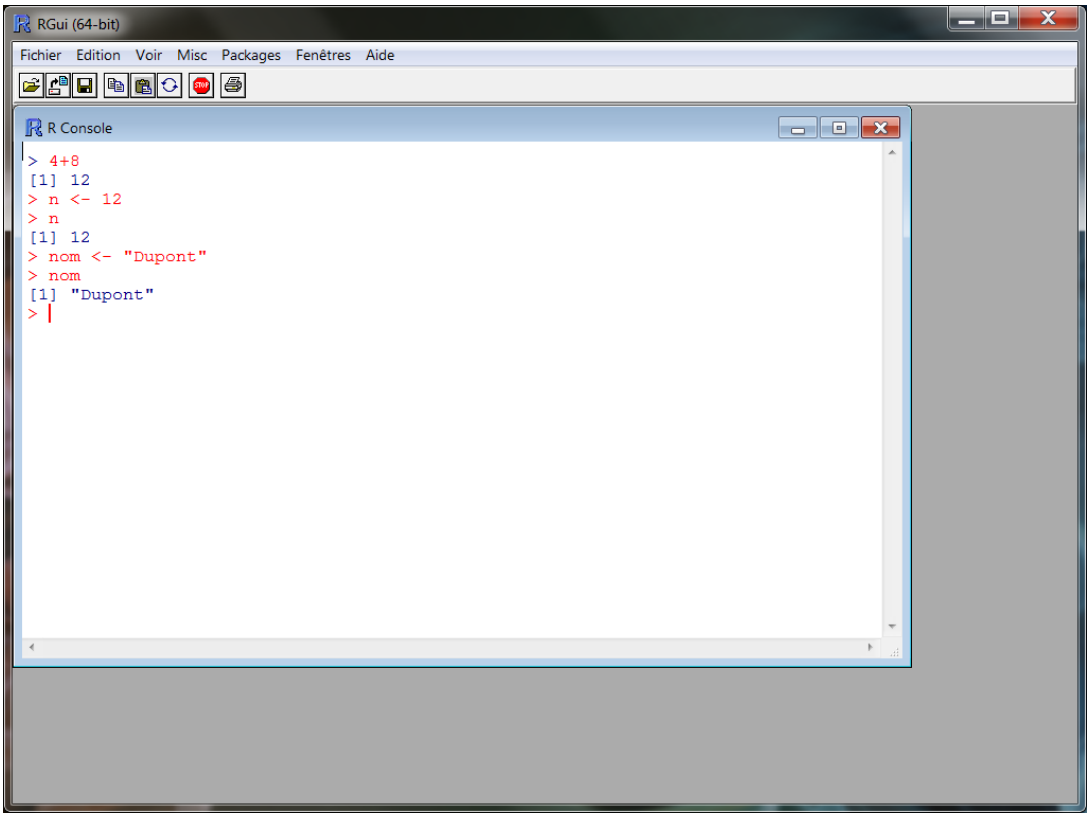
1. The R Software

R is a free interactive scientific computing software that provides a large collection of statistical, data analysis and graphics tools.

The website http://www.r-project.org/ provides a comprehensive description of the R software as well as the links for its download, access to the various libraries and help documents. Compiled versions of R are available for Linux, Windows and Mac OS-X.

R is an interactive language, so always waiting for an instruction indicated by the ">" prompt in the *R Console* command window.

R Console GUI Window



R has a very comprehensive online help that is very useful to define the arguments and options of R function calls. This online help provides extensive descriptions and examples of the use of functions.

All elementary operations can be executed on the command line at the R prompt. The instructions are executed by pressing the enter key on the keyboard.

Notes: The `$`, `[`, `[[`, `:`, `?` and `<-` characters are operators for R, and R is case-sensitive as it differentiates between uppercase and lowercase characters.

☛ Download and install R from http://www.r-project.org/.

1.1. First Manipulations with R

☛ Run the calculation instructions below in the *R Console* window:

```
> 4+8
[1] 12
> (6+5*2)/2
[1] 8
> 2^2                          // power calculation
[1] 4
> sqrt(9)                      // squared root calculation
[1] 3
```

Note: Using the arrow keys you can retrieve the last instructions executed to modify and re-execute

them.

☛ An object (variable) can be created using the "assign" operator which is written with the "<-" symbol:

```
> n <- 100
```

This object is stored in RAM and can be modified by assigning it another value. The content of the object is displayed by typing its name (if this object does not require arguments).

☛ Execute the instructions for manipulating objects in memory below.

```
> n
[1] 100
> n <- 100/2+5
> n
[1] 55
```

The names of the objects stored in RAM and/or their content can be displayed by the `ls()` function.

☛ Execute the instructions below

```
> n <- 12
> m <- 8
> s <- 12+8
> nom <- "Dupont"
> ls()
[1] "n" "m" "s" "nom"
> ls.str()
m : num 8                          // num : numerical variable
n : num 12
nom : chr "Dupont"                 // chr : string type variable
s : num 20
```

1.2. Online Help

R has a very comprehensive online help.

By typing `? instruction` (ex: `? sqrt`) or `help("instruction")`, the description of this instruction, its arguments, the type of object it returns and some examples of uses are shown.

The command `example(instruction)` shows advanced examples of using the instruction.

The command:

```
> help.start()
```

launches an Internet browser and accesses directly to the help stored locally with the R installation: http://127.0.0.1:11490/doc/html/index.html.

1.3. Vectors

The basic object in R is the *vector*. Even when you create a variable containing a single value (ex: `n <- 12.5`), a vector containing a single element is created.

In R, all objects have a mode and a length:

- The mode determines the type of data stored in the vector. A vector contains a set of elements of the same atomic type that can be: character, logical (`T` / `F` or `TRUE` / `FALSE`), numeric or complex. It can be displayed using the `mode()` function.
- The length determines the current number of elements in the vector. It can be displayed using the `length()` function.

☛ Create a vector `v` using the concatenation function `c()`, that combines its arguments to form a vector, by the following command:

```
> v <- c(4, 7, 23.5, 76.2, 80)
> v
[1] 4.0 7.0 23.5 76.2 80.0
> length(v)
[1] 5
> mode(v)
```

```
[1] "numeric"
```

All elements of a vector must be of the same type. If this is not the case, R will force the typing by coercion.

☛ Execute the commands below to illustrate this forcing:

```
> v <- c(4, 7, 23.5, 76.2, 80, "rrt")
> v
[1] "4" "7" "23.5" "76.2" "80" "rrt"
```

All elements have been converted to characters (the most permissive type here). Characters and strings are surrounded by symbols `"` or `'` indifferently.

Vectors can contain the particular value `NA` which represents a missing value.

☛ Execute the commands that make use of the `NA` value below:

```
> u <- c(4, 6, NA, 2)
> u
[1] 4 6 NA 2
> k <- c(T, F, NA, TRUE)
> k
[1] TRUE FALSE NA TRUE
```

The main functions for manipulating vectors are listed in the table below.

Main Functions of Vector Manipulation

Function	Description
sum(x)	Sum of elements in x
prod(x)	Product of elements in x
max(x)	Greatest of elements in x
min(x)	Smallest of elements in x
range(x)	Value domain of elements in x
length(x)	Number of elements in x
mean(x)	Mean of elements in x
median(x)	Median of elements in x
var(x)	Variance of elements in x (computed with n-1)
sd(x)	Standard deviation of elements in x
quantile(x)	Thresholds defining the 4 quartiles of x
cov(x)	Variance-covariance matrix
cor(x)	Correlation matrix
cov(x,y)	Covariance between x and y
cor(x,y)	Correlation between x and y
sort(x)	Orders elements in x
table(x)	Membership table of x
table(x,y)	Contingency table of x and y

1.4. Data Frames

A *data frame* in R can store a heterogeneous data matrix such as for example a data file. The columns of a data frame, each corresponding to a vector, can be of different types.

☛ Execute the instructions below to create a data frame nammed `my.dataset` containing data of different types, with three attributes named `site`, `season`, and `pH`, and then display it:

```
> my.dataset <- data.frame(site=c('A','B','A','A','B'), season = c('Winter',
'Summer', 'Summer', 'Spring', 'Fall'),  pH = c(7.4,6.3,8.6,7.2,8.9))
> my.dataset
```

The content of a column can be displayed (referenced) by the statement *dframe_name*`$`*column_name* and the contents of a cell by the selection statement *dframe_name*`[`*row*`, `*column*`]`.

☛ Display the contents of the `pH` column and the content of the 2nd cell of the 3rd row:

```
> my.dataset$pH
[1] 7.4 6.3 8.6 7.2 8.9
> my.dataset[3,2]
[1] Summer
Levels: Fall Spring Summer Winter
```

Selections are made simply by adding conditions to the selector *dframe_name*`[`*row*`, `*column*`]`. For example:

```
> my.dataset[my.dataset$pH > 7, ]
  site season  pH
1    A Winter 7.4
3    A Summer 8.6
4    A Spring 7.2
5    B   Fall 8.9
> my.dataset[my.dataset$site == "A", "pH"]
[1] 7.4 8.6 7.2
> my.dataset[my.dataset$season == "Summer", c("site", "pH")]
  site  pH
2    B 6.3
3    A 8.6
```

☛ Display the values of `site` and `pH` attributes for lines whose season is different from `'Fall'`.

☛ Display the values of all attributes for all rows with `pH` less than or equal to `8.1`.

Note: The `attach(dframe_name)` command simplifies the writing of these selections as it allows to directly access the attributes by their name (ex: `season`), without having to use the name of the data frame as prefix (ex: `my.dataset$season`). For example:

```
> attach(my.dataset)
> my.dataset[site=='B', ]
  site season  pH
2    B Summer 6.3
5    B   Fall 8.9
> season
[1] Winter Summer Summer Spring Fall
Levels: Fall Spring Summer Winter
```

The `detach(dframe_name)` command deactivates this attachment. For example:

```
> detach(my.dataset)
> season
Error: Object "season" not found
```

2. The RStudio Integrated Development Environment

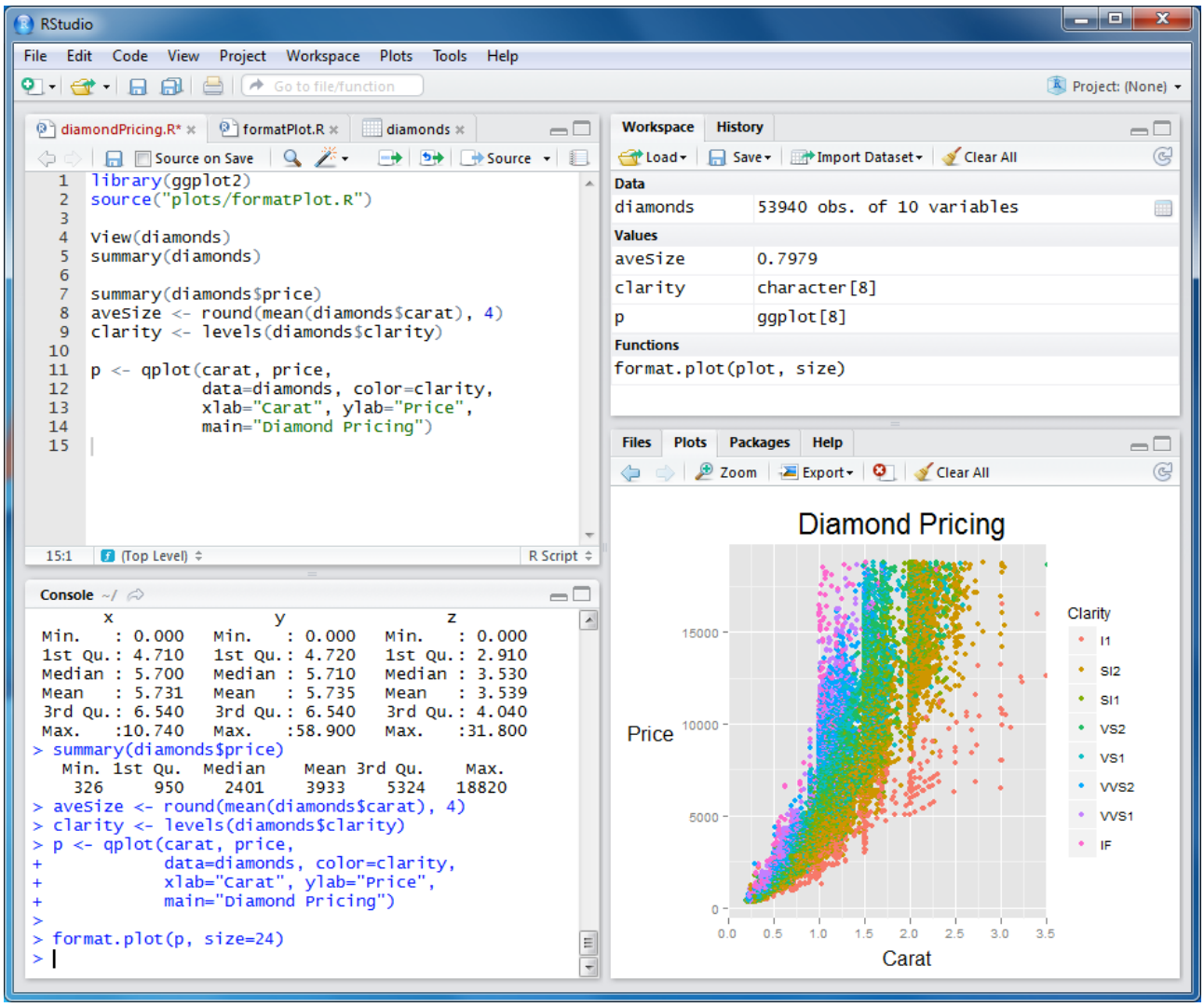
RStudio is a free open source IDE for R. It is available for Windows, Mac OS-X and Linux operating systems. Both compiled and on-the-go (without installation) versions of RStudio are available for Linux, Windows and Mac OS-X.

The window of RStudio is divided into 4 zones, as it is visible in the capture below:

- The console (bottom-left) to type and execute directly R commands.
- The code manager (top-left) that allows to display both codes (R scripts, etc.) and textual objects (data frames, matrices, etc.)
- The environment manager (top-right) provides information about the environment that is the objects stored in RAM (`Workspace`) and the list of commands executed during the session (`History`).
- The file manager (bottom-right) that allows to display graphics (`Plots`), to install and activate R li-

braries (`Packages`) and files (`Files`), and to display the help topics (`Help`).

RStudio Windows



RStudio is available for the Windows, Linux and Mac OS-X operating systems in two versions:

- Installer: Performs the installation among the operating system software,
- Zip/Tarball: Does not require installation, only to:
  - Unzip the .zip or .tar.gz archive file to your machine. This operation creates the directory structure containing all the files required by RStudio.
  - Start the execution of RStudio by running the executable binary file (ex: `rstudio.exe`) in the `/bin/` created directory.

☛ Access the web-page: https://www.rstudio.com/products/rstudio/download/ to download and install RStudio Desktop for your computer.

Note : The *History* tab of the environment manager, that displays the list of R commands executed during the current session, allows to copy-paste these commands on the *Console* prompt to re-execute them, or in a script file in the file manager zone.

2.1. Working Directory

The current directory, or *working directory*, is the directory in which R will read and write the files by default (i.e. unless another directory is explicitly specified).

To define the working directory for the current session, you can use:

- The RStudio interface (menu *Session* option *Set working directory*).
- The `setwd("path")` function.

☛ Define the working directory as the one where you will place all the files used during the session.

Note: When you quit R, the `.Rhistory` and `.Rdata` files, that contain respectively the list of executed

commands and the environment (data frames, vectors, etc.) of the session, will be saved in the working directory. To resume the session in the state in which you left it, you can load these files at the beginning of the next session

2.2. R Libraries and Packages

The possibilities offered by R can be extended through the *libraries* available on the CRAN (The Comprehensive R Archive Network) at: https://cran.r-project.org/. A specific version of a library (ex: 2.2.1) is called a *package*, and several packages of the same library can be installed (but not activated) at the same time in an R installation (to ensure compatibility between dependent libraries, which is however rarely required).

In order to install (download and install) packages in R, and to load (activate) packages for the current R session, you can use:

- The RStudio interface (menu *Tools* option *Install Packages*).
- The following functions in command-line:
  - `install.packages("library")` to download and install from CRAN
  - `library(library)` to activate it for the current session.

The list of libraries currently installed on the computer can be displayed by the command:

```
> library()
```

and in a detailed format (installation path, version number, etc.) by the command:

```
> installed.packages()
```

The `RSiteSearch()` function allows to search for some key words in R mailing lists, archives, manuals, and help pages. For example:

```
> RSiteSearch('association rules')
```

This function requires an Internet connection.

☛ Install the *ggplot2* graphics creation library and activate it in R.

☛ Display the list of installed libraries, where *ggplot2* should now appear.

3. Application Example

This application consists to download data in HTML format from a web-page, store it in a data frame, and process it in order to display comparative values on USA maps with R.

We will use the data from the following Wikipedia page on life expectancy in the USA: https://en.wikipedia.org/wiki/List_of_U.S._states_by_life_expectancy.

☛ Create an R script file using the menu *File* / *New File* / *R Script* of RStudio.

☛ Type the R code below in the new tab in the top-left zone of RStudio.

☛ Execute one by one the commands by positioning the cursor on the first line and then successively typing the CTRL-R keyboard shortcut to execute the current line and automatically position the cursor on the next line.

The steps of this application are the following:

This application requires the R packages named *rvest*, *ggplot2*, *dplyr*, *scales*, *maps*, *mapproj* and *plotly* that must be downloaded and installed, and then activated in the R session:

```
# Packages download and installation
install.packages("rvest")
install.packages("ggplot2")
install.packages("dplyr")
install.packages("scales")
install.packages("maps")
install.packages("mapproj")
install.packages("plotly")
# Packages activation
library(rvest)
library(ggplot2)
library(dplyr)
```

```
library(scales)
library(maps)
library(mapproj)
library(plotly)
```

The `le` object will contain the data that are retrieved from the Internet formatted in XML format and transformed into tables:

```
# Downloading data from Internet
le = read_html("https://en.wikipedia.org/wiki/List_of_U.S._states_by_life_expectancy")
print(le)
le = le %>% html_nodes("table") %>% .[[2]] %>% html_table(fill=T)
print(le)
```

The data is then filtered to generate the `le_african` and `le_caucasian` variables that will store the data for display:

```
# Selecting column with relevant data
le = le[c(1:8)]
# Naming column in the table
names(le) = le[3,]
# Deleting useless columns
le = le[-c(1:3), ]
le = le[, -c(5:7)]
# Renaming 4th and 5th columns
names(le)[c(4,5)] = c("le_african", "le_caucasian")
# Converting variables to numeric format
le = le %>% mutate(le_african = as.numeric(le_african), le_caucasian = as.nu-
meric(le_caucasian))
```

Calculating the differences in the life expectancy of the Caucasian and African-American ethnic groups for each state of the USA:

```
# Calculating differences
le = le %>% mutate(le_diff = (le_caucasian - le_african))
# Loading USA state data
states = map_data("state")
# Creating a variable for storing state names
le$region = tolower(le$State)
# Merging data
states = merge(states, le, by="region", all.x=T)
```

The results are then displayed on the US state map (note that some data is missing):

```
# Life expectancy of the African-American ethnic group
ggplot(states, aes(x = long, y = lat, group = group, fill = le_african)) +
geom_polygon(color = "white") + scale_fill_gradient(name = "Years", low =
"#ffe8ee", high = "#c81f49", guide = "colorbar", na.value="#eeeeee", breaks =
pretty_breaks(n = 5)) + labs(title="Life expectancy of the African-American
ethnic group") + coord_map()
# Life expectancy of the Caucasian ethnic group
ggplot(states, aes(x = long, y = lat, group = group, fill = le_caucasian)) +
geom_polygon(color = "white") + scale_fill_gradient(name = "Years", low =
"#ffe8ee", high = "#c81f49", guide = "colorbar", na.value="Gray", breaks =
pretty_breaks(n = 5)) + labs(title="Life expectancy of the Caucasian ethnic
group") + coord_map()
```

The differences in the life expectancy of the two ethnic groups are then displayed:

```
# Differences in the life expectancy
ggplot(states, aes(x = long, y = lat, group = group, fill = le_diff)) +
geom_polygon(color = "white") + scale_fill_gradient(name = "Years", low =
"#ffe8ee", high = "#c81f49", guide = "colorbar", na.value="#eeeeee", breaks =
pretty_breaks(n = 5)) + labs(title="Differences in the life expectancy of Cau-
casian and African-American\n ethnic groups in the USA by state") + coord_map()
```

The *plotly* package allows you to display the information (in the form of contextual messages) on a state by positioning the mouse on it:

```
# Plot the USA map with contextual messages for states with plotly
map_plot = ggplot(states, aes(x = long, y = lat, group = group, fill =
le_african)) + geom_polygon(color = "white") + scale_fill_gradient(name =
"Years", low = "#ffe8ee", high = "#c81f49", guide = "colorbar", na.value =
"#eeeeee", breaks = pretty_breaks(n = 5)) + labs(title="Life expectancy of the
African-American ethnic group") + coord_map()
ggplotly(map_plot)
```

4. Subsidiary Exercises

The purpose of these exercises is to reproduce the graphics for data exploration presented in the slides of the lecture course.

These graphics will be produced using the Weather dataset (`Data_Weather.csv`) containing 14 instances (rows) corresponding each to a day, and 5 variables (columns) describing meteorological conditions of the day (`Outlook`, `Temperature`, `Humidity` and `Windy`) and if the matches of the tennis tournament could take place this day (`Play`).

Variable names are given in the first row of the dataset, and columns are separated by a tabulation. The `Outlook` variable is categorical, the `Temperature` and `Humidity` variable are numerical, and the `Windy` and `Play` variables are boolean.

- Load the `Data_Weather.csv` dataset in a data frame `weather` with the command:

```
> weather <- read.csv("Data_Weather.csv", header = TRUE, sep = "\t")
```

- Display the dataset using the `View()` function:

```
> View(weather)
```

- Display the summary descriptive statistics for all variables of the `weather` data frame using the `summary()` function:

```
> summary(weather)
```

  For discrete variables (`Outlook`, `Windy` and `Play`), the counts of instances for each value of the variable are displayed. For continuous variables (`Temperature` and `Humidity`), the quartiles and the mean of the variable values are displayed.

- Create a boxplot of the `Temperature` variable for the two classes `Play=Yes` and `Play=No` with the command:

```
> boxplot(Temperature~Play, data=weather, col=c("red","turquoise"), main=" Tem-
perature Boxplots for each Class", ylab="Temperature", xlab="Play")
```

- Create a boxplot of the `Humidity` variable for the two classes `Play=Yes` and `Play=No`.
- Install/update the *ggplot2* library with the command:

```
> install.packages("ggplot2")
```

- Activate the *ggplot2* library for the session with the command:

```
> library(ggplot2)
```

- Create a histogram for the `Outlook` discrete variable with different colors for the two classes `Play=Yes` and `Play=No` using the `qplot()` function:

```
> qplot(Outlook, data=weather, fill=Play)
```

- Create a histogram for the `Windy` discrete variable with different colors for the two classes `Play=Yes` and `Play=No` using the `qplot()` function.
- Create a histogram for the `Temperature` numerical variable with different colors for the two classes

`Play=Yes` and `Play=No` and 5 bars (5 intervals of values) using the `qplot()` function:

```
> qplot(Temperature, data=weather, fill=Play, bins=5)
```

- Create a histogram for the `Humidity` numerical variable with different colors for the two classes `Play=Yes` and `Play=No` and 5 bars (5 intervals of values) using the `qplot()` function.
- Create a scatter plot for the `Temperature` and `Humidity` numerical variables with different colors for the two classes `Play=Yes` and `Play=No` using the `qplot()` function:

```
> qplot(Temperature, Humidity, data=weather, color=Play)
```

- Create a scatter plot for the `Temperature` numerical variable and the `Oultlook` discrete variable with different colors for the two classes `Play=Yes` and `Play=No` using the `qplot()` function:

```
> qplot(Temperature, Outlook, data=weather, color=Play)
```

- Create a scatter plot for the `Temperature` numerical variable and the `Windy` discrete variable with different colors for the two classes `Play=Yes` and `Play=No` using the `qplot()` function:
- Create a scatter plot for the `Outlook` and `Windy` discrete variables with different colors for the two classes `Play=Yes` and `Play=No` using the `qplot()` function:

```
> qplot(Outlook, Windy, data=weather, color=Play)
```

  Only 6 points are drawn on the scatter plot as there are only 6 possible combinations of values for the `Outlook` (`Sunny`, `Overcast` or `Rainy`) and `Windy` (`True` or `False`) variables. Instances with similar values for both variables are superimposed and thus indistinguishable.

- To distinguish the points that are superimposed in the scatter plot, add the following `geom_jitter()` formatting parameter after the previous `qplot()` command:

```
> qplot(…) + geom_jitter(width = 0.2, height = 0.2)
```

  The scale of the random displacement of each point with `geom_jitter()` can be parameterized by setting the `width` and `height` values.

- Create a scatter plot for the `Humidity` numerical variable and the `Windy` discrete variable with different colors for the two classes `Play=Yes` and `Play=No` using the `qplot()` function.

  How many points are displayed in the scatter plot?

  If necessary, use the `geom_jitter()` formatting parameter with the `qplot()` function to adapt the plot and distinguish all instances.

  Note that points are drawn for both the initial position and the randomly determined position.