DOCUMENTATION ABOUT PYHTON AND R WORKGROUP – Task 2.2

API: MAGIC THE GATHERING

C. Di Razza, M. Gioia, V. J. Striano

Our work was divided into two main tasks: data collection and data designing, and data analysis. This documentation relates to the second tasks.

Our purpose was to import a dataframe and through various techniques of manipulation and visualization, create graphs to simplify the visualization of the data in R. R is a free software environment for statistical computing and graphics.

Magic: The Gathering, also Magic or MTG, is a strategy card game created by Richard Garfield in 1993 and published by Wizards of the Coast. Magic holds the title of "Most Played Trading Card Game". The related API provides information about all the cards in the game.

The documentation related to the APIs are provided at the following link: https://api.magicthegathering.io/v1/cards

R packages are a collection of R functions, complied code and sample data. They are stored under a directory called "library" in the R environment. By default, R installs a set of packages during installation. More packages are added later when they are needed for some specific purpose. When we start the R console, only the default packages are available by default. Other packages which are already installed must be loaded explicitly to be used by the R program that is going to use them.

There are two ways to add new R packages. One is installing directly from the CRAN directory, and another is downloading the package to your local system and installing it manually. Before a package can be used in the code, it must be loaded to the current R environment. You also need to load a package that is already installed previously but not available in the current environment.

Library is a statement to load a packages. We load all the packages that we need with the command library().

[1] library (tidyverse)

…

[6]library(jsonlite)

We import the json dataframe from Simone Lu's GitHub repository thanks to the function fromJSON used to convert JSON data in R objects. We noticed that Simone and his team kept editing the source file, so we *forked* the "raw" version into our GitHub.

```
[9] cards <- fromJSON("url/.json")
```

We extract the dataframe from the json and filter the column of our interest.

```
[12] df <- as.data.frame(cards)

[13] df <- df[c(1,3,4,7,8,9,11,13,15,16,29)]
```

At this point we decide to create two new columns of variables. The new two columns contain:

- the average of each card representing the sum of power and toughness
- the ratio calculated as the ratio between the average and the mana cost of evocation of each card.

```
[16] df$average <- as.numeric(df$power) +
        as.numeric(df$toughness)

[17] df$ratio <- as.numeric(df$average) / df$cmc
```

We note that some columns contain null values named by the initials NA or special characters. We decide to omit those lines from our dataframe as they would complicate the graphic transposition. All this is possible thanks to the filter() function of the dplyr package. Its documentation available here:
https://dplyr.tidyverse.org/reference/filter.html

```
[20]df = filter(df, !(power== c("NA","*","*")))
```

We create a new dataframe by selecting only the columns that contain the quantitative variables. Thanks to the use of the pipe operator the code is better readable. The pipe operator, written as %>%, has been a longstanding feature of the magrittr package for R. Magrittr documentation available here: https://cran.r-project.org/web/packages/magrittr/vignettes/magrittr.html. It takes the output of one function and passes it into another function as an argument. This allows us to link a sequence of analysis steps.

```
[23] df_clean <- df  %>%  select(name, cmc, rarity, power:ratio)
        %>%  arrange(desc(ratio))
```

Out of curiosity we want to see how many legendary cards are present in this dataframe:

```
[28] legendary_c <- filter(df, supertypes == "Legendary"
```

Before starting to create charts, we import an image from the web.

```
[31] image_to_add <- ("url/1280px-Magicthegathering-
        logo.svg.png")
```

It’s time to create graphs with a specific function. Ggplot2 is one of the packages of tidyverse. Tidyverse is a set of packages needed for data science. You provide the data, tell ggplot2 how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details. It’s hard to succinctly describe how ggplot2 works because it embodies a deep philosophy of visualisation. However, in most cases you start with ggplot(), supply a dataset and aesthetic mapping (with aes()). You then add on layers (like geom_point() or geom_histogram()), scales (like scale_colour_brewer()), faceting specifications (like facet_wrap()) and coordinate systems (like coord_flip()). Ggplot guidelines available here: https://ggplot2.tidyverse.org

Themes are a powerful way to customize the non-data components of your plots. Themes can be used to give plots a consistent customized look. Modify a single plot's theme using theme(). The cowplot package is a simple add-on to ggplot. It provides various features that help with creating publication-quality figures, such as a set of themes, functions to align plots and arrange them into complex compound figures, and functions that make it easy to annotate plots and or mix plots with images. Documentation here: https://cran.r-project.org/web/packages/cowplot/vignettes/introduction.html

```
[32] theme_set(theme_cowplot())
```

We use the attach function not to repeat the *data* argument in ggplot.

```
[36] attach(df_clean)
```

We name the first graph ggp1:

```
[37] ggp1 <- ggplot(data = df_clean) + geom_count(mapping = aes(x
= cards.average , y = cards.cmc ))
```

```
[39] facet_wrap(~ cards.rarity, nrow = 3)
```

Finally, the image we imported can be inserted in our graph thanks to the ggdraw() function of the cowplot package specifying the coordinates and the scale to maintain in the graph:

```
[41] ggdraw() + draw_image(image_to_add, x = 0.4, y = -0.47, scale
= 0.14) + draw_plot(ggp1)
```

We create 7 other graphs of various types but with the pattern of commands quite similar.

Finally, the function ggarrange() in ggpubr is useful to arrange multiple ggplots over multiple pages. It can also create a common unique legend for multiple plots. For this function, we simply specify the different ggplot objects in order, followed by the number of columns (ncol) and number of rows (nrow).

```
[98] ggarrange(ggp1,ggp2, nrow = 2,ncol = 1)
```

```
[99] ggarrange(ggp3, ggp4, nrow = 2,ncol = 1)
```

```
[100] ggarrange(ggp5,ggp6,ggp7,ggp8,nrow=2, ncol= 2)
```

In the end, we also tried to visualize our code thanks to R Markdown, whose documentation is available here:
https://rmarkdown.rstudio.com/lesson-1.html

Unfortunately, some incompatibilities emerged between R Markdown and ggplot.

---
title: "R Markdown for Py&R Project - Task 2.2"
author: "Matteo Gioia and Vincenzo Junior Striano"
date: "December 13, 2021"
output:
html_document: default
pdf_document: default

We load all the packages that we need with the command library().

```{r message=FALSE, warning=FALSE}
library(tidyverse)
library(cowplot)
library(ggplot2)
library(magick)
library(ggpubr)
library(jsonlite)
```