Learning Radar Charts in R: A Step-by-Step Guide with Examples

Authored by

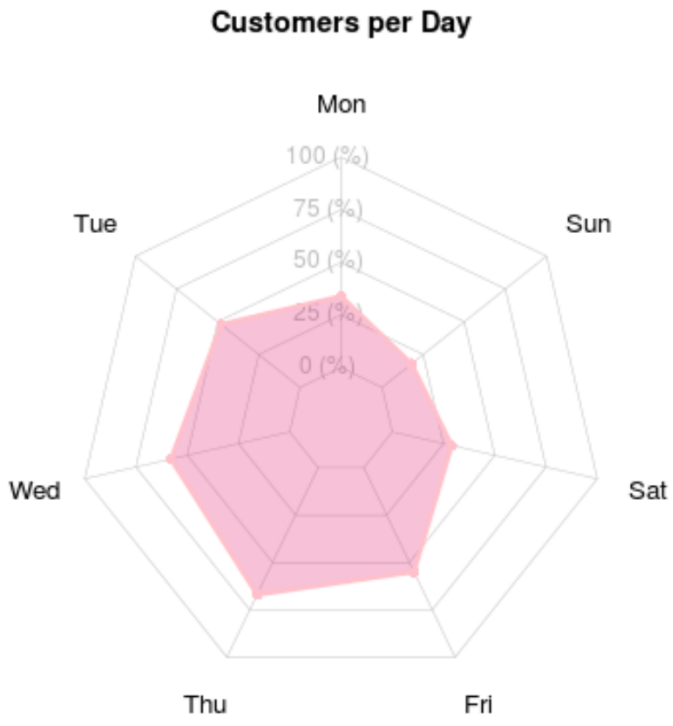
Mohammed looti

November 7, 2025

The **radar chart**, often referred to as a **spider chart** or a star plot, is an exceptionally versatile graphical technique widely employed in **data visualization**. This visualization excels at comparing multiple entities across three or more quantitative variables simultaneously. It achieves this by plotting values on distinct axes that radiate outward from a shared central point, generating a unique polygon whose shape offers immediate intuitive insight into relative performance or characteristics.

For statistical computing and generating highly customized plots, the **R** environment stands out as a leading platform. To efficiently produce professional-grade radar charts, data practitioners rely heavily on the specialized **fmsb** package. This package provides the essential functions needed to translate complex, multivariate data tables into the distinctive and visually informative polygon structure of the spider chart.

This comprehensive tutorial serves as a step-by-step guide, detailing the crucial data preparation steps, the necessary library installations, and the exact code execution required to generate both a fundamental and an extensively customized radar chart within **R**. By the end of this guide, you will be equipped to create sophisticated visualizations mirroring the example provided below.



Deconstructing the Radar Chart Structure

A **radar chart** is fundamentally organized around a central origin point, from which a series of

spokes or axes extend radially. Each of these axes is dedicated to representing one specific quantitative variable that is being measured. The magnitude of the variable's value for a given observation is reflected by the distance of the plotted point from the center along its corresponding axis. Effectively, the further a point is from the center, the higher the score or value for that variable.

When the data points on all axes are connected, they form a closed, non-intersecting polygon. It is the overall shape and area of this polygon that facilitates a quick, holistic comparison of an entity's performance across multiple dimensions. For instance, in a corporate performance review, a polygon that is large and relatively uniform in shape suggests strong, balanced scores across all criteria, whereas sharp inward indentations clearly flag specific areas of underperformance or weakness.

Crucial to the effective use of the **spider chart** is the appropriate normalization or scaling of all input variables. This step is necessary even when variables are measured in different units. By establishing consistent minimum and maximum boundaries for every axis, we ensure that the visual comparison is fair and prevent variables that inherently possess larger numeric ranges from unfairly dominating the resulting visualization.

When implementing this visualization technique in **R** using the **fmsb** package, the precise structure of the input data is paramount, as it directly dictates the geometry of the final plot. Therefore, understanding and adhering to the package's unique data formatting prerequisites is the single most critical technical requirement for generating an accurate and meaningful visualization.

Essential Prerequisites and Data Formatting in R

To successfully invoke the `radarchart()` function provided by the **fmsb** package, the input data must conform to a very specific structure. Unlike many standard **R** plotting functions, this package requires the input to be formatted as a **data frame** where the first two rows are explicitly reserved for defining the plotting boundaries.

These initial two rows of the **data frame** are indispensable, as they establish the visual scale for the entire chart. The plotting function uses the first row to determine the absolute maximum value (the ceiling) for each axis, which defines the outermost perimeter of the grid. Conversely, the second row specifies the absolute minimum value (the floor), which dictates the innermost point of the scale. This mandatory scaling mechanism ensures that all subsequent observational data points are correctly contained and plotted within a fixed, standardized visual range.

The input **data frame** must strictly adhere to the following structural requirements for successful plotting:

Each variable intended to be plotted on a radial axis must correspond to its own dedicated column within the **data frame**.
The **first row** must contain the maximum permissible value for every corresponding column, establishing the peak of the scale.
The **second row** must contain the minimum permissible value for every corresponding column, establishing the base of the scale.
All subsequent rows, beginning from Row 3 onward, must contain the actual observational data points that the **radar chart** will visualize.

To illustrate this mandatory structure, we will construct a sample **data frame** tracking the number of customers visiting a shop across the seven days of the week. We standardize the scale by establishing a maximum customer count of 100 and a minimum of 0 for every day. The following code snippet demonstrates the creation and structure of this required input:

```
#create data
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
```

Step-by-Step: Generating the Basic Radar Chart

With the data successfully structured according to the **fmsb** package's requirements, the initial process of generating the **radar chart** becomes highly efficient. Assuming the **fmsb** package has been installed (if not, the command `install.packages("fmsb")` should be executed), the immediate next step is to load the necessary library into the active R session using the `library()` function.

The core visualization functionality is encapsulated within the `radarchart()` function. By simply

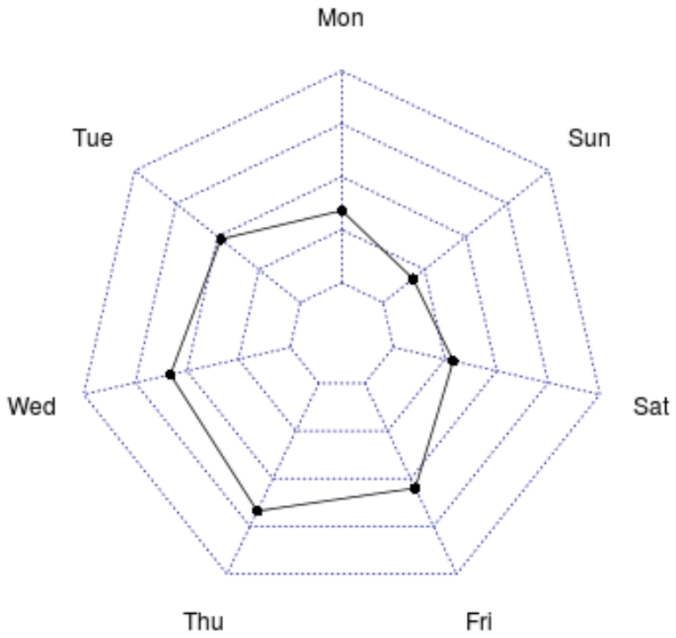
passing the prepared **data frame** (named `df` in our example) to this function, **R** automatically handles the complex tasks of rendering the axes, drawing the concentric grid lines, and plotting the polygon. The polygon's shape is determined by the values in the third row, positioned relative to the standardized boundaries established by the first two rows.

Executing the following minimal code will produce the default **radar chart**. This basic plot is invaluable as a verification step, confirming that the data structure is correctly interpreted by the package and providing immediate visual insight into the multivariate relationships within the dataset.

library(fmsb)

radarchart(df)

The resulting default visualization clearly illustrates the pattern of customer traffic throughout the week. The polygon expands significantly outward along the Thursday axis, unambiguously indicating peak customer numbers during the mid-week period. Conversely, the axes representing the weekend (Saturday and Sunday) remain visually closer to the chart's center, highlighting significantly lower activity.



Implementing Advanced Customization Parameters

While the default output offers immediate functional insight, transforming a basic plot into a professional, publication-ready graphic necessitates thoughtful customization. The `radarchart()`

function within **fmsb** is designed with extensive flexibility, providing numerous arguments that allow granular control over the plot's aesthetics, including color schemes, line weights, grid styles, and label sizing.

Customization is particularly vital when integrating **data visualization** into formal reports, academic papers, or interactive dashboards, where visual hierarchy, clarity, and adherence to specific branding are critical. By strategically manipulating these parameters, developers can ensure that the plotted data polygon receives the appropriate visual emphasis against the backdrop of the reference grid.

Below is a detailed breakdown of the primary arguments available for precisely tailoring the appearance of your **radar chart**:

pcol: Determines the color used for the perimeter line of the plotted data polygon.
pfcol: Sets the fill color for the area enclosed by the polygon. Crucially, the `rgb()` function should be used here to apply transparency (opacity) to prevent the fill from hiding grid lines.
plwd: Specifies the line width (thickness) of the polygon's perimeter boundary.
cglcol: Controls the color of the concentric grid lines (the "net" radiating from the center).
cglty: Specifies the line type (e.g., solid, dotted, or dashed) used for the concentric grid lines.
axislabcol: Defines the color for the numerical labels that mark the specific concentric axis levels (e.g., 0, 50, 100).
caxislabels: Allows the user to supply a custom character vector of labels to replace the default numeric scale values on the concentric axes.
cglwd: Adjusts the line width of the concentric grid lines, independent of the polygon line width.
vlcex: Controls the character expansion factor (size) for the variable labels (e.g., Mon, Tue, Wed) that are positioned at the outermost tips of the radial axes.

Visualizing the Customized Result

To showcase the power of these aesthetic controls, we will apply a robust set of changes to our customer traffic plot. Our objective is to create a visually striking chart featuring a prominent polygon outline and a semi-transparent fill, which significantly improves both visual appeal and data readability compared to the monochromatic default plot.

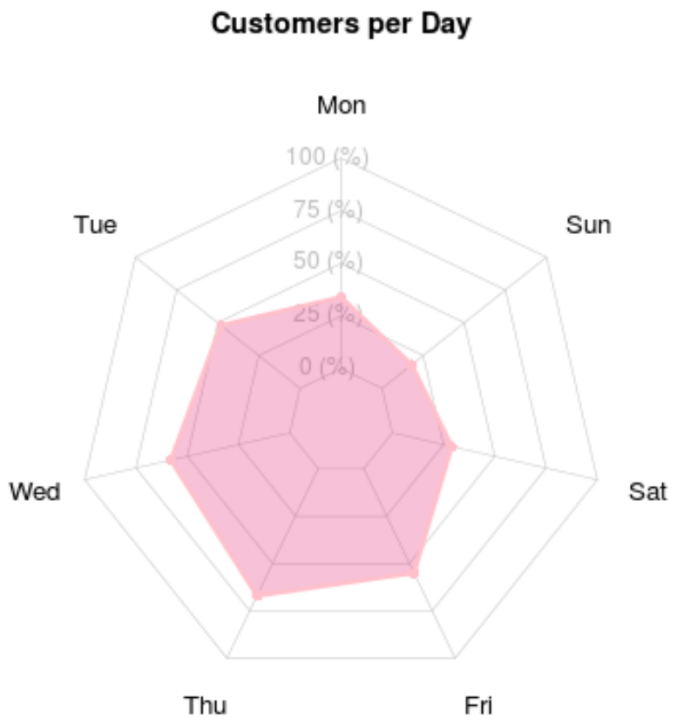
We implement transparency using the **R** color command `pfcol=rgb(0.9, 0.2, 0.5, 0.3)`. Here, the final value of `0.3` assigns 30% opacity to the pink-toned fill color, ensuring that the grid lines beneath remain visible. We also substantially increase the line width using `plwd=3` to heavily emphasize the polygon boundary and add a descriptive title to clearly articulate the chart's content.

Additionally, the argument `axistype=1` is employed to suppress the default numerical axis labels. This strategic choice directs the viewer's attention primarily to the relative shape and size of the

polygon against the grid, rather than focusing on the explicit numerical values of the underlying scale.

```
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'
)
```

The resulting customized **radar chart** is a professional-quality visualization that effectively communicates the distribution pattern of customer traffic over the week. This example clearly demonstrates the robust capabilities of the **fmsb** package in generating tailored statistical graphics essential for high-impact communication.



Conclusion and Further Exploration

Mastering the creation of a **radar chart** in **R** primarily relies on correctly formatting the input **data frame** for the **fmsb** package. The inclusion of the mandatory maximum and minimum boundary rows is the foundational technical requirement that ensures accurate scaling and meaningful interpretation of the multivariate data.

By effectively leveraging the extensive set of customization parameters available within the `radarchart()` function, users can elevate functional default plots into highly engaging and informative visual tools. The ability to precisely control colors, transparency, line styles, and labeling is absolutely critical for producing high-impact **data visualization** suitable for dissemination across academic, corporate, or public sectors.

To further expand your skills in creating sophisticated statistical graphics within the **R** environment, we recommend exploring these related visualization techniques:

How to Create Heatmaps in R
How to Create a Lollipop Chart in R
How to Create a Population Pyramid in R