Radar Chart in Markdown

Marco Baldo

26 March 2024

R Markdown introduction

Markdown is an **easy to use** format for writing reports. It resembles what you naturally write every time you compose an email or a word file. In fact, you may have already used markdown *without realizing it.* These websites all rely on markdown formatting. This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com. When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document.
CRAN
Github
R Markdown Book
HTML Radar chart markdown

Table 1. YAML Header options

Option	Description	Values
title	Title of the document	Text
author	Author(s) of the document	Text
date	Date of the document	YYYY-MM-DD
output	Output format and associated options	See below
toc	Include table of contents	`true`, `false`
toc_depth	Depth of table of contents	Numeric value
theme	Document theme	Theme names (e.g., `cosmo`, `lumen`)
css	Custom CSS file	File path or URL
fig_caption	Include captions for figures	`true`, `false`
code_folding	Enable code folding in HTML output	`hide`, `show`, `none`
df_print	Method for printing data frames	`paged`, `kable`, `tibble`, etc.
include_graphics	Method for including graphics	`base64`, `url`, `path`
highlight	Syntax highlighting theme for code chunks	Theme names (e.g., `tango`, `zenburn`)
latex_engine	LaTeX engine for PDF output	`pdflatex`, `xelatex`, `lualatex`
keep_md	Keep Markdown source file after rendering	`true`, `false`
keep_tex	Keep LaTeX intermediate file after rendering	`true`, `false`
number_sections	Number section headings	`true`, `false`

You can embed an R code chunk like this:

- eval = Specifies whether to evaluate the code chunks when rendering the document. When set to FALSE, the code chunks will not be executed, and their output will not be included in the final document. This can be useful when you want to show example code without actually running it.
- echo = Controls whether the code itself is displayed in the final document. When set to TRUE, the code chunks will be displayed along with their output. When set to FALSE, only the output of the code chunks will be displayed.
- message = Controls whether messages generated by the code chunks are displayed in the final document. When set to TRUE, messages will be displayed. When set to FALSE, messages will be suppressed.
- warnings = Controls whether warnings generated by the code chunks are displayed in the final document. When set to TRUE, warnings will be displayed. When set to FALSE, warnings will be suppressed.

Table 2. of chunk setting option

Option	Description	Possible Values
`eval`	Evaluate the code chunk	`TRUE`, `FALSE`
`echo`	Display the code in the output	`TRUE`, `FALSE`
`include`	Include the code and output	`TRUE`, `FALSE`
`message`	Display messages generated by the code	`TRUE`, `FALSE`
`warning`	Display warnings generated by the code	`TRUE`, `FALSE`
`error`	Display errors generated by the code	`TRUE`, `FALSE`
`results`	Display method for code results	`markup`, `hide`, etc.
`collapse`	Collapse the code chunk by default	`TRUE`, `FALSE`
`comment`	Add a comment to the code chunk	Text
`fig.cap`	Caption for output plots	Text
`fig.width`	Width of output plots	Numeric value
`fig.height`	Height of output plots	Numeric value
`out.width`	Width of output images	Numeric value
`out.height`	Height of output images	Numeric value
`fig.align`	Alignment of figures	`default`, `left`, etc.
`fig.pos`	Position of figures	`H`, `h`, `ht`, etc.
`dev`	Graphics device for plots	`pdf`, `png`, etc.
`cache`	Enable caching of code chunk results	`TRUE`, `FALSE`
`cache.path`	Directory for cached results	Directory path
`class.output`	Custom CSS classes for the output	CSS class names
`class.source`	Custom CSS classes for the source code	CSS class names
`class.note`	Custom CSS classes for chunk notes	CSS class names
`class.message`	Custom CSS classes for messages	CSS class names
`class.warning`	Custom CSS classes for warnings	CSS class names
`class.error`	Custom CSS classes for errors	CSS class names

Create Data - To let the radar chart work properly you need to specify maximum, minumum and the value of the variable. For example:

```
# Create a data frame
df <- data.frame(Mon=c(100, 0, 34),
                 Tue=c(100, 0, 48),
                 Wed=c(100, 0, 58),
                 Thu=c(100, 0, 67),
```

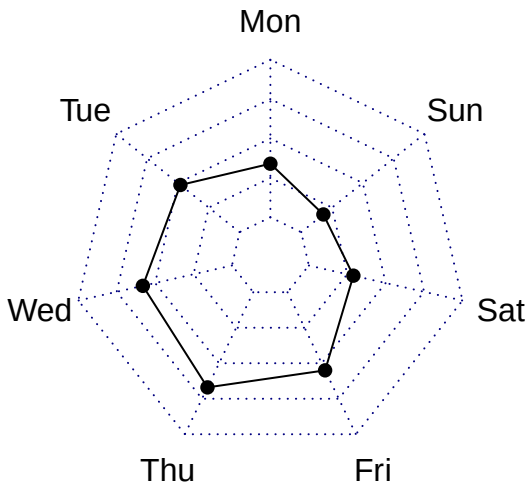
```
                 Fri=c(100, 0, 55),
                 Sat=c(100, 0, 29),
                 Sun=c(100, 0, 18))

# View data
df
```

```
##   Mon Tue Wed Thu Fri Sat Sun
## 1 100 100 100 100 100 100 100
## 2   0   0   0   0   0   0   0
## 3  34  48  58  67  55  29  18
```

```
# Once the data is in this format, we can use the radarchart() function from the # fmsb package to crea
# fmsb package to create a basic radar chart:

library(fmsb)
radarchart(df)
```



Customizing Radar Charts in R

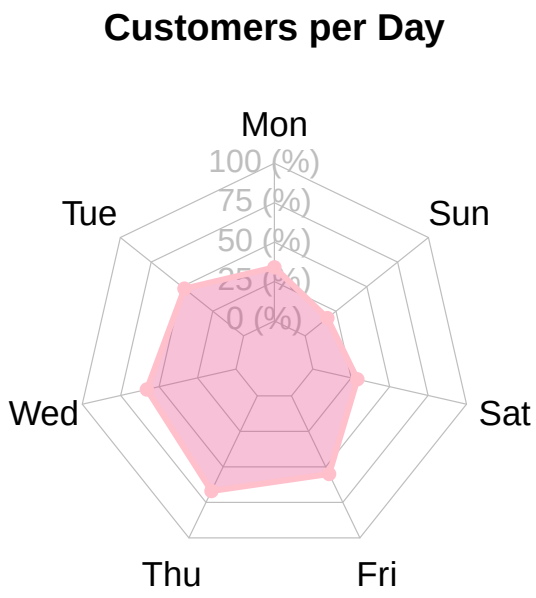
We can customize the radar chart by using the following arguments:

- pcol: Line color
- pfcol: Fill color
- plwd: Line width

- cglcol: Net color
- cglty: Net line type
- axislabcol: Axis label color
- caxislabels: Vector of axis labels to display
- cglwd: Net width
- vlcex: Group labels size

Here an example from the previous data frame (NB for radar chart you need always the specific data frame format):

```
radarchart(df,
           axistype=1,
           pcol='pink',
           pfcol=rgb(0.9,0.2,0.5,0.3),
           plwd=3,
           cglcol='grey',
           cglty=1,
           axislabcol='grey',
           cglwd=0.6,
           vlcex=1.1,
           title='Customers per Day'
)
```



Doing the same but with an average or threshold line - can be useful also for other analysis Example

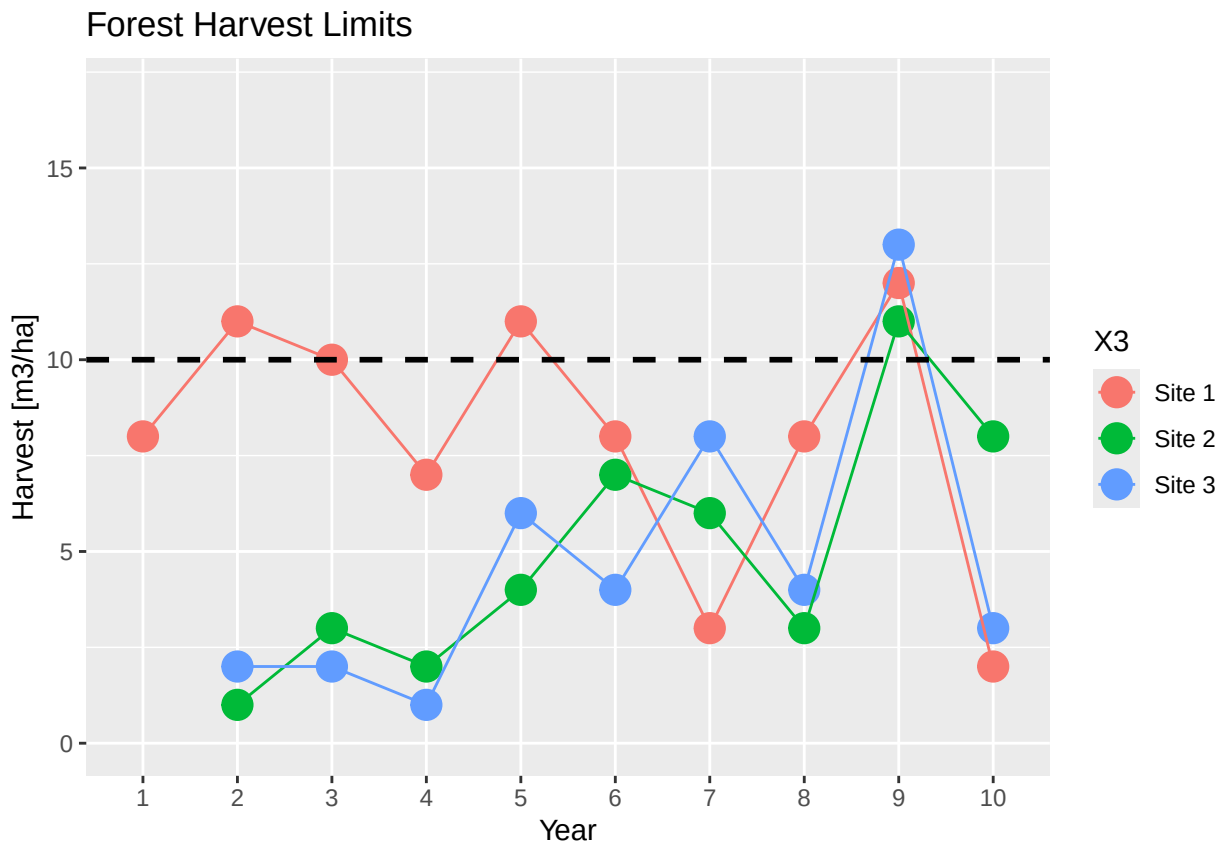
```r
library(ggplot2)

# Import data
data <- read.csv(
  "C:/Users/baldo/Documents/GitHub/R_intro_Markdown/Radar_chart/data.csv")

# View the structure of your data
str(data)
```

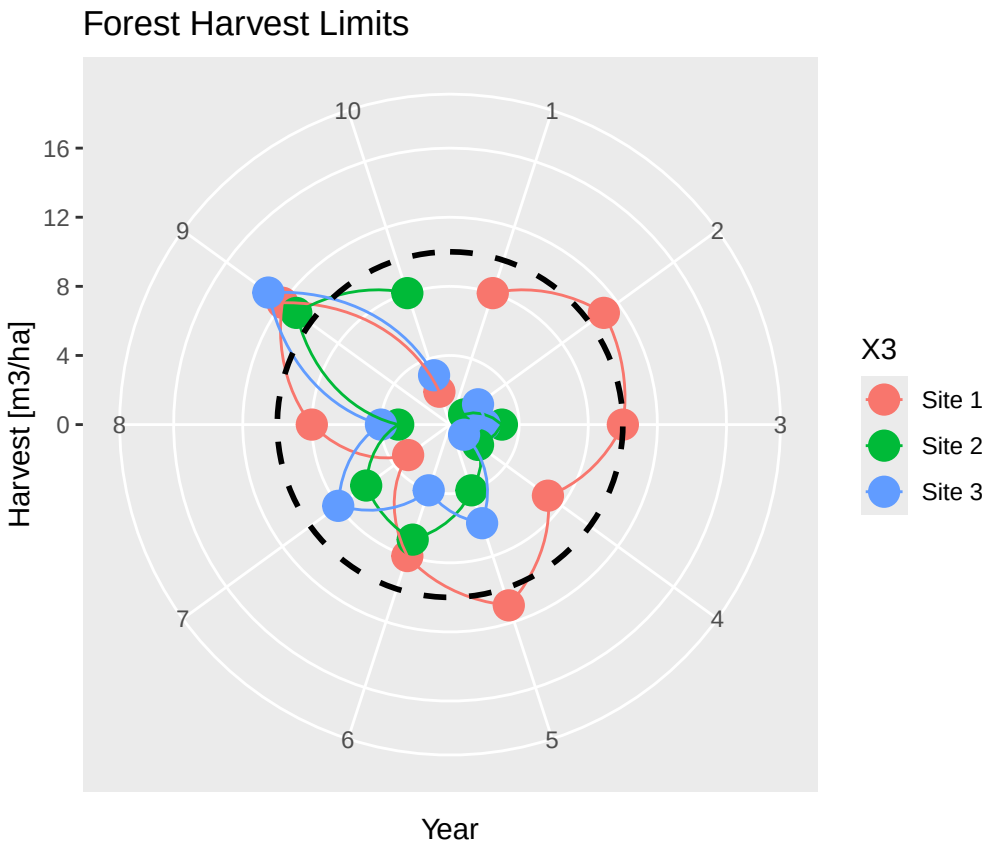
```
## 'data.frame':    30 obs. of  4 variables:
##  $ Harvest: num  8 NaN NaN 11 1 2 10 3 2 7 ...
##  $ X1     : chr  "Mean" "Mean" "Mean" "Mean" ...
##  $ X2     : int  1 1 1 2 2 2 3 3 3 4 ...
##  $ X3     : chr  "Site 1" "Site 2" "Site 3" "Site 1" ...
```

```r
# Plot you data
ggplot(data=data,  aes(x=X2, y=Harvest, group=X3, colour=X3)) +
  geom_point(size=5) +
  geom_line() +
  xlab("Year") +
  ylab("Harvest [m3/ha]") +
  ylim(0,17) + ggtitle("Forest Harvest Limits")  +
  geom_hline(aes(yintercept=10), lwd=1, lty=2) +
  scale_x_discrete(limits=c("1","2","3","4","5","6","7","8","9","10"))
```



Let's visualize it in a radar chart:

```
ggplot(data=data,  aes(x=X2, y=Harvest, group=X3, colour=X3)) +
  geom_point(size=5) +
  geom_line() +
  xlab("Year") +
  ylab("Harvest [m3/ha]") +
  ylim(0,17) + ggtitle("Forest Harvest Limits")  +
  geom_hline(aes(yintercept=10), lwd=1, lty=2) +
  scale_x_discrete(limits=c("1","2","3","4","5","6","7","8","9","10")) +
  coord_polar()
```



Followed by this website

Beautiful Radar Chart in R using `fmsb` and `ggplot2` Packages

A radar chart, also known as a spider plot is used to visualize the values or scores assigned to an individual over multiple quantitative variables, where each variable corresponds to a specific axis.

This article describes how to create a radar chart in R using two different packages: the fmsb or the ggradar R packages.

You will learn:

- how to create a beautiful fmsb radar chart
- how to create ggplot radar chart
- alternatives to radar charts

Demo data

We'll use a demo data containing exam scores for 3 students on 9 topics (Biology, Physics, etc). The scores range from 0 to 10. Columns are quantitative variables and rows are individuals:

```
# Demo data
exam_scores <- data.frame(
  row.names = c("Student.1", "Student.2", "Student.3"),
  Biology = c(7, 5, 9),
  Physics = c(6, 6, 8),
  Maths = c(6, 4, 7.5),
  Sport = c(7, 9, 4),
  English = c(8, 6, 7),
  Geography = c(6, 7, 9),
  Art = c(7, 5, 6),
  Programming = c(6, 4.5, 8),
  Music = c(7, 9, 6)
)
exam_scores
```

```
##           Biology Physics Maths Sport English Geography Art Programming Music
## Student.1       7       6   6.0     7       8         6   7         6.0     7
## Student.2       5       6   4.0     9       6         7   5         4.5     9
## Student.3       9       8   7.5     4       7         9   6         8.0     6
```

Data preparation

The data should be organized as follow:

- **The row 1 must contain the maximum values for each variable**
- **The row 2 must contain the minimum values for each variable**
- **Data for cases or individuals should be given starting from row 3**
- **The number of columns or variables must be more than 2.**

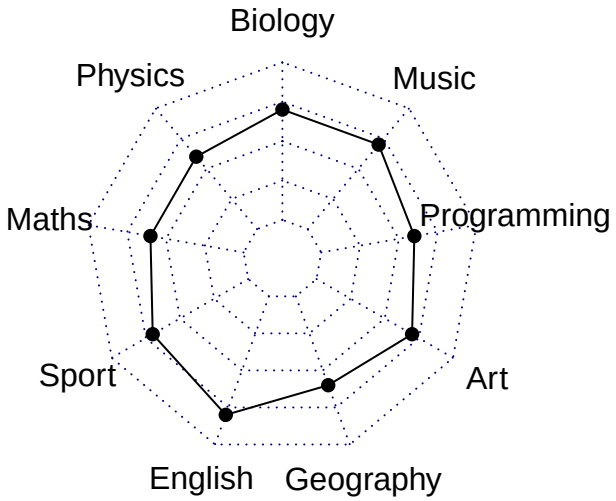
```
# Define the variable ranges: maximum and minimum
max_min <- data.frame(
  Biology = c(10, 0), Physics = c(10, 0), Maths = c(10, 0),
  Sport = c(10, 0), English = c(10, 0), Geography = c(10, 0),
  Art = c(10, 0), Programming = c(10, 0), Music = c(10, 0)
)
rownames(max_min) <- c("Max", "Min")

# Bind the variable ranges to the data
df <- rbind(max_min, exam_scores)
df
```

```
##           Biology Physics Maths Sport English Geography Art Programming Music
## Max            10      10  10.0    10      10        10  10        10.0    10
## Min             0       0   0.0     0       0         0   0         0.0     0
## Student.1       7       6   6.0     7       8         6   7         6.0     7
## Student.2       5       6   4.0     9       6         7   5         4.5     9
## Student.3       9       8   7.5     4       7         9   6         8.0     6
```

Basic radar plot

```r
# Plot the data for student 1
student1_data <- df[c("Max", "Min", "Student.1"), ]
radarchart(student1_data)
```



Customize the radar charts

Key arguments to customize the different components of the `fmsb` radar chart:

- Variable options
  - vlabels: variable labels
  - vlcex: controls the font size of variable labels
- Polygon options
  - pcol: line color
  - pfcol: fill color
  - plwd: line width
  - plty: line types. Can be a numeric vector 1:6 or a character vector c("solid", "dashed", "dotted", "dotdash", "longdash", "twodash"). To remove the line, use plty = 0 or plty = "blank"*
- Grid options
  - cglcol: line color

  - cglty: line type
  - cglwd: line width

- Axis options

  - axislabcol: color of axis label and numbers. Default is "blue"
  - caxislabels: Character vector to be used as labels on the center axis

Helper function to produce a beautiful radar chart:

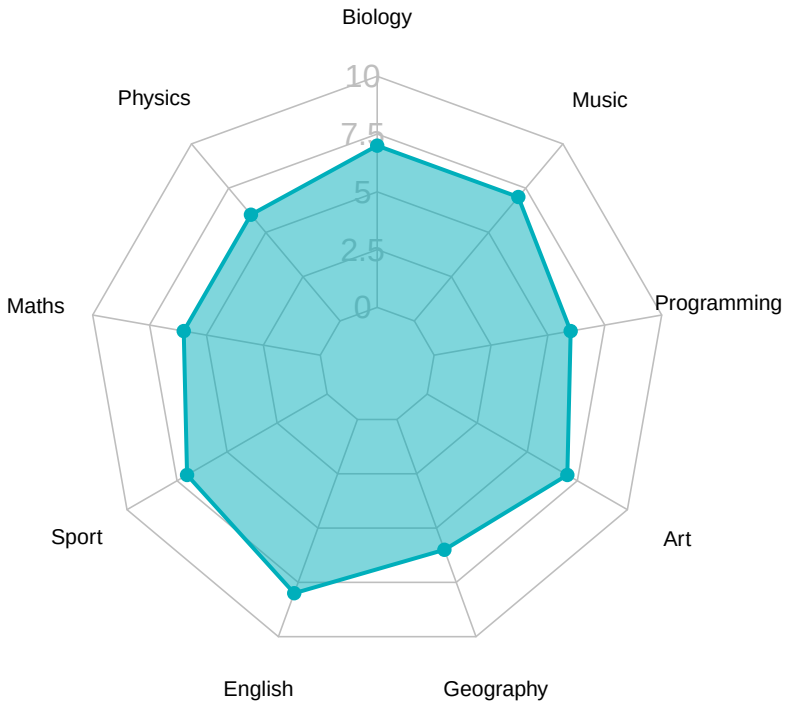
```
library(scales)

create_beautiful_radarchart <- function(data, color = "#00AFBB",
                                        vlabels = colnames(data), vlcex = 0.7,
                                        caxislabels = NULL, title = NULL, ...){
  radarchart(
    data, axistype = 1,
    # Customize the polygon
    pcol = color, pfcol = scales::alpha(color, 0.5), plwd = 2, plty = 1,
    # Customize the grid
    cglcol = "grey", cglty = 1, cglwd = 0.8,
    # Customize the axis
    axislabcol = "grey",
    # Variable labels
    vlcex = vlcex, vlabels = vlabels,
    caxislabels = caxislabels, title = title, ...
  )
}
```

In the code above, we used the function alpha() [in scales package] to change the polygon fill color transparency.

```
# Reduce plot margin using par()
op <- par(mar = c(1, 2, 2, 1))

create_beautiful_radarchart(student1_data, caxislabels = c(0, 2.5, 5, 7.5, 10))
```




```
par(op) # This line restores the original graphical parameters
```

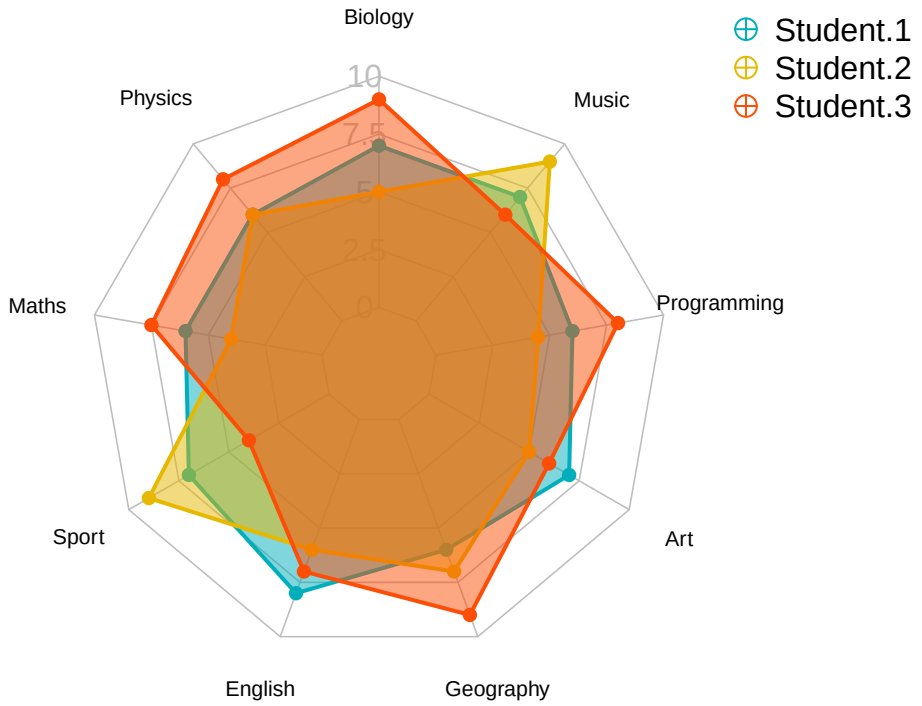
Create radar charts for multiple individuals

Create the radar chart of the three students on the same plot:

```
# Reduce plot margin using par()
op <- par(mar = c(1, 2, 2, 2))

# Create the radar charts
create_beautiful_radarchart(
  data = df, caxislabels = c(0, 2.5, 5, 7.5, 10),
  color = c("#00AFBB", "#E7B800", "#FC4E07")
)

# Add an horizontal legend
legend( x = "topright",
        legend = rownames(df[-c(1,2),]),
        horiz = FALSE,
        bty = "n",
        pch = 10 ,
        col = c("#00AFBB", "#E7B800", "#FC4E07"),
        text.col = "black",
        cex = 1,
        pt.cex = 1.5)
```




```
# Close the plot and reset graphical parameters
par(op)
```

In the legend() function of ggplot2, you can specify the position of the legend using the x and y arguments. Here are some of the possible positions for the legend:

- "top": Places the legend at the top of the plot.
- "bottom": Places the legend at the bottom of the plot.
- "left": Positions the legend on the left side of the plot.
- "right": Positions the legend on the right side of the plot.
- "topleft": Positions the legend in the top left corner of the plot.
- "topright": Positions the legend in the top right corner of the plot.
- "bottomleft": Positions the legend in the bottom left corner of the plot.
- "bottomright": Positions the legend in the bottom right corner of the plot.

c(x_coordinate, y_coordinate): Allows you to specify custom coordinates for the legend. You can provide the x and y coordinates as numeric values.

For example, if you want to place the legend on the top right corner of the plot, you can use legend(x = "topright"). Similarly, if you want to specify custom coordinates, you can use legend(x = 0.8, y = 0.2) to position the legend at the coordinates (0.8, 0.2) within the plot.

These different positions give you flexibility in placing the legend in various parts of the plot depending on your visualization requirements.

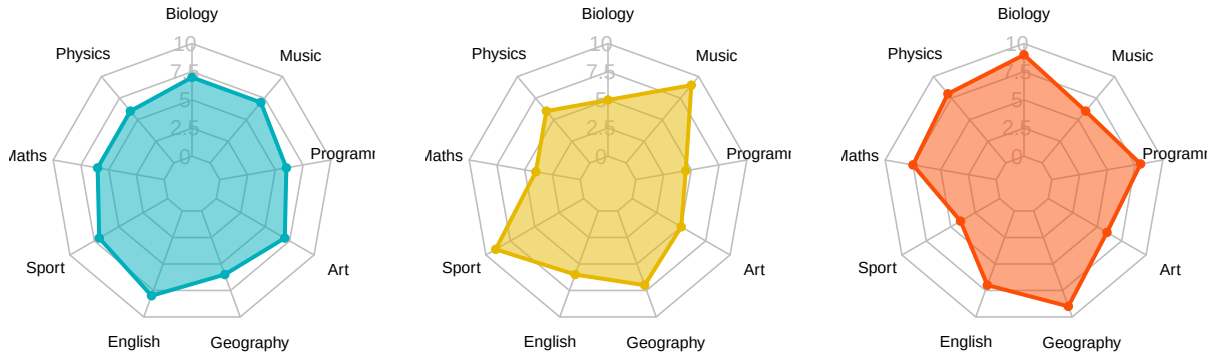
Create separated spider charts for each individual. This is recommended when you have more than 3 series.

```r
# Define colors and titles
colors <- c("#00AFBB", "#E7B800", "#FC4E07")
titles <- c("Student.1", "Student.2", "Student.3")

# Reduce plot margin using par()
# Split the screen in 3 parts
op <- par(mar = c(1, 1, 1, 1))
par(mfrow = c(1,3))

# Create the radar chart
for(i in 1:3){
  create_beautiful_radarchart(
    data = df[c(1, 2, i+2), ],
    caxislabels = c(0, 2.5, 5, 7.5, 10),
    color = colors[i],
    title = titles[i]
  )
}
```

Student.1 **Student.2** **Student.3**




```r
par(op)
```

Compare every profile to an average profile

Radar charts are most useful if the profile of every individual is compared to an average profile.

```r
# Create a demo data containing exam scores for 10 students:

set.seed(123) # To set the randomization of values

df <- as.data.frame(
  matrix(sample(2:10 , 90 , replace = TRUE),
         ncol=9, byrow = TRUE)
)

# In this example the matrix we build up has a vote (variable) between 1 and 10,
# x90 times (total votes=values 10 student x9 subjects), replace the value in
# the empty matrix. ncol = at the number of subject (9), order the frame by
# number of rows of the data frame = 10!

df
```

```
##    V1 V2 V3 V4 V5 V6 V7 V8 V9
## 1   4  4  3  7  6  5  7 10  6
## 2   4 10 10 10  4  9  8 10  4
## 3   5  2  8  6  8 10 10  8  6
## 4   8  6  7 10  3  6  9  3  2
## 5  10 10  7  6 10  5  7  9  7
## 6   7  8  2  7  3  2  3  5  6
## 7   7  4 10  5  7 10 10  8  4
## 8   9 10  4  8  4  8  7  6  6
## 9   9  4  3  3  7  5  2  7  4
## 10  9  4  9  2  8  8  8  7  8
```

```r
colnames(df) <- c(
  "Biology", "Physics", "Maths", "Sport", "English",
  "Geography", "Art", "Programming", "Music"
)
rownames(df) <- paste0("Student.", 1:nrow(df))
head(df)
```

```
##           Biology Physics Maths Sport English Geography Art Programming Music
## Student.1       4       4     3     7       6         5   7          10     6
## Student.2       4      10    10    10       4         9   8          10     4
## Student.3       5       2     8     6       8        10  10           8     6
## Student.4       8       6     7    10       3         6   9           3     2
## Student.5      10      10     7     6      10         5   7           9     7
## Student.6       7       8     2     7       3         2   3           5     6
```

Rescale each variable to range between 0 and 1:

The provided code performs the following operations:

- `df_scaled <- round(apply(df, 2, scales::rescale), 2)`: *This line scales the columns of the data frame df using the scales::rescale function from the scales package. The apply() function is used to apply the scaling operation column-wise (2 indicates columns) on the data frame. The rescale function scales the values of each column to a new range (usually between 0 and 1). The resulting scaled values are then rounded to two decimal places using the round() function. The scaled values are assigned to a new data frame called df_scaled.*

- `df_scaled <- as.data.frame(df_scaled)`: *This line converts the df_scaled object (which was previously a matrix resulting from the apply function) into a data frame. This step is done to ensure that the resulting object is in the desired data frame format.*
- `head(df_scaled)`: *This line displays the first few rows of the df_scaled data frame, allowing you to inspect the scaled values.*

In summary, the code scales the columns of the original data frame df using the rescale function, rounds the scaled values, converts the result into a data frame, and then displays the first few rows of the scaled data frame. This is often done to normalize or standardize the values in a dataset, making them comparable or easier to interpret.

```
# Scales each column to the range [0, 1]
df_scaled <- round(apply(df, 2, scales::rescale), 2)

# Structure as a data frame
df_scaled <- as.data.frame(df_scaled)

head(df_scaled)
```

```
##           Biology Physics Maths Sport English Geography  Art Programming Music
## Student.1    0.00    0.25  0.12  0.62    0.43      0.38 0.62        1.00  0.67
## Student.2    0.00    1.00  1.00  1.00    0.14      0.88 0.75        1.00  0.33
## Student.3    0.17    0.00  0.75  0.50    0.71      1.00 1.00        0.71  0.67
## Student.4    0.67    0.50  0.62  1.00    0.00      0.50 0.88        0.00  0.00
## Student.5    1.00    1.00  0.62  0.50    1.00      0.38 0.62        0.86  0.83
## Student.6    0.50    0.75  0.00  0.62    0.00      0.00 0.12        0.29  0.67
```

To scale the data between 0 and 9 instead of the default range (usually 0 to 1), you can modify the code as follows:

```
df_scaled <- round(apply(df, 2, function(x) scales::rescale(x, to = c(0, 9))), 2)

df_scaled <- as.data.frame(df_scaled)

head(df_scaled)
```

Prepare the data for creating the radar plot using the fmsb package:

```
# Variables summary
# Get the minimum and the max of every column
col_max <- apply(df_scaled, 2, max)
col_min <- apply(df_scaled, 2, min)
# Calculate the average profile
col_mean <- apply(df_scaled, 2, mean)
# Put together the summary of columns
col_summary <- t(data.frame(Max = col_max, Min = col_min, Average = col_mean))


# Bind variables summary to the data
df_scaled2 <- as.data.frame(rbind(col_summary, df_scaled))
head(df_scaled2)
```

```
##           Biology Physics Maths Sport English Geography   Art Programming Music
## Max         1.000   1.000 1.000 1.000   1.000     1.000 1.000       1.000  1.00
```

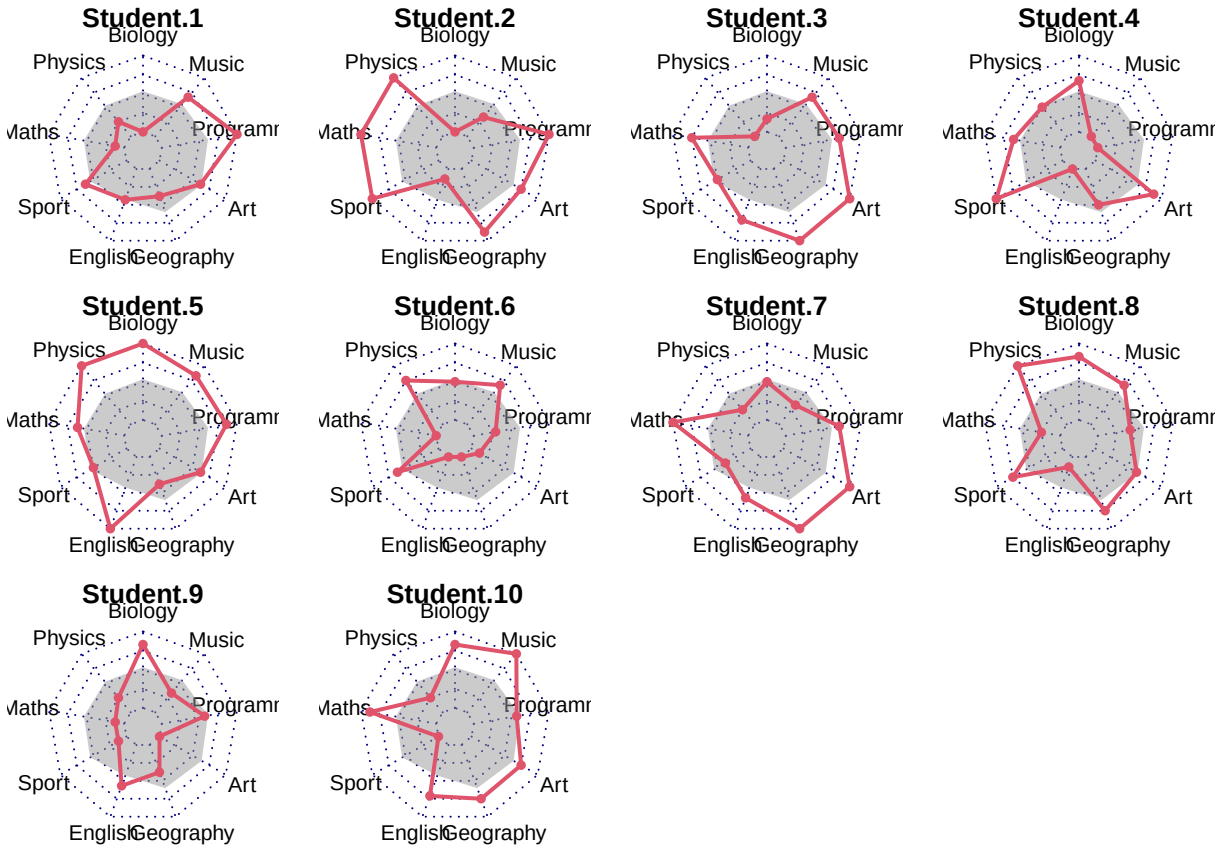
```
## Min          0.000   0.000 0.000 0.000   0.000     0.000 0.000       0.000  0.00
## Average      0.533   0.525 0.536 0.549   0.427     0.602 0.636       0.614  0.55
## Student.1    0.000   0.250 0.120 0.620   0.430     0.380 0.620       1.000  0.67
## Student.2    0.000   1.000 1.000 1.000   0.140     0.880 0.750       1.000  0.33
## Student.3    0.170   0.000 0.750 0.500   0.710     1.000 1.000       0.710  0.67
```

Here we created a data frame in which are present also the minimum scaled value, the maximum scaled value and the mean scaled value, for every subject in our matrix. This is added to the original data frame above the others rows.

Produce radar plots showing both the average profile and the individual profile:

```
opar <- par()
# Define settings for plotting in a 3x4 grid, with appropriate margins:
par(mar = rep(0.8,4))
par(mfrow = c(3,4))

# Produce a radar-chart for each student
for (i in 4:nrow(df_scaled2)) {
  radarchart(
    df_scaled2[c(1:3, i), ],
    pfcol = c("#99999980",NA),
    pcol= c(NA,2), plty = 1, plwd = 2,
    title = row.names(df_scaled2)[i]
  )
}
# Restore the standard par() settings
par <- par(opar)
```



The provided code performs the following operations:

- 1. `opar <- par()`: This line saves the current graphical parameters in the `opar` object. It is done so that you can later restore the original settings after making changes.
- 2. `par(mar = rep(0.8, 4))`: This sets the margins (in inches) for the plot. By using `rep(0.8, 4)`, it sets all four margins (bottom, left, top, right) to a value of 0.8 inches. This adjusts the space between the plot area and the edges of the graphics device.
- 3. `par(mfrow = c(3, 4))`: This sets the layout of the plots in a 3x4 grid. It divides the graphics device into a grid with 3 rows and 4 columns, allowing you to create multiple plots in a single figure.
- 4. The `for` loop: This loop iterates over the rows of the `df_scaled2` data frame, starting from the 4th row (`4:nrow(df_scaled2)`). It produces a radar chart for each student using the `radarchart()` function.
- 5. Inside the loop, `radarchart()` function is called with the following arguments:
  - `df_scaled2[c(1:3, i), ]`: This selects the rows 1 to 3 and the current `i`th row from the `df_scaled2` data frame, creating a subset of the data to be plotted.
  - `pfcol = c("#99999980", NA)`: This sets the fill color for the polygon area in the radar chart. The first color `#99999980` represents a light gray color with 50% transparency, and `NA` indicates no fill color for the data points.
  - `pcol = c(NA, 2)`: This sets the color for the polygon border and the data points. `NA` means no border color for the polygon, and `2` represents a color code for the data points.
  - `plty = 1`: This sets the line type for the polygon border to a solid line.
  - `plwd = 2`: This sets the line width for the polygon border to a value of 2.

- `title = row.names(df_scaled2)[i]`: This sets the title of the radar chart to the row name of the current ith row in the `df_scaled2` data frame.

- 6. `par <- par(opar)`: This line restores the original graphical parameters by assigning the saved `opar` object back to `par`. It ensures that the settings are reverted to the state before the changes made in the code.

Overall, this code generates a grid of radar charts, with each chart representing a different student's data. The charts are created using the `radarchart()` function and customized using various parameters. The `par()` function is used to modify and restore graphical settings for the plot.

==========

Final Section

ggplot radar chart using the ggradar R package

Prerequisites:

`install.packages("devtools") devtools::install_github("ricardo-bion/ggradar", dependencies = TRUE)`

```
library (ggradar)
```

Key function and arguments

`ggradar(plot.data, values.radar = c("0%", "50%", "100%"),grid.min = 0, grid.mid = 0.5, grid.max = 1, )`

- plot.data: data containing one row per individual or group
- values.radar: values to show at minimum, average and maximum grid lines
- grid.min: value at which minimum grid line is plotted
- grid.mid: value at which average grid line is plotted
- grid.max: value at which maximum grid line is plotted

Data preparation

NB:

All variables in the data should be at the same scale. If this is not the case, you need to rescale the data.

For example, you can rescale the variables to have a minimum of 0 and a maximum of 1 using the function `rescale()` [scales package]. We'll describe this method in the next sections.

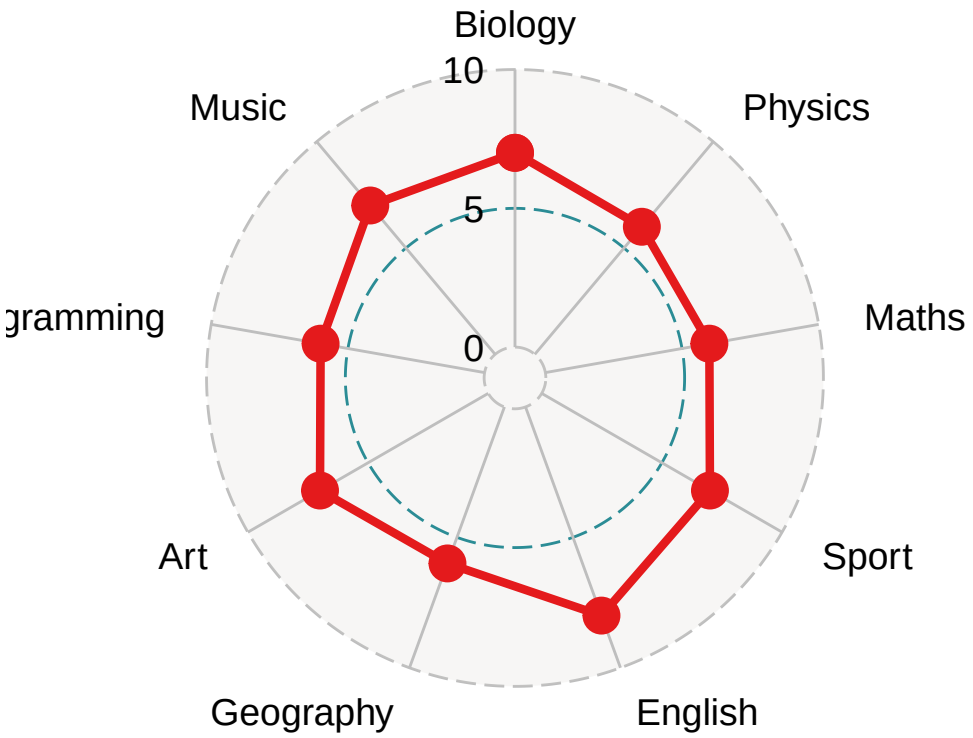
```
library(tidyverse)

# Put row names into  a column named group
df <- exam_scores %>% rownames_to_column("group")
df
```

```
##       group Biology Physics Maths Sport English Geography Art Programming Music
## 1 Student.1       7       6   6.0     7       8         6   7         6.0     7
## 2 Student.2       5       6   4.0     9       6         7   5         4.5     9
## 3 Student.3       9       8   7.5     4       7         9   6         8.0     6
```

Basic radar plot

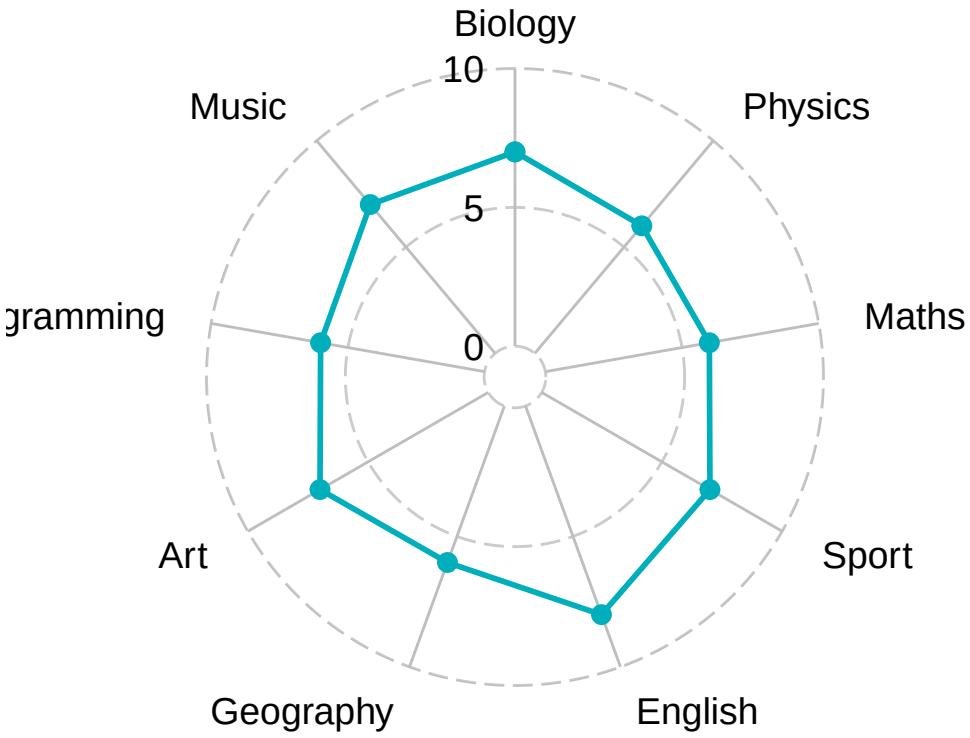
```r
# Plotting student 1
ggradar(
  df[1, ],
  values.radar = c("0", "5", "10"),
  grid.min = 0, grid.mid = 5, grid.max = 10
)
```



Customize radar charts

Key arguments to customize the different components of the ggplot radar chart. For more options see the documentation.

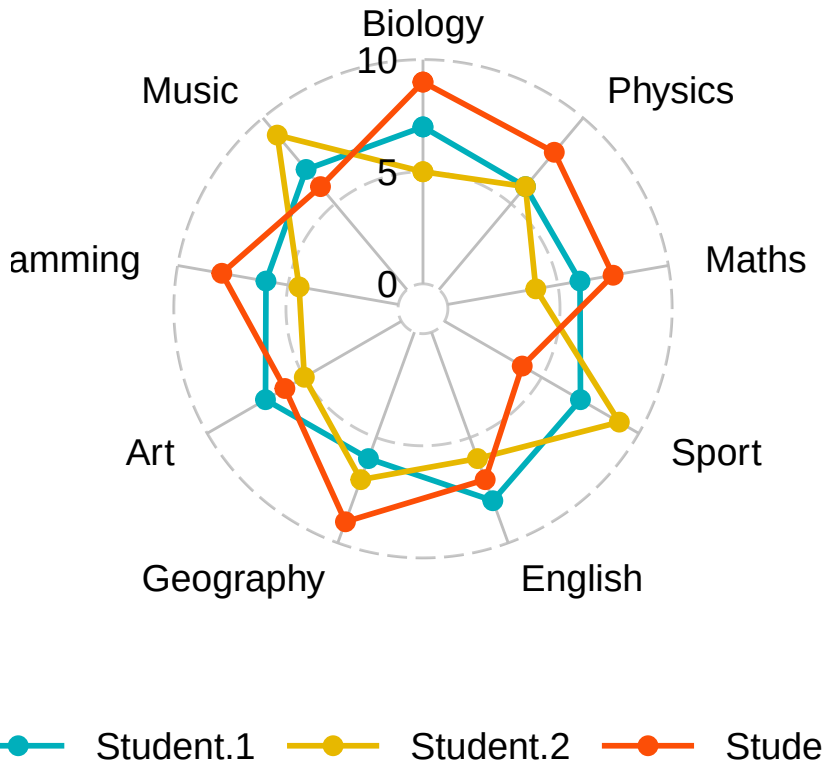
```r
ggradar(
  df[1, ],
  values.radar = c("0", "5", "10"),
  grid.min = 0, grid.mid = 5, grid.max = 10,
  # Polygons
  group.line.width = 1,
  group.point.size = 3,
  group.colours = "#00AFBB",
  # Background and grid lines
  background.circle.colour = "white",
  gridline.mid.colour = "grey"
  )
```



Radar chart with multiple individuals or groups

Create the radar chart of the three students on the same plot:

```
ggradar(
  df,
  values.radar = c("0", "5", "10"),
  grid.min = 0, grid.mid = 5, grid.max = 10,
  # Polygons
  group.line.width = 1,
  group.point.size = 3,
  group.colours = c("#00AFBB", "#E7B800", "#FC4E07"),
  # Background and grid lines
  background.circle.colour = "white",
  gridline.mid.colour = "grey",
  legend.position = "bottom"
  )
```



Alternatives to radar charts

A circular plot is difficult to read. An alternative to a radar chart is an ordered lolliplot or dotchart. This section describes how to create dotcharts. The ggpubr R package will be used in this section to create a dotchart.

Load required packages:

```
library(ggpubr)
```

Case when all quantitative variables have the same scale

Displaying one individual

Data preparation:

```
df2 <- t(exam_scores) %>%
  as.data.frame() %>%
  rownames_to_column("Field")
df2
```

```
##         Field Student.1 Student.2 Student.3
## 1     Biology         7       5.0       9.0
## 2     Physics         6       6.0       8.0
## 3       Maths         6       4.0       7.5
## 4       Sport         7       9.0       4.0
## 5     English         8       6.0       7.0
```

```
## 6   Geography         6       7.0       9.0
## 7         Art         7       5.0       6.0
## 8 Programming         6       4.5       8.0
## 9       Music         7       9.0       6.0
```

The code provided performs the following operations:

- **t(exam_scores)**: This transposes the exam_scores object. If exam_scores is a matrix or data frame, transposing it will swap the rows and columns, resulting in a new object.

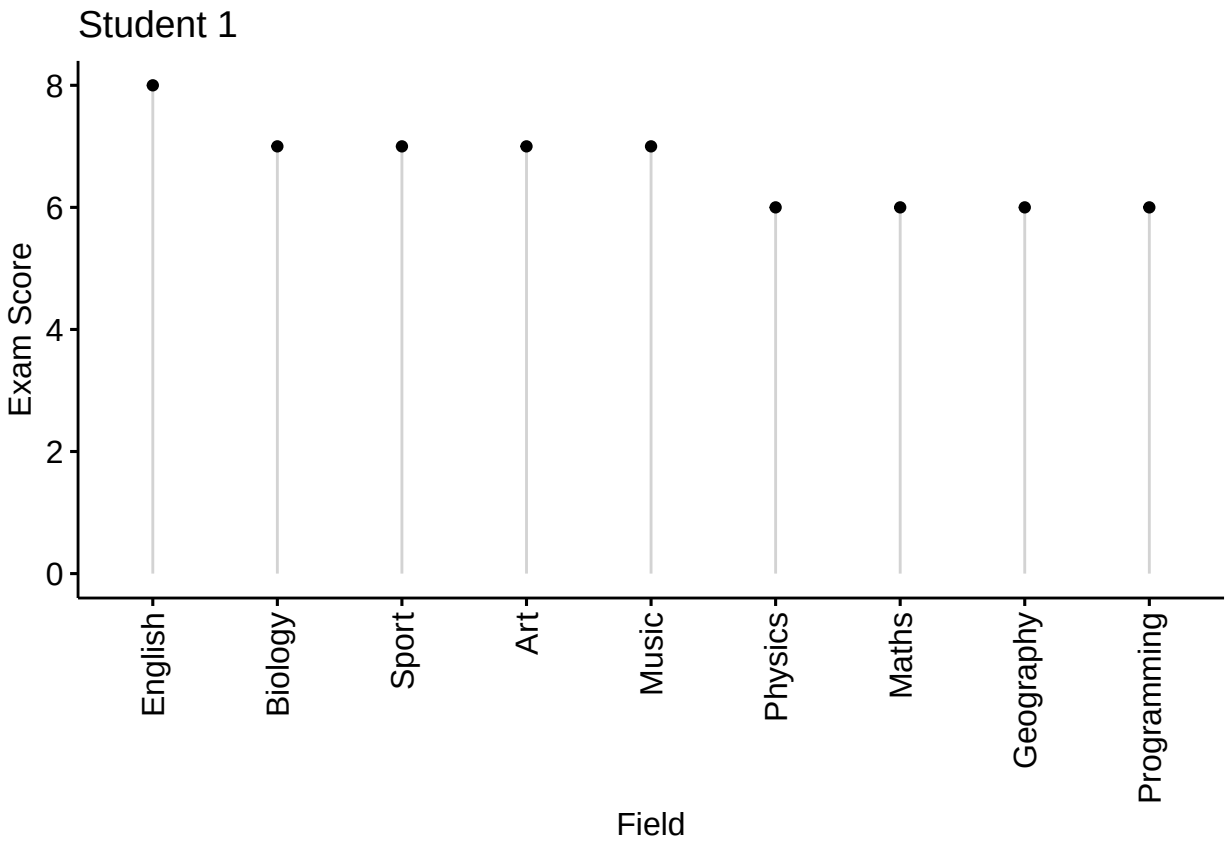
%>%: This is the pipe operator, which allows you to chain multiple operations together. It takes the output from the previous operation and passes it as the first argument to the next function call.

- **as.data.frame()**: This converts the transposed object into a data frame. If exam_scores was originally a matrix, this step converts it into a data frame format.
- **rownames_to_column("Field")**: This function from the dplyr package adds a new column called "Field" to the data frame, containing the row names from the transposed object. It assigns the row names as a separate column, making them accessible for further data manipulation or analysis.
- **df2**: This assigns the resulting object from the previous operations to the variable df2.

Overall, this code takes the exam_scores object, transposes it, converts it into a data frame, and adds a new column with the row names. The final result is stored in the df2 variable, which contains the transposed data with row names as a separate column.

Plot creation:

```
ggdotchart(
  df2, x = "Field", y = "Student.1",
  add = "segments", sorting = "descending",
  ylab = "Exam Score", title = "Student 1"
  )
```



Displaying two individuals

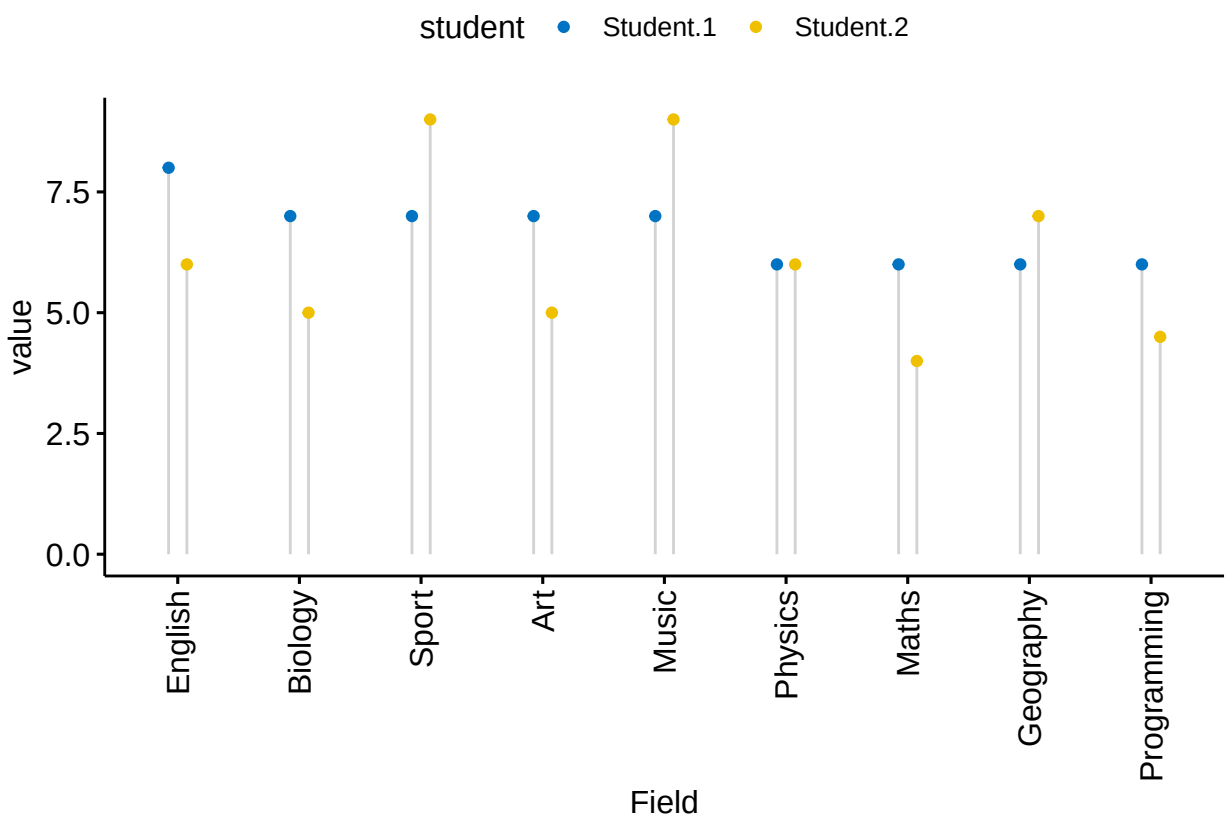
Data preparation:

```
df3 <- df2 %>%
  select(Field, Student.1, Student.2) %>%
  pivot_longer(
    cols = c(Student.1, Student.2),
    names_to = "student",
    values_to = "value"
  )
head(df3)
```

```
## # A tibble: 6 x 3
##   Field   student   value
##   <chr>   <chr>     <dbl>
## 1 Biology Student.1     7
## 2 Biology Student.2     5
## 3 Physics Student.1     6
## 4 Physics Student.2     6
## 5 Maths   Student.1     6
## 6 Maths   Student.2     4
```

Plot creation

```
ggdotchart(
  df3, x = "Field", y = "value",
  group = "student", color = "student", palette = "jco",
  add = "segment", position = position_dodge(0.3),
  sorting = "descending"
  )
```



Displaying multiple individuals

Data preparation:

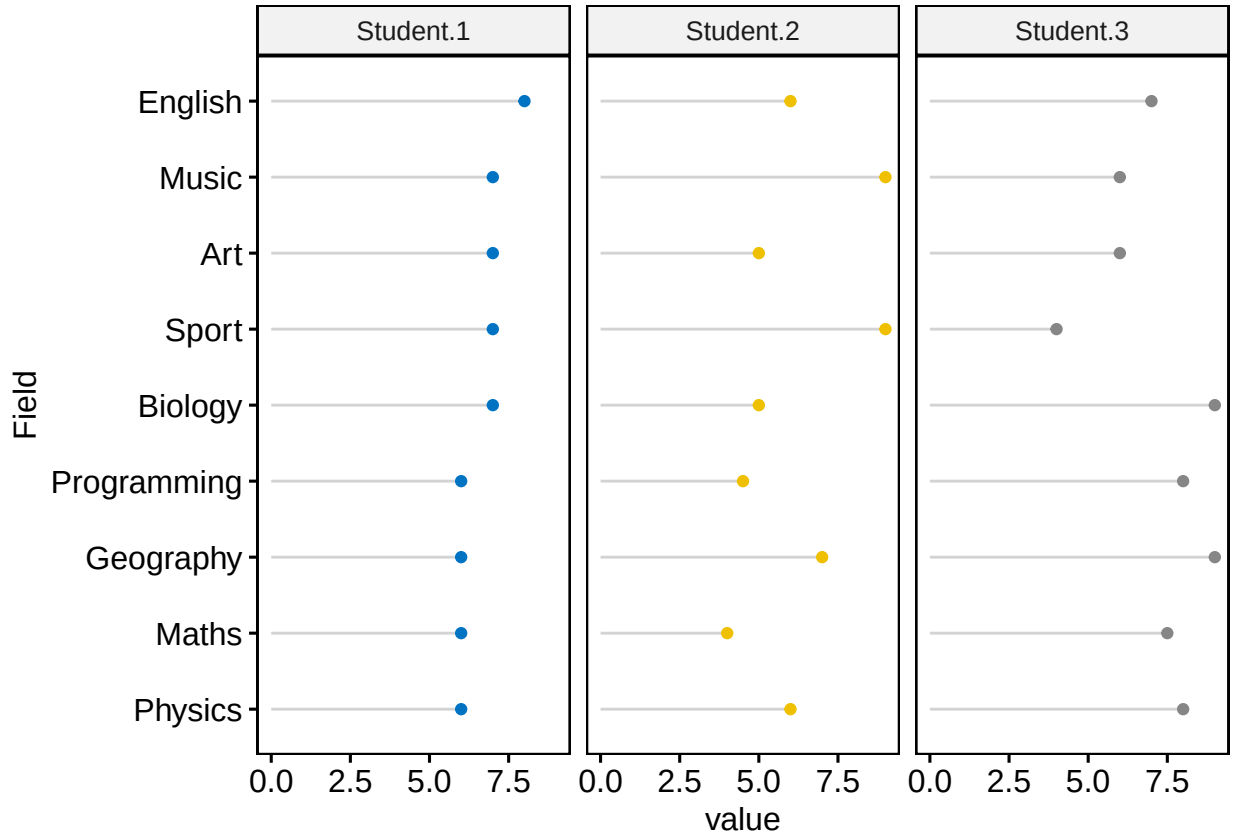
```
df4 <- df2 %>%
  select(Field, Student.1, Student.2, Student.3) %>%
  pivot_longer(
    cols = c(Student.1, Student.2, Student.3),
    names_to = "student",
    values_to = "value"
  )
head(df4)
```

```
## # A tibble: 6 x 3
##   Field   student   value
##   <chr>   <chr>     <dbl>
## 1 Biology Student.1     7
## 2 Biology Student.2     5
## 3 Biology Student.3     9
```

```
## 4 Physics Student.1     6
## 5 Physics Student.2     6
## 6 Physics Student.3     8
```

Plot creation

```
ggdotchart(
  df4, x = "Field", y = "value",
  group = "student", color = "student", palette = "jco",
  add = "segment", position = position_dodge(0.3),
  sorting = "descending", facet.by = "student",
  rotate = TRUE, legend = "none"
  )
```



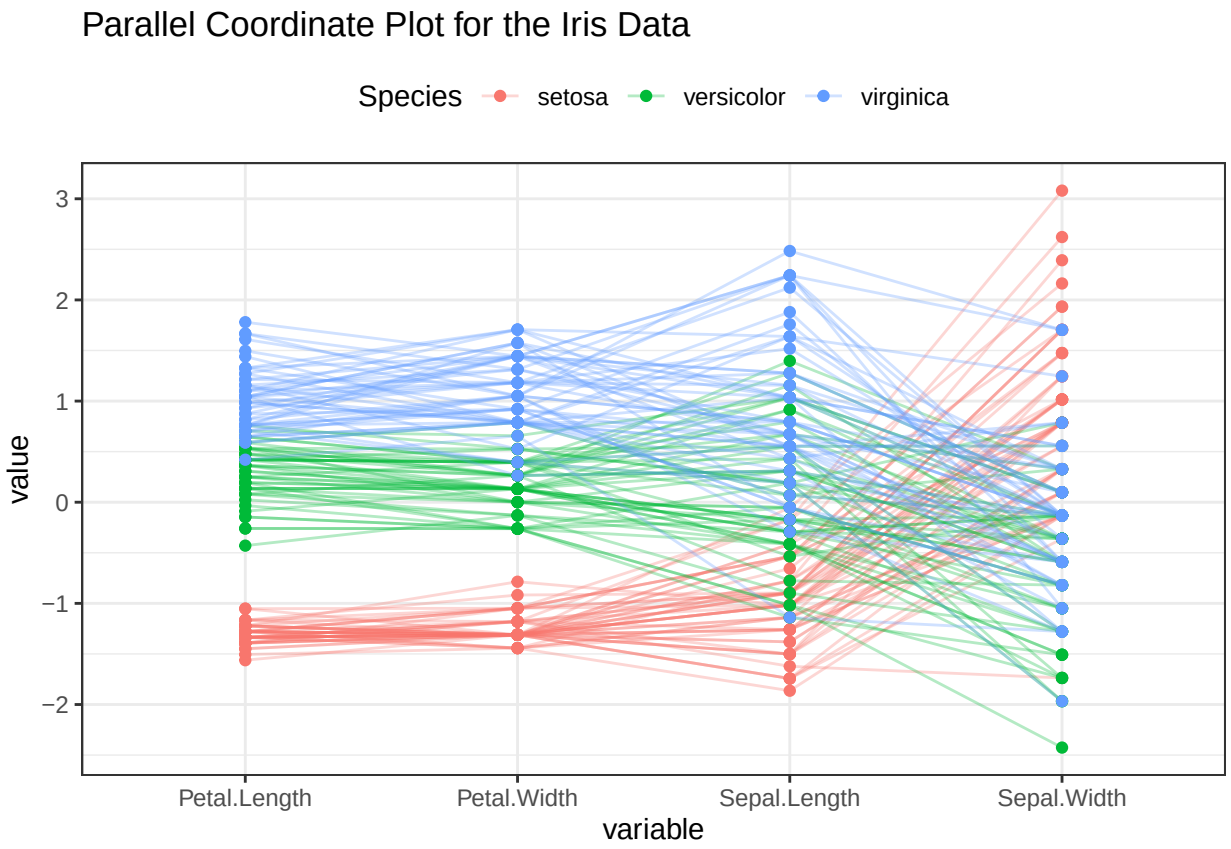
Case when you have a lot of individuals to plot or if your variables have different scales

A solution is to create a parallel coordinates plot.

```
library(GGally)

ggparcoord(
  iris,
```

```
  columns = 1:4, groupColumn = 5, order = "anyClass",
  showPoints = TRUE,
  title = "Parallel Coordinate Plot for the Iris Data",
  alphaLines = 0.3
) +
  theme_bw() +
  theme(legend.position = "top")
```



Note that, the default of the function ggparcoord() is to rescale each variable by subtracting the mean and dividing by the standard deviation.

```
sessionInfo()
```