Radar Chart in Markdown

Marco

2023-06-19

Script for a Radar Chart example with R

Markdown is an **easy to use** format for writing reports. It resembles what you naturally write every time you compose an email. In fact, you may have already used markdown *without realizing it*. These websites all rely on markdown formatting. In case word is not working use *tinytex::install_tinytex()* before to start.
* Github * StackOverflow * Reddit
eval = TRUE evaluate the script
echo = FALSE means only the result shown
message = FALSE remove the messages from the report
warnings = FALSE remove the warning messages

R Markdown

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.

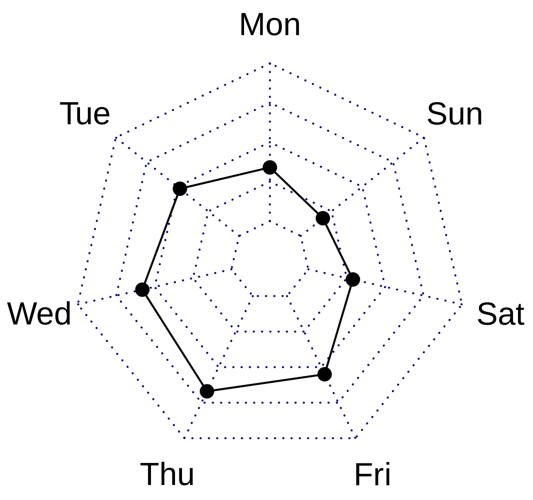
When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

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

```r
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 library to create a basic radar chart:


radarchart(df)
```



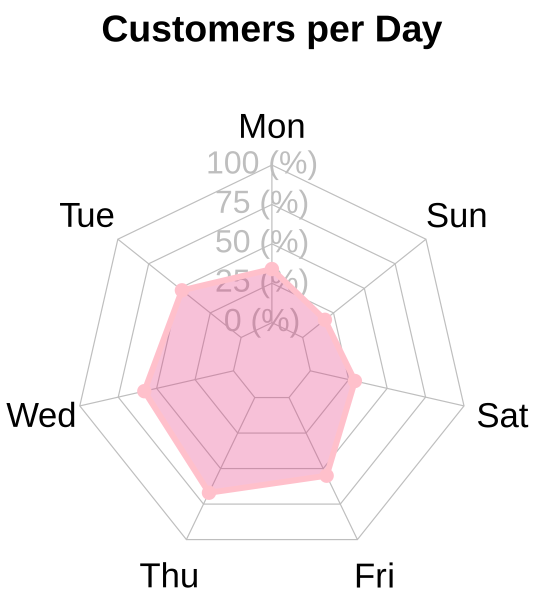
Customizing Radar Charts in R

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 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'
)
```



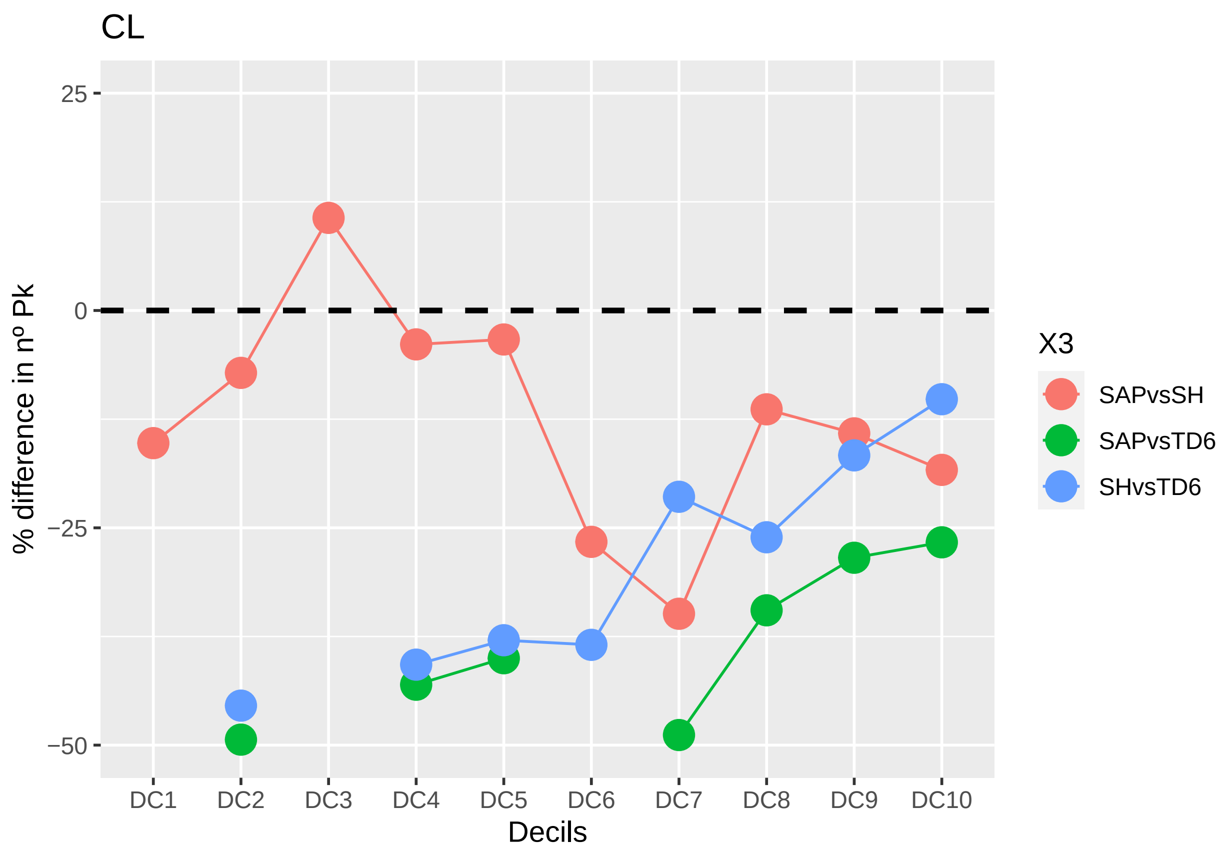
Second Part

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

```
data <- read.csv("C:/_R/R_visual_radar/Radar_chart/data.csv")
str(data)
```

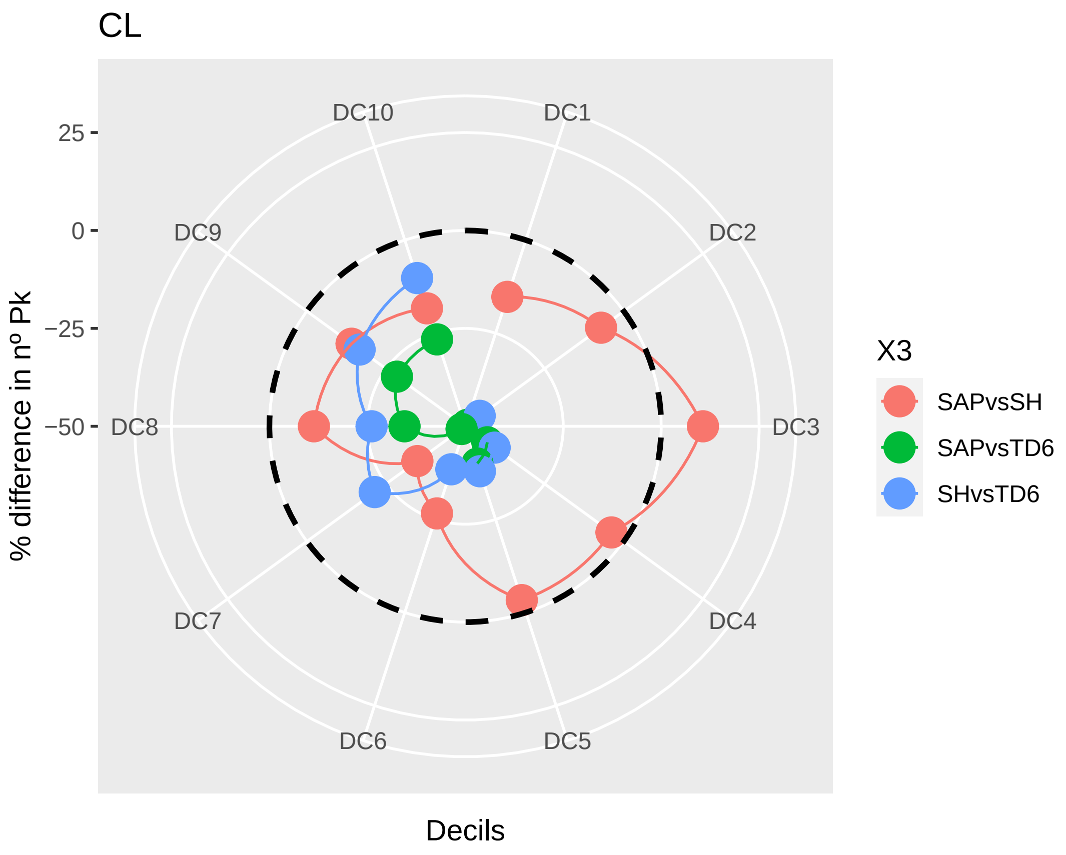
```
## 'data.frame':    30 obs. of  5 variables:
##  $ X    : chr  "Mean_DC1_SAPvsSH" "Mean_DC1_SAPvsTD6" "Mean_DC1_SHvsTD6" "Mean_DC2_SAPvsSH" ...
##  $ Count: num  -15.26 NaN NaN -7.17 -49.37 ...
##  $ X1   : chr  "Mean" "Mean" "Mean" "Mean" ...
##  $ X2   : chr  "DC1" "DC1" "DC1" "DC2" ...
##  $ X3   : chr  "SAPvsSH" "SAPvsTD6" "SHvsTD6" "SAPvsSH" ...
```

```
# data <- data[,2:5]
# str(data)


ggplot(data=data,  aes(x=X2, y=Count, group=X3, colour=X3)) +
  geom_point(size=5) +
  geom_line() +
  xlab("Decils") +
  ylab("% difference in nº Pk") +
  ylim(-50,25) + ggtitle("CL")  +
  geom_hline(aes(yintercept=0), lwd=1, lty=2) +
  scale_x_discrete(limits=c("DC1","DC2","DC3","DC4","DC5","DC6","DC7","DC8","DC9","DC10"))
```



I would like to transform this chart in a radar chart. I tried to use ggradar or ggRadar, but unsuccessfully. Something like this would be amazing:

```
ggplot(data=data,  aes(x=X2, y=Count, group=X3, colour=X3)) +
  geom_point(size=5) +
  geom_line() +
  xlab("Decils") +
  ylab("% difference in nº Pk") +
  ylim(-50,25) + ggtitle("CL")  +
  geom_hline(aes(yintercept=0), lwd=1, lty=2) +
  scale_x_discrete(limits=c("DC1","DC2","DC3","DC4","DC5","DC6","DC7","DC8","DC9","DC10")) +
  coord_polar()
```



Third Part - The most important 10+

Followed by this website Beautiful Radar Chart in R using FMSB and GGPlot 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.

Note that, the fmsb radar chart is an R base plot. The ggradar package builds a ggplot spider plot.

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 20. Columns are quantitative variables and rows are individuals:

```r
# Demo data
exam_scores <- data.frame(
  row.names = c("Student.1", "Student.2", "Student.3"),
  Biology = c(7.9, 3.9, 9.4),
  Physics = c(10, 20, 0),
  Maths = c(3.7, 11.5, 2.5),
  Sport = c(8.7, 20, 4),
  English = c(7.9, 7.2, 12.4),
  Geography = c(6.4, 10.5, 6.5),
  Art = c(2.4, 0.2, 9.8),
  Programming = c(0, 0, 20),
  Music = c(20, 20, 20)
)
exam_scores
```

```
##           Biology Physics Maths Sport English Geography Art Programming Music
## Student.1     7.9      10   3.7   8.7     7.9       6.4 2.4           0    20
## Student.2     3.9      20  11.5  20.0     7.2      10.5 0.2           0    20
## Student.3     9.4       0   2.5   4.0    12.4       6.5 9.8          20    20
```

Data preparation

The data should be organized as follow:

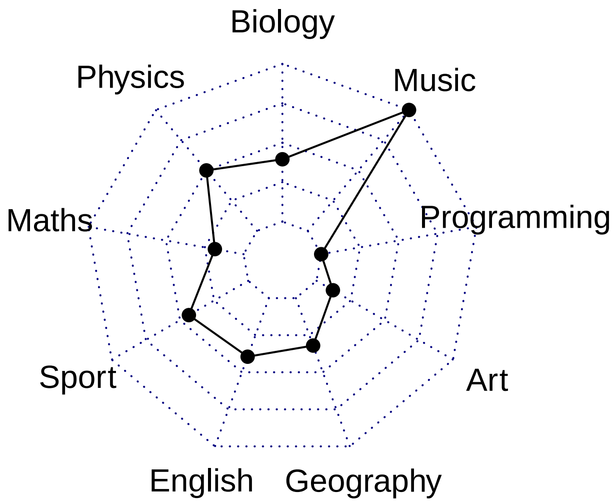
- **The row 1 must contain the maximum values for each variable**
- **The row 2 must contain the minimum values for each variable**
- **Data for cases or individuals should be given starting from row 3**
- **The number of columns or variables must be more than 2.**

```r
# Define the variable ranges: maximum and minimum
max_min <- data.frame(
  Biology = c(20, 0), Physics = c(20, 0), Maths = c(20, 0),
  Sport = c(20, 0), English = c(20, 0), Geography = c(20, 0),
  Art = c(20, 0), Programming = c(20, 0), Music = c(20, 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          20.0      20  20.0  20.0    20.0      20.0 20.0          20    20
## Min           0.0       0   0.0   0.0     0.0       0.0  0.0           0     0
## Student.1     7.9      10   3.7   8.7     7.9       6.4  2.4           0    20
## Student.2     3.9      20  11.5  20.0     7.2      10.5  0.2           0    20
## Student.3     9.4       0   2.5   4.0    12.4       6.5  9.8          20    20
```

Basic radar plot

```r
# Plot the data for student 1
library(fmsb)
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'`
- **Grid options:**
- `   *cglcol: line color*`
- `   *cglty: line type*`
- `   *cglwd: line width*`

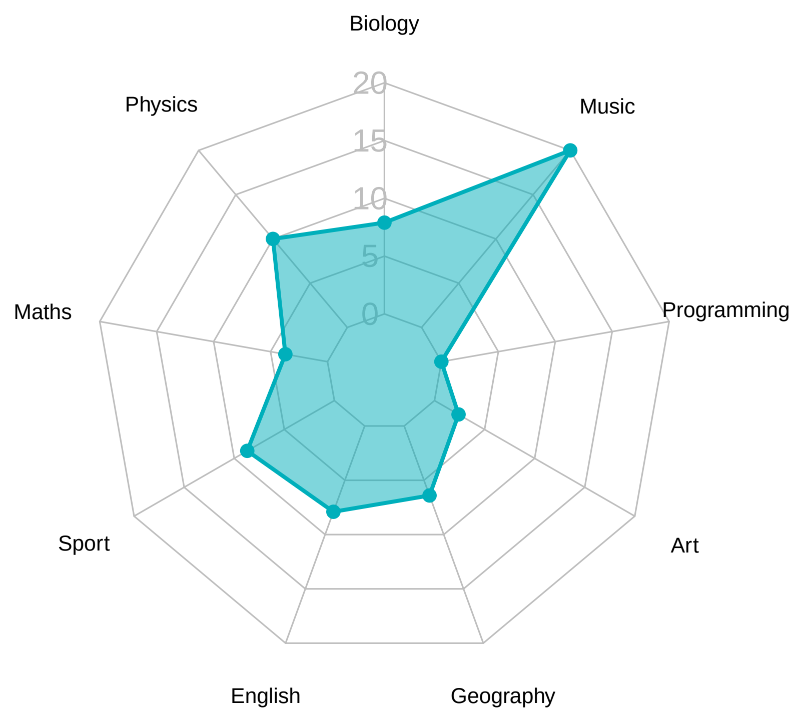
- **Axis options:**
- `*axislabcol: color of axis label and numbers. Default is "blue".*`
- `*caxislabels: Character vector to be used as labels on the center axis.*`

Helper function to produce a beautiful radar chart:

```r
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.

```r
# Reduce plot margin using par()
op <- par(mar = c(1, 2, 2, 1))
create_beautiful_radarchart(student1_data, caxislabels = c(0, 5, 10, 15, 20))
```

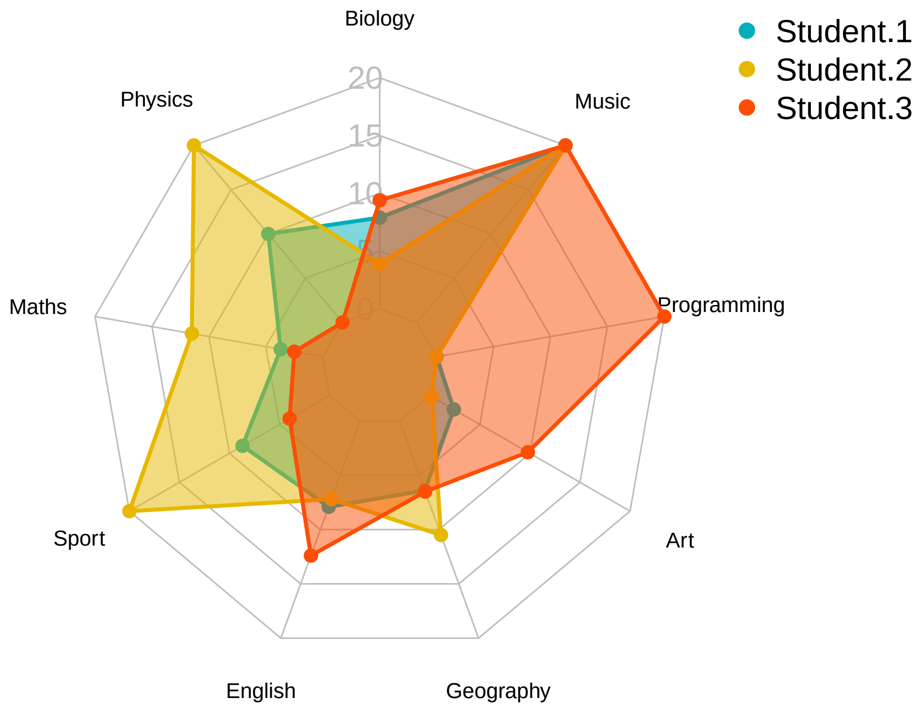


```
par(op)
```

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, 5, 10, 15, 20),
  color = c("#00AFBB", "#E7B800", "#FC4E07")
)
# Add an horizontal legend
legend(
  x = "topright", legend = rownames(df[-c(1,2),]), horiz = FALSE,
  bty = "n", pch = 20 , col = c("#00AFBB", "#E7B800", "#FC4E07"),
  text.col = "black", cex = 1, pt.cex = 1.5
)
```



```
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.

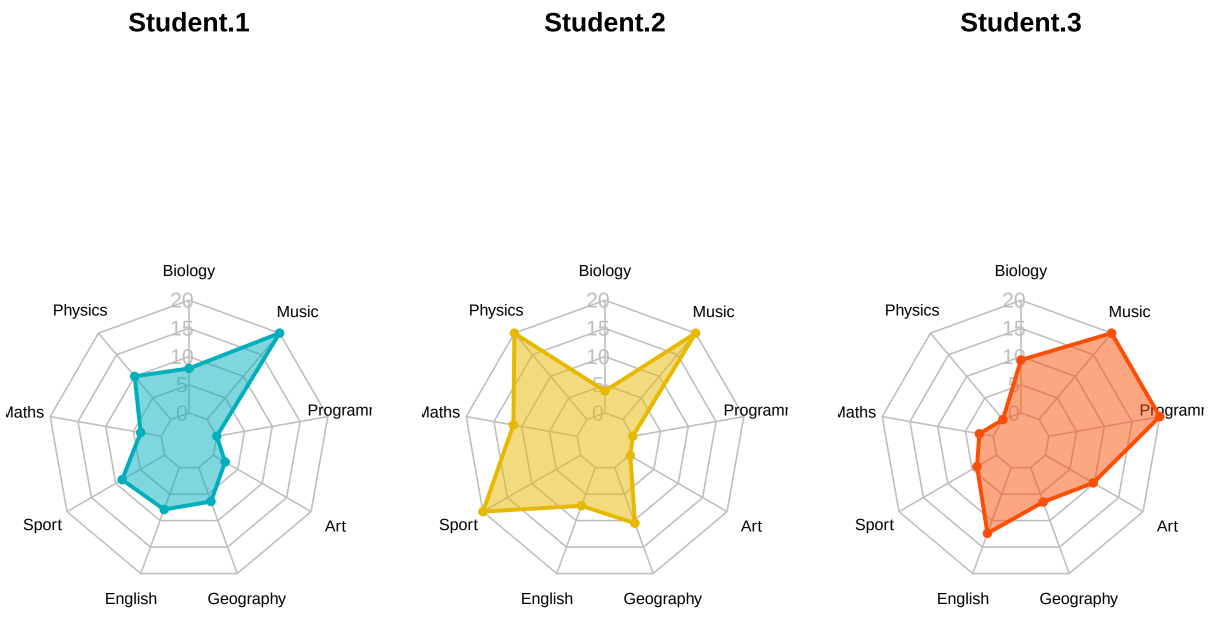
c(x_coordinate, y_coordinate): Allows you to specify custom coordinates for the legend. You can provide the x and y coordinates as numeric values.

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, 5, 10, 15, 20),
    color = colors[i], title = titles[i]
  )
}
```



```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.

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

set.seed(123)
df <- as.data.frame(
  matrix(sample(2:20 , 90 , replace = TRUE),
         ncol=9, byrow = TRUE)
)

# In this example the matrix we build up has a vote (variable) between 2 and 20,
# 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  16 20 15  4 11 19 12  6 15
## 2   6 20 10  4  9  8 11 10 20
## 3   5 15 18 12  8 13 16 11 14
## 4   8 10 10 11  8  7  3  6  9
## 5  13 14 19  2  7 16 10 16 17
## 6   7 12  9  8 17 18 19 18  3
## 7   5 14  6 20 15  4  9 17 13
## 8  15  4 15  8  4 16  6  9 20
## 9  11 19 11 13  3 11 13 15 18
## 10 15  4  9 15 20 16 18 12  8
```

```
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      16      20    15     4      11        19  12           6    15
## Student.2       6      20    10     4       9         8  11          10    20
## Student.3       5      15    18    12       8        13  16          11    14
## Student.4       8      10    10    11       8         7   3           6     9
## Student.5      13      14    19     2       7        16  10          16    17
## Student.6       7      12     9     8      17        18  19          18     3
```

```
# Give the name based on a root = Student and adding the row number from 1 to
# end (10), Same is giving the name of the subject per every column
```

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.

```
library(scales)
df_scaled <- round(apply(df, 2, scales::rescale), 2)
df_scaled <- as.data.frame(df_scaled)
head(df_scaled)
```

```
##           Biology Physics Maths Sport English Geography  Art Programming Music
## Student.1    1.00    1.00  0.69  0.11    0.47      1.00 0.56        0.00  0.71
## Student.2    0.09    1.00  0.31  0.11    0.35      0.27 0.50        0.33  1.00
## Student.3    0.00    0.69  0.92  0.56    0.29      0.60 0.81        0.42  0.65
## Student.4    0.27    0.38  0.31  0.50    0.29      0.20 0.00        0.00  0.35
## Student.5    0.73    0.62  1.00  0.00    0.24      0.80 0.44        0.83  0.82
## Student.6    0.18    0.50  0.23  0.33    0.82      0.93 1.00        1.00  0.00
```

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:

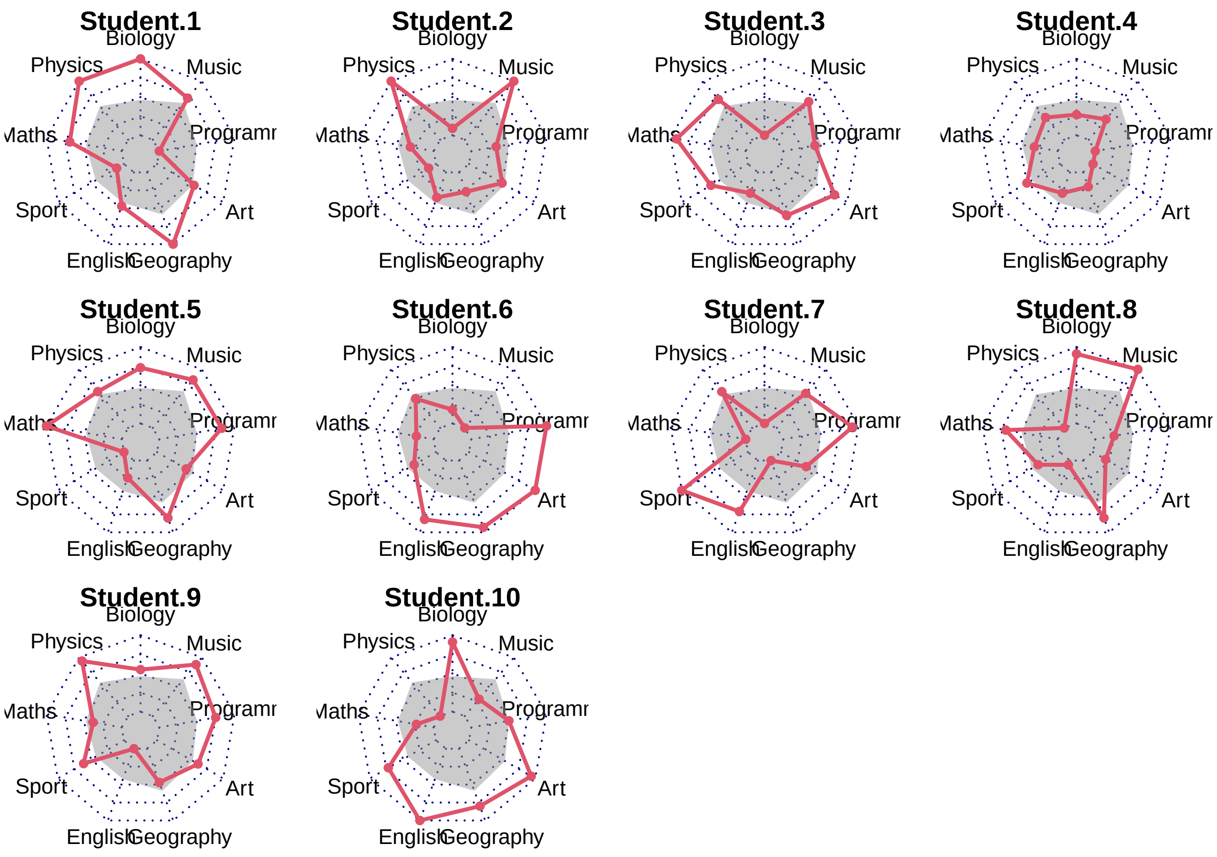
```
# Variables summary
# Get the minimum and the max of every column
col_max <- apply(df_scaled, 2, max)
col_min <- apply(df_scaled, 2, min)
# Calculate the average profile
col_mean <- apply(df_scaled, 2, mean)
# Put together the summary of columns
col_summary <- t(data.frame(Max = col_max, Min = col_min, 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        1.00 1.000
## Min         0.000   0.000 0.000 0.000   0.000     0.000 0.000        0.00 0.000
## Average     0.464   0.575 0.476 0.427   0.423     0.587 0.544        0.50 0.629
## Student.1   1.000   1.000 0.690 0.110   0.470     1.000 0.560        0.00 0.710
## Student.2   0.090   1.000 0.310 0.110   0.350     0.270 0.500        0.33 1.000
## Student.3   0.000   0.690 0.920 0.560   0.290     0.600 0.810        0.42 0.650
```

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 `i`th 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.

==

# Forth Part

ggplot radar chart using the ggradar R package

Prerequisites

```
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.

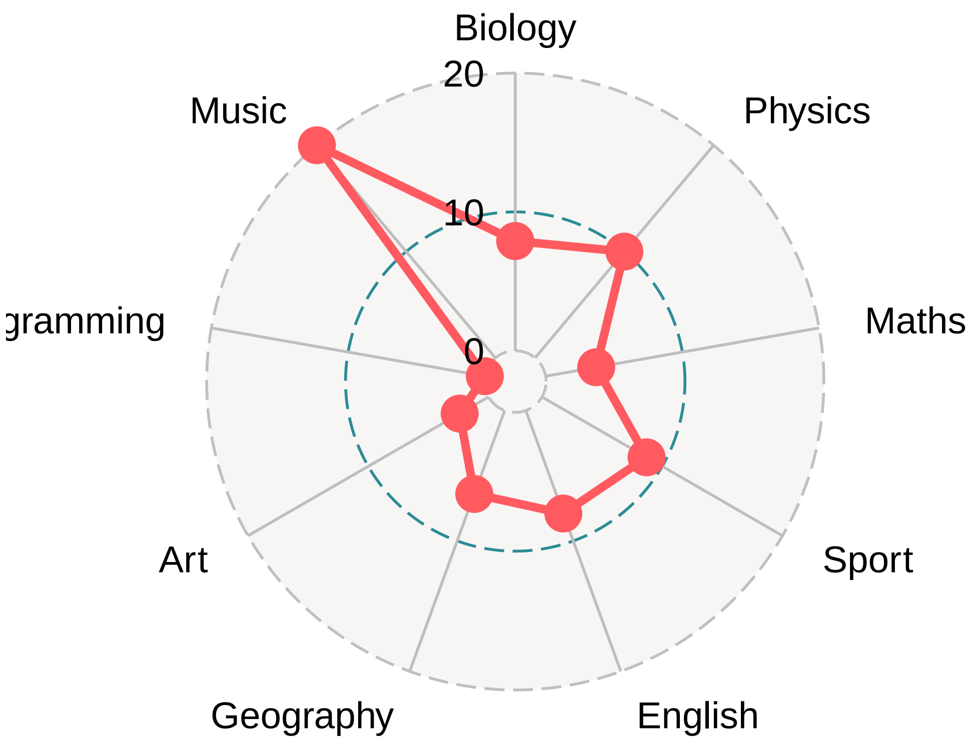
For example, you can rescale the variables to have a minimum of 0 and a maximum of 1 using the function `rescale()` [scales package]. We'll describe this method in the next sections.

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

```
##       group Biology Physics Maths Sport English Geography Art Programming Music
## 1 Student.1     7.9      10   3.7   8.7     7.9       6.4 2.4           0    20
## 2 Student.2     3.9      20  11.5  20.0     7.2      10.5 0.2           0    20
## 3 Student.3     9.4       0   2.5   4.0    12.4       6.5 9.8          20    20
```

Basic radar plot

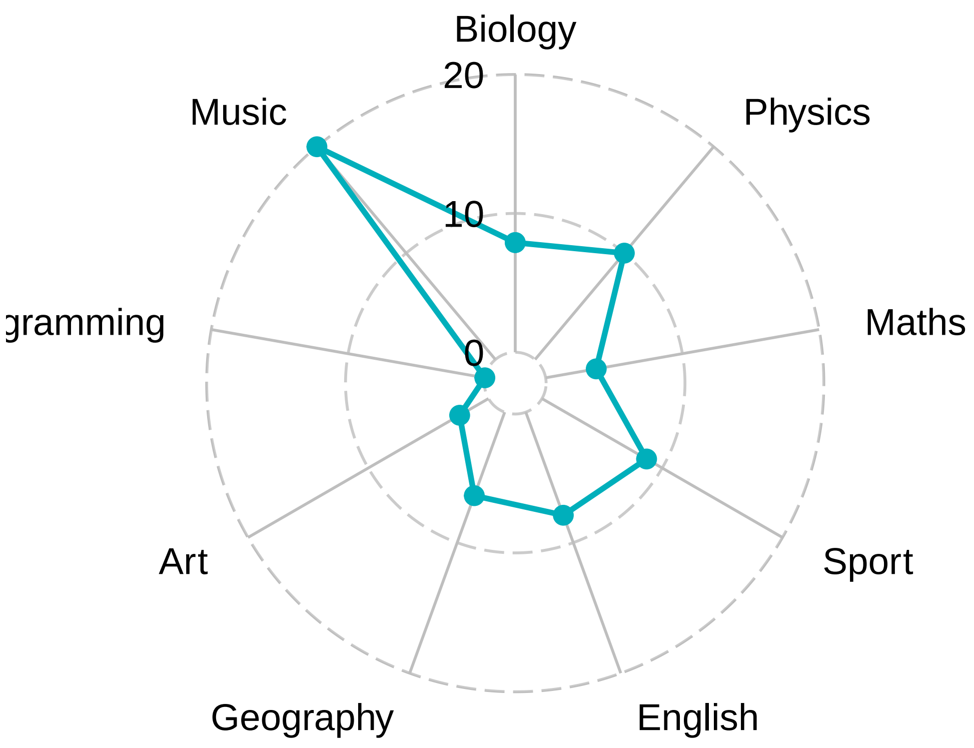
```
# Plotting student 1
ggradar(
  df[1, ],
  values.radar = c("0", "10", "20"),
  grid.min = 0, grid.mid = 10, grid.max = 20
)
```



Customize radar charts

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

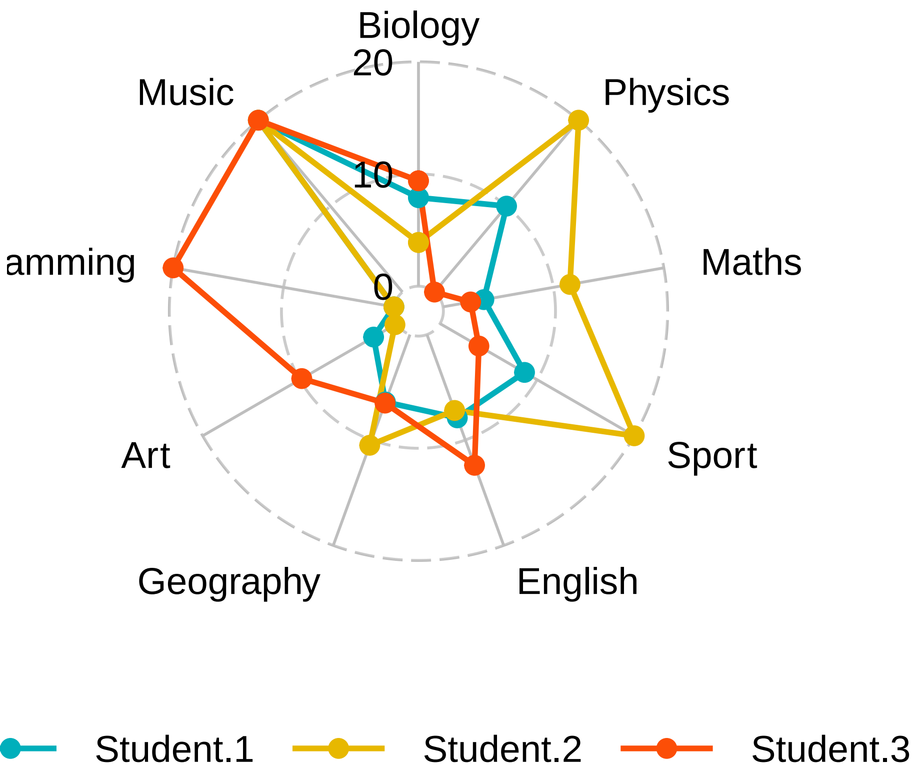
```r
ggradar(
  df[1, ],
  values.radar = c("0", "10", "20"),
  grid.min = 0, grid.mid = 10, grid.max = 20,
  # Polygons
  group.line.width = 1,
  group.point.size = 3,
  group.colours = "#00AFBB",
  # Background and grid lines
  background.circle.colour = "white",
  gridline.mid.colour = "grey"
  )
```



Radar chart with multiple individuals or groups

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

```r
ggradar(
  df,
  values.radar = c("0", "10", "20"),
  grid.min = 0, grid.mid = 10, grid.max = 20,
  # 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(tidyverse)
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       7.9       3.9       9.4
## 2     Physics      10.0      20.0       0.0
## 3       Maths       3.7      11.5       2.5
## 4       Sport       8.7      20.0       4.0
## 5     English       7.9       7.2      12.4
## 6   Geography       6.4      10.5       6.5
## 7         Art       2.4       0.2       9.8
## 8 Programming       0.0       0.0      20.0
## 9       Music      20.0      20.0      20.0
```

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.

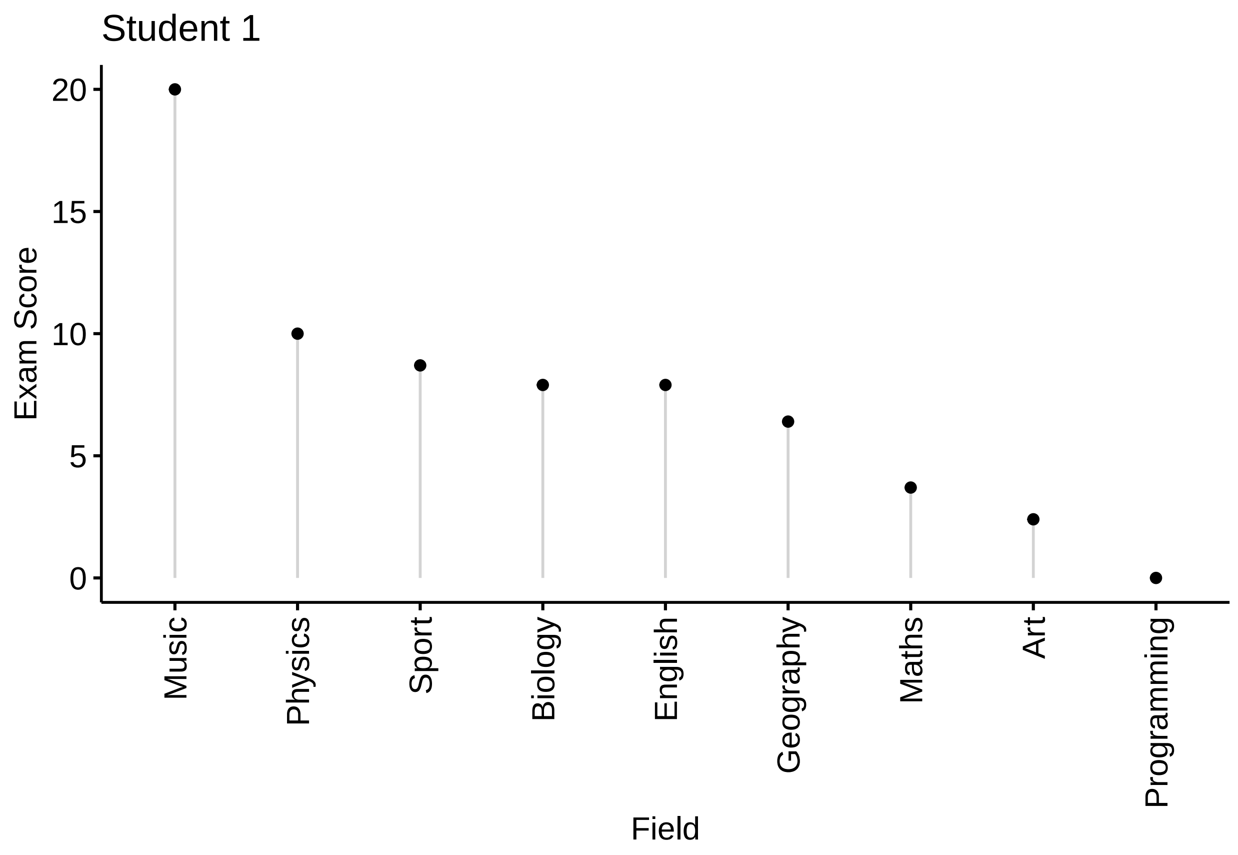
%>%: This is the pipe operator, which allows you to chain multiple operations together. It takes the output from the previous operation and passes it as the first argument to the next function call.

- **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

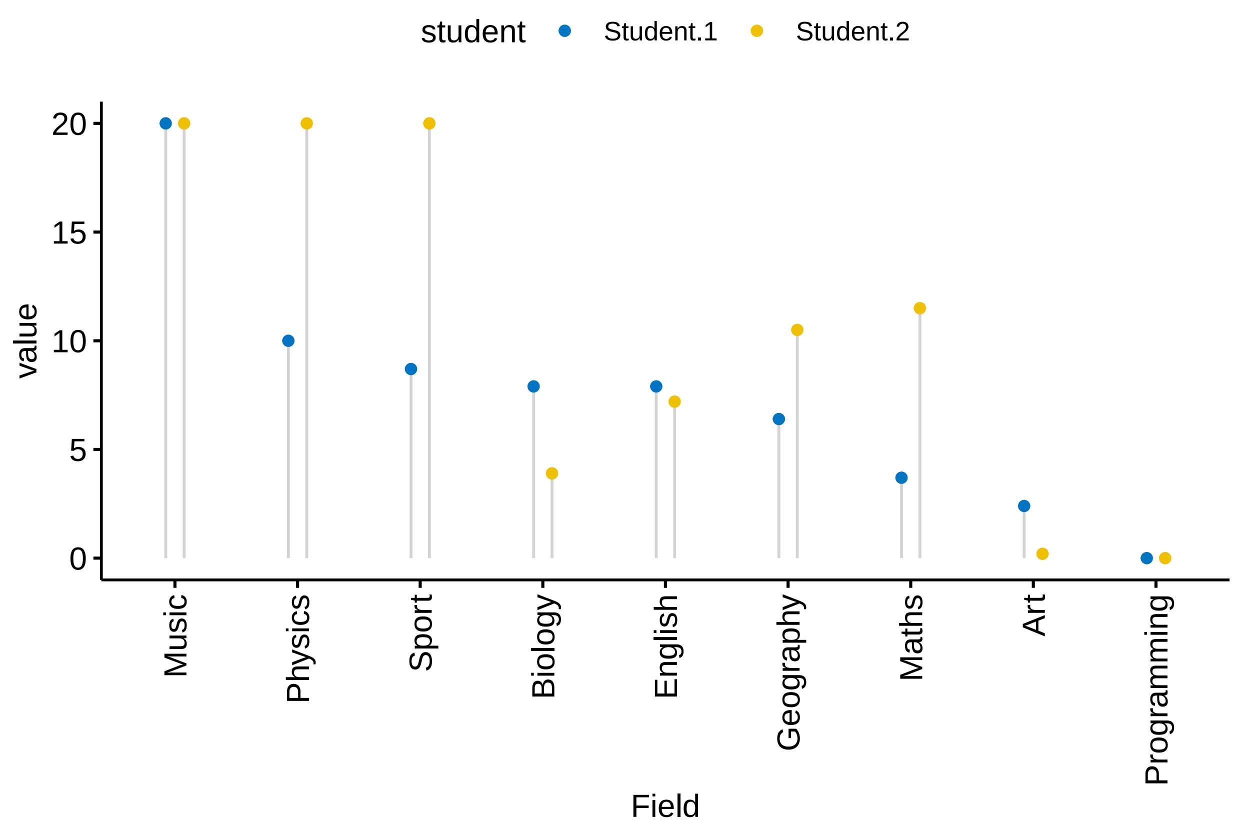
Data preparation:

```
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   7.9
## 2 Biology Student.2   3.9
## 3 Physics Student.1  10
## 4 Physics Student.2  20
## 5 Maths   Student.1   3.7
## 6 Maths   Student.2  11.5
```

Plot creation

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



Displaying multiple individuals

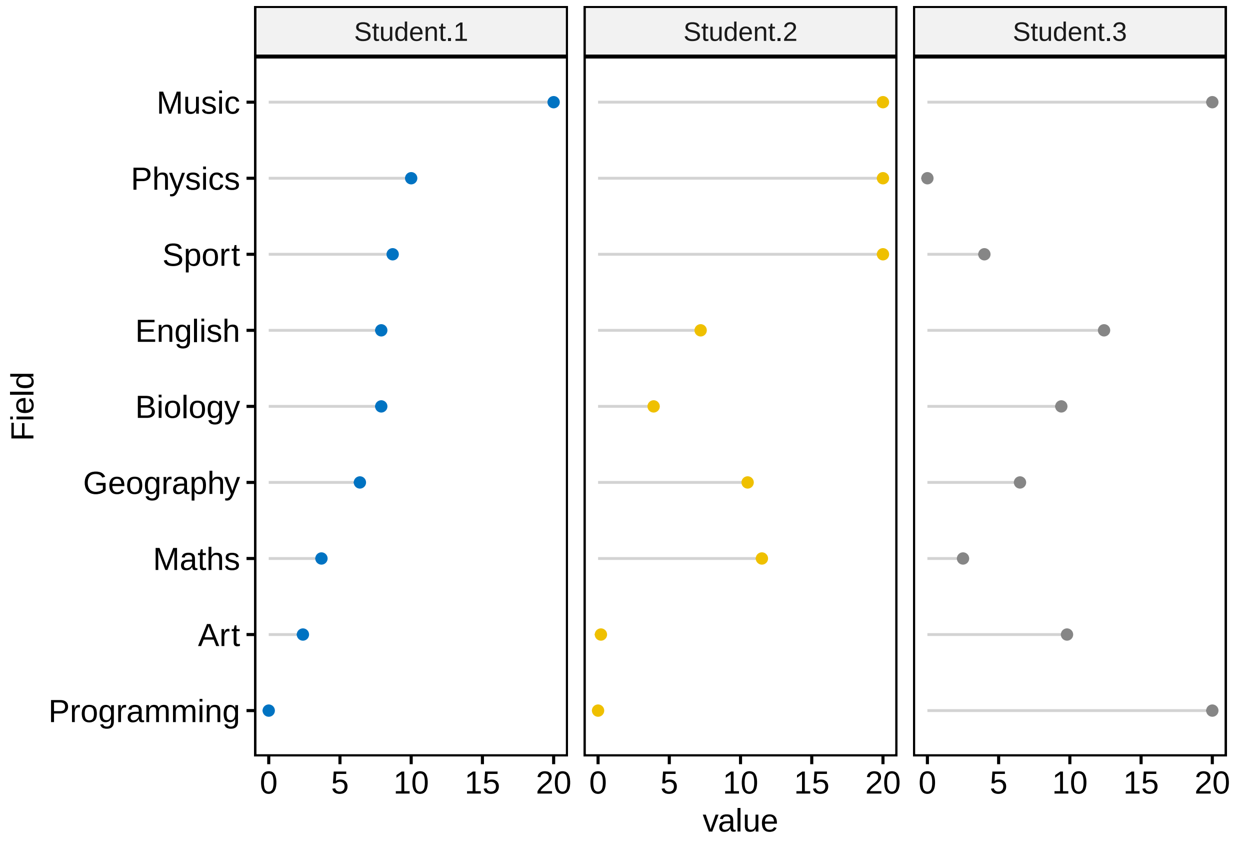
Data preparation:

```r
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   7.9
## 2 Biology Student.2   3.9
## 3 Biology Student.3   9.4
## 4 Physics Student.1  10
## 5 Physics Student.2  20
## 6 Physics Student.3   0
```

Plot creation

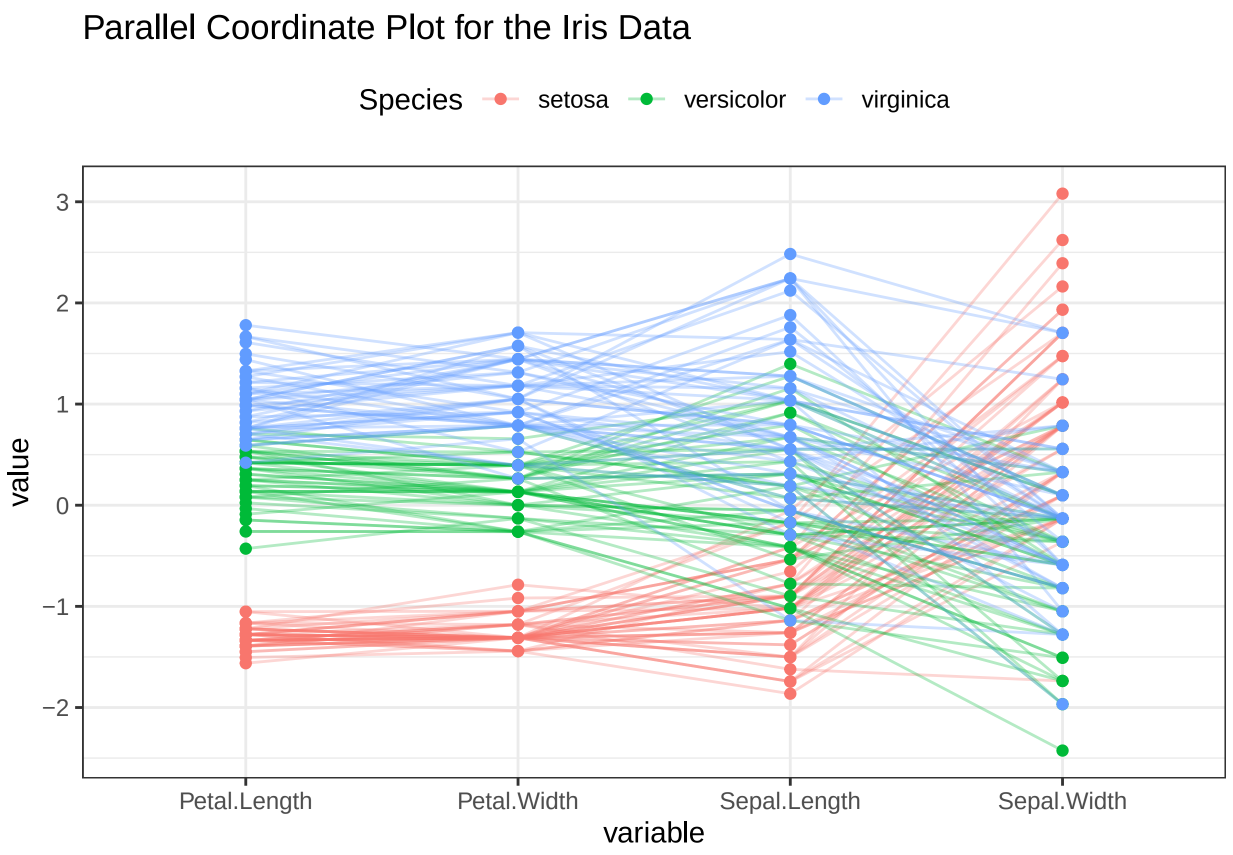
```
ggdotchart(
  df4, x = "Field", y = "value",
  group = "student", color = "student", palette = "jco",
  add = "segment", position = position_dodge(0.3),
  sorting = "descending", facet.by = "student",
  rotate = TRUE, legend = "none"
  )
```



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.

. **END** .

```
sessionInfo()
```