

# How to Create Lollipop Charts in R with ggplot2

Authored by  
**stats writer**

March 3, 2026

## RECOMMENDED CITATION

stats writer (2026). *How to Create Lollipop Charts in R with ggplot2*. PSYCHOLOGICAL SCALES. Retrieved from <https://scales.arabpsychology.com/?p=133608>

The **lollipop chart** has emerged as a sophisticated alternative to the traditional **bar chart**, offering a cleaner and more modern aesthetic for **data visualization**. In the **R programming language**, this chart is essentially a composite graphic that layers a **scatter plot** over a series of line segments. By utilizing the **ggplot2** package, users can create these visualizations to highlight the relationships between **categorical variables** and **numerical variables**. The primary advantage of this format is its ability to reduce visual clutter, allowing the reader to focus on the exact **data points** represented by the "lollipops" rather than the bulk of a solid bar.

To construct a **lollipop chart**, one typically maps the categories to one axis and the quantitative values to the other. The **Grammar of Graphics** framework, which serves as the foundation for **ggplot2**, allows for the independent customization of the lines (the sticks) and the points (the heads). This flexibility is particularly useful when dealing with large datasets where standard bars might overlap or appear too dense. By adjusting the **size** and **color** parameters within the **geom\_point** function, a developer can emphasize specific categories or create a **color scheme** that aligns with professional branding or publication standards.

Furthermore, the integration of **labels** and **titles** using the **labs** and **ggtitle** functions ensures that the visualization is not only aesthetically pleasing but also informative. This method of visualization is highly valued in **exploratory data analysis** because it facilitates rapid comparisons across multiple categories. Whether you are analyzing sales figures, scientific observations, or performance metrics, mastering the **lollipop chart** in **R** will significantly enhance your ability to communicate complex data insights clearly and effectively.

## Comprehensive Guide: How to Create a Lollipop Chart in R

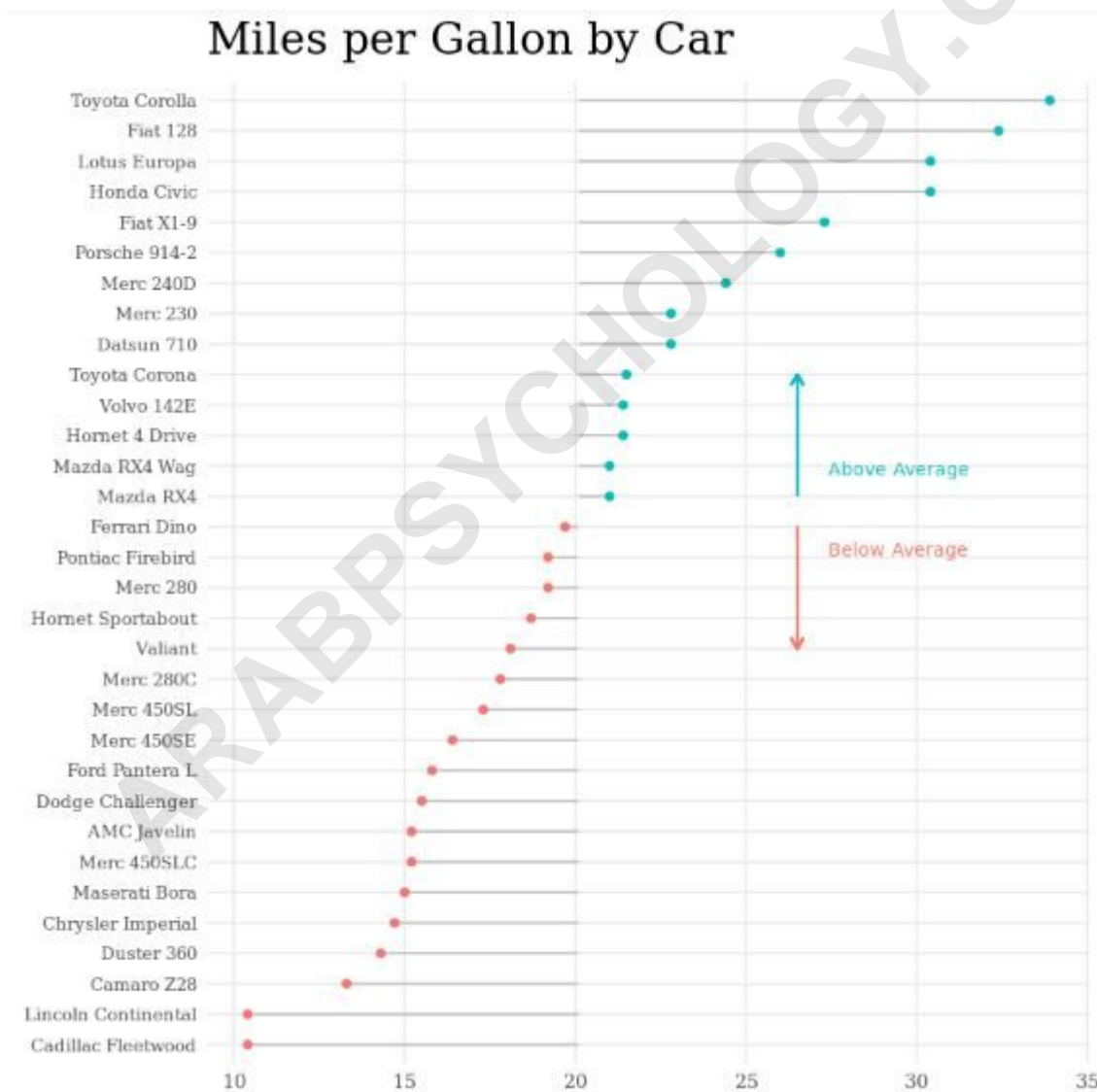
### The Conceptual Foundation of Lollipop Charts

Functioning as a minimalist variation of the standard **bar chart**, a **lollipop chart** is exceptionally useful for comparing the **quantitative values** associated with a **categorical variable**. Rather than employing the thick, often distracting bars found in traditional graphs, this visualization uses thin lines capped with circles. This design choice minimizes the "ink-to-data" ratio, a concept popularized by **Edward Tufte**, which suggests that a chart is most effective when it uses the least amount of ink possible to convey its message. By stripping away the visual weight of the bars, the **lollipop chart** draws the viewer's eye directly to the **data point** at the end of the segment.

In addition to its functional benefits, many analysts prefer the **lollipop chart** for its **aesthetic appeal**. In a world saturated with standard **Excel**-style graphics, the **lollipop chart** stands out as a more refined and professional option. It is particularly effective in **dashboards** and **infographics**

where space is at a premium and clarity is paramount. The thin lines allow for more whitespace, which can make a dense **plot** feel more approachable and easier to navigate for non-technical audiences. Furthermore, the **ggplot2** library in **R** makes the transition from a bar chart to a lollipop chart nearly seamless.

In the following sections of this tutorial, we will explore the technical implementation of these charts. We will cover everything from basic construction to advanced **theming** and **data manipulation**. By the end of this guide, you will be proficient in generating high-quality visualizations that are ready for inclusion in academic papers or business reports. We will walk through the necessary steps to create the following **lollipop chart**, ensuring you understand the logic behind each line of **code**.



## Example Walkthrough: Utilizing the `mtcars` Dataset

For the practical portion of this tutorial, we will utilize the `mtcars` dataset, which is a classic resource built directly into **R**. This dataset comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973-74 models). It is a staple in **data science** education because it provides a clean, well-structured environment for practicing **statistical programming**. By using a familiar dataset, we can focus entirely on the mechanics of the **lollipop chart** without being distracted by complex **data cleaning** tasks.

Before we begin plotting, it is always a best practice to inspect the **data structure**. This allows us to identify the names of the columns we wish to use and understand the **data types** involved. In our case, we will be focusing on the `mpg` (miles per gallon) variable as our quantitative measure and the car names as our categorical labels. Understanding the range and distribution of the `mpg` values will help us later when we decide on the appropriate **axis** limits and **baselines** for our chart.

The following code snippet demonstrates how to view the first few rows of the `mtcars` dataset using the `head()` function. This is a crucial step in any **data analysis** workflow, as it confirms that the data has loaded correctly and provides a snapshot of the variables available for **visualization**. As you can see in the output, the names of the cars are currently stored as **row names** rather than a dedicated column, a detail we will need to address in the subsequent steps.

```
#view first six rows of mtcars  
head(mtcars)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	car
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4	Mazda RX4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4	Mazda RX4 Wag
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1	Datsun 710
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1	Hornet 4 Drive
Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2	Hornet Sportabout
Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1	Valiant

## Constructing a Basic Lollipop Chart in R

The initial step in creating our visualization is to prepare the **data frame** so that **ggplot2** can easily interpret the variables. Since the car models are currently stored as **row names**, we must transform them into an explicit column. This is a common requirement in **tidy data** workflows, where every variable must have its own column. Once the **categorical variable** is established, we can proceed to load the **ggplot2** library, which is the industry standard for **data visualization** in the **R** ecosystem.

The construction of the chart itself relies on two primary **geometric layers**. First, we use **geom\_segment** to draw the "stick" of the lollipop. This function requires four **aesthetic mappings**: the starting and ending coordinates for both the x and y axes. In this instance, we set the starting x-value to 0 and the ending x-value to the **mpg** value. Second, we apply **geom\_point** to place a circle at the end of each segment. This layered approach is a perfect example of the power and flexibility inherent in the **Grammar of Graphics**.

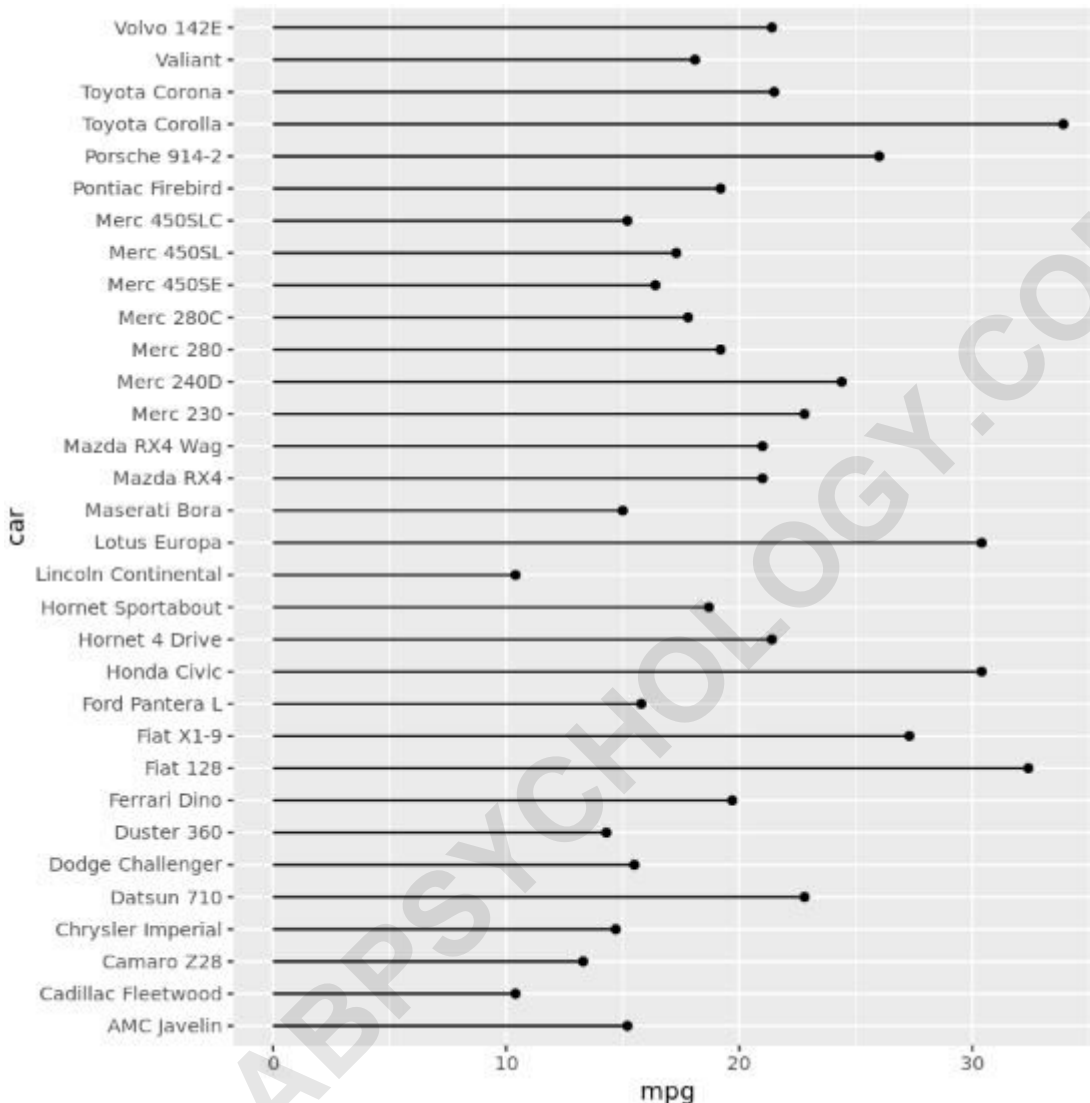
By mapping the **car** variable to the **y-axis** and the **mpg** variable to the **x-axis**, we create a horizontal orientation. This is often preferred for **categorical data** with long labels, as it prevents the text from overlapping and makes the chart significantly easier to read. The following **code** block illustrates the full process of defining the data, loading the necessary **package**, and executing the **plot** commands to generate a functional, albeit basic, **lollipop chart**.

```
#create new column for car names
mtcars$car <- row.names(mtcars)

#load ggplot2 library
library(ggplot2)

#create lollipop chart
ggplot(mtcars, aes(x = mpg, y = car)) +
```

```
geom_segment(aes(x = 0, y = car, xend = mpg, yend = car)) +
geom_point()
```



## Improving Readability with Labels and Customization

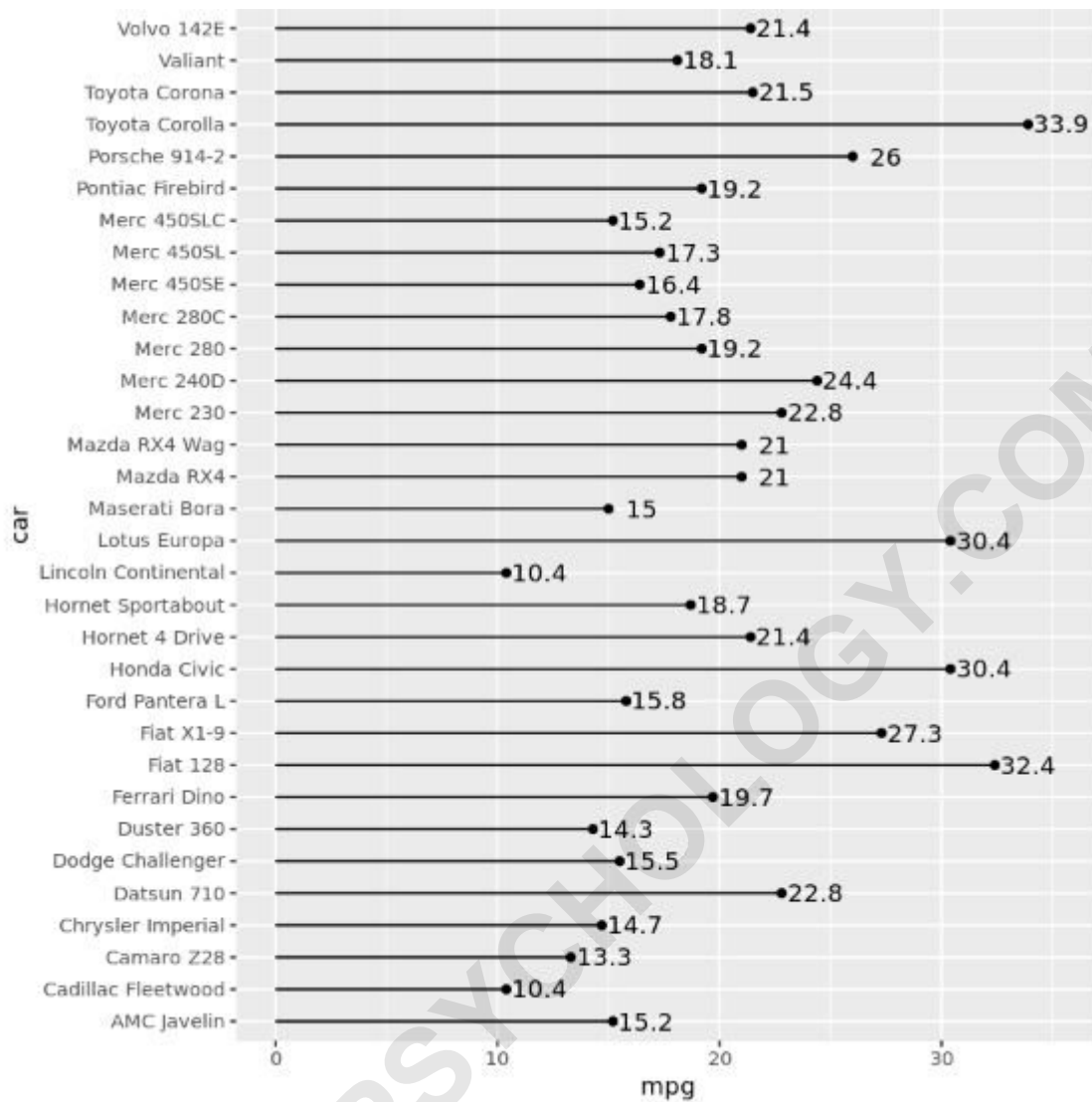
While a basic chart conveys the general trend of the data, adding specific **labels** can provide the precise **numerical values** that a reader might need. In **ggplot2**, this is achieved by adding a **label** aesthetic within the **aes()** function and then calling **geom\_text**. To prevent the text from overlapping with the point, we use the **nudge\_x** parameter, which shifts the label a specified distance along the horizontal axis. This subtle adjustment ensures that the **visualization** remains clean while offering a higher level of detail.

An alternative labeling strategy involves placing the **text** directly inside the **point** of the lollipop.

This is an excellent technique for creating a very compact and modern look. To accomplish this, we must increase the **size** of the points so they can accommodate the numbers and change the **color** of the text to a high-contrast shade like white. This method is particularly effective when the range of values is small and the **font size** can remain legible without overwhelming the **chart**.

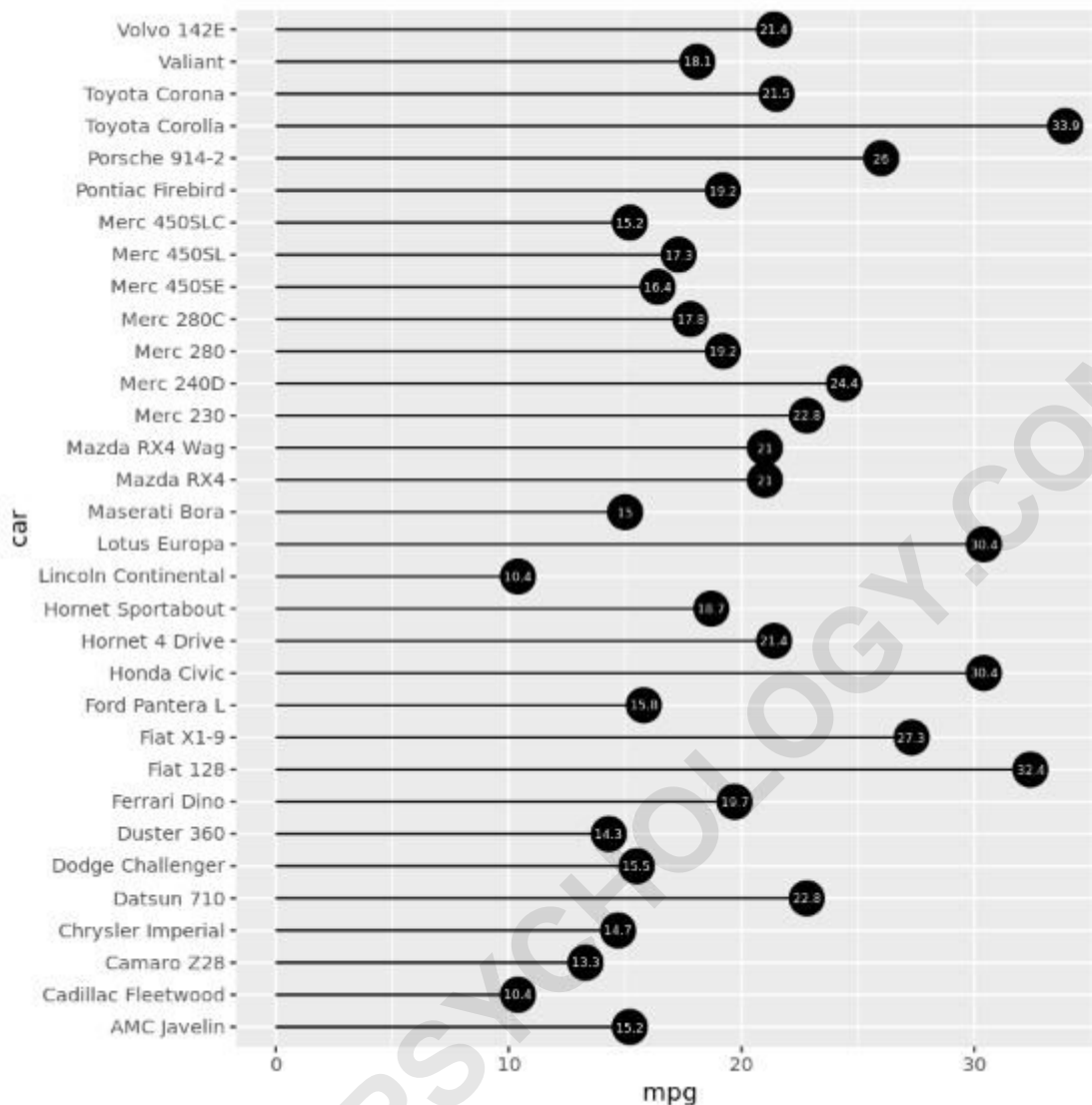
The choice between external and internal labels often depends on the intended **audience** and the medium of publication. External labels are generally easier to read in printed reports, whereas internal labels can look striking in **digital dashboards**. The following **pre-formatted** code blocks demonstrate both techniques, allowing you to experiment with different **visual styles** to see which best suits your specific **data storytelling** needs. Remember that consistency in labeling is key to maintaining a professional appearance throughout your **data analysis** project.

```
ggplot(mtcars, aes(x = mpg, y = car, label = mpg)) +  
geom_segment(aes(x = 0, y = car, xend = mpg, yend = car)) +  
geom_point() +  
geom_text(nudge_x = 1.5)
```



By enlarging the circles and adjusting the text parameters, we can achieve the following result:

```
ggplot(mtcars, aes(x = mpg, y = car, label = mpg)) +
  geom_segment(aes(x = 0, y = car, xend = mpg, yend = car)) +
  geom_point(size = 7) +
  geom_text(color = 'white', size = 2)
```



## Advanced Comparative Analysis: Benchmarking Against the Mean

One of the most powerful applications of the **lollipop chart** is comparing individual **observations** to a baseline, such as the **arithmetic mean**. This allows the viewer to instantly identify which categories are performing above or below average. To perform this **comparative analysis**, we leverage the **dplyr** package, which is part of the **Tidyverse** ecosystem. We use **dplyr** to calculate the mean **mpg** and create a logical **flag** that categorizes each car based on its performance relative to that mean.

Furthermore, we can use **dplyr** to sort the data. A sorted **lollipop chart** is much more effective than one arranged alphabetically because it allows the reader to perceive the **ranking** and **magnitude** of differences immediately. By converting the car names into a **factor** with levels ordered by their **mpg** values, we ensure that **ggplot2** renders the chart in a logical, descending or

ascending sequence. This step is a critical component of **data storytelling**, as it highlights the "best" and "worst" performers in the dataset.

In the **code** below, we demonstrate how to pipe these **data manipulation** steps together. By creating a new **data frame** called **mtcars\_new**, we preserve the original data while generating a version specifically optimized for our **diverging lollipop chart**. This workflow is highly reproducible and follows the best practices of **modern data science**, ensuring that your analysis is both transparent and easy to modify for future datasets.

**#load library dplyr**

**library(dplyr)**

**#find mean value of mpg and arrange cars in order by mpg descending**

**mtcars\_new <- mtcars %>%**

**arrange(mpg) %>%**

**mutate(mean\_mpg = mean(mpg),**

**flag = ifelse(mpg - mean\_mpg > 0, TRUE, FALSE),**

**car = factor(car, levels = .\$car))**

**#view first six rows of mtcars\_new**

**head(mtcars\_new)**

mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	car	mean_mpg	flag
10.4	8	472	205	2.93	5.250	17.98	0	0	3	4	Cadillac Fleetwood	20.09062	FALSE
10.4	8	460	215	3.00	5.424	17.82	0	0	3	4	Lincoln Continental	20.09062	FALSE
13.3	8	350	245	3.73	3.840	15.41	0	0	3	4	Camaro Z28	20.09062	FALSE
14.3	8	360	245	3.21	3.570	15.84	0	0	3	4	Duster 360	20.09062	FALSE
14.7	8	440	230	3.23	5.345	17.42	0	0	3	4	Chrysler Imperial	20.09062	FALSE
15.0	8	301	335	3.54	3.570	14.60	0	1	5	8	Maserati Bora	20.09062	FALSE

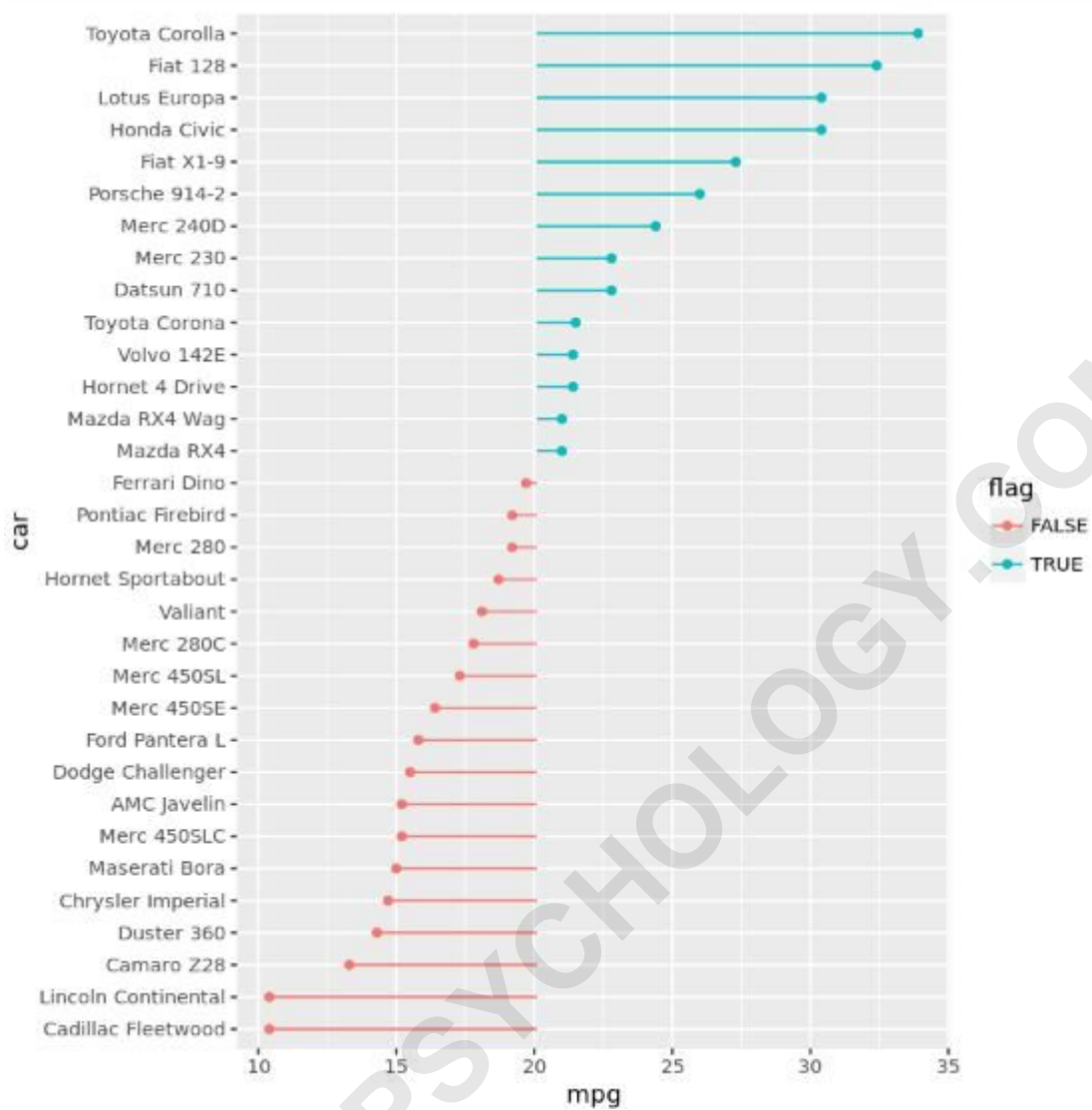
## Visualizing Deviations with Custom Color Schemes

Once the data is prepared with logical flags, we can map the **color** aesthetic to our **flag** variable. This creates a **diverging chart** where the lollipops extend from the **mean** value rather than from zero. This visual shift is significant because it changes the focus from absolute values to **deviations** from the average. In **ggplot2**, we modify the **geom\_segment** function so that the starting x-coordinate is the **mean\_mpg**, resulting in a chart that clearly distinguishes between positive and negative performance.

While **R** provides default colors for discrete scales, professional **data visualization** often requires custom **color palettes**. The `scale_colour_manual` function allows us to specify exact colors, such as purple and blue, to represent the "Above Average" and "Below Average" categories. Choosing high-contrast, **color-blind friendly** palettes is an important consideration for ensuring that your **charts** are accessible to all readers. This level of customization is what separates a standard **statistical plot** from a compelling visual narrative.

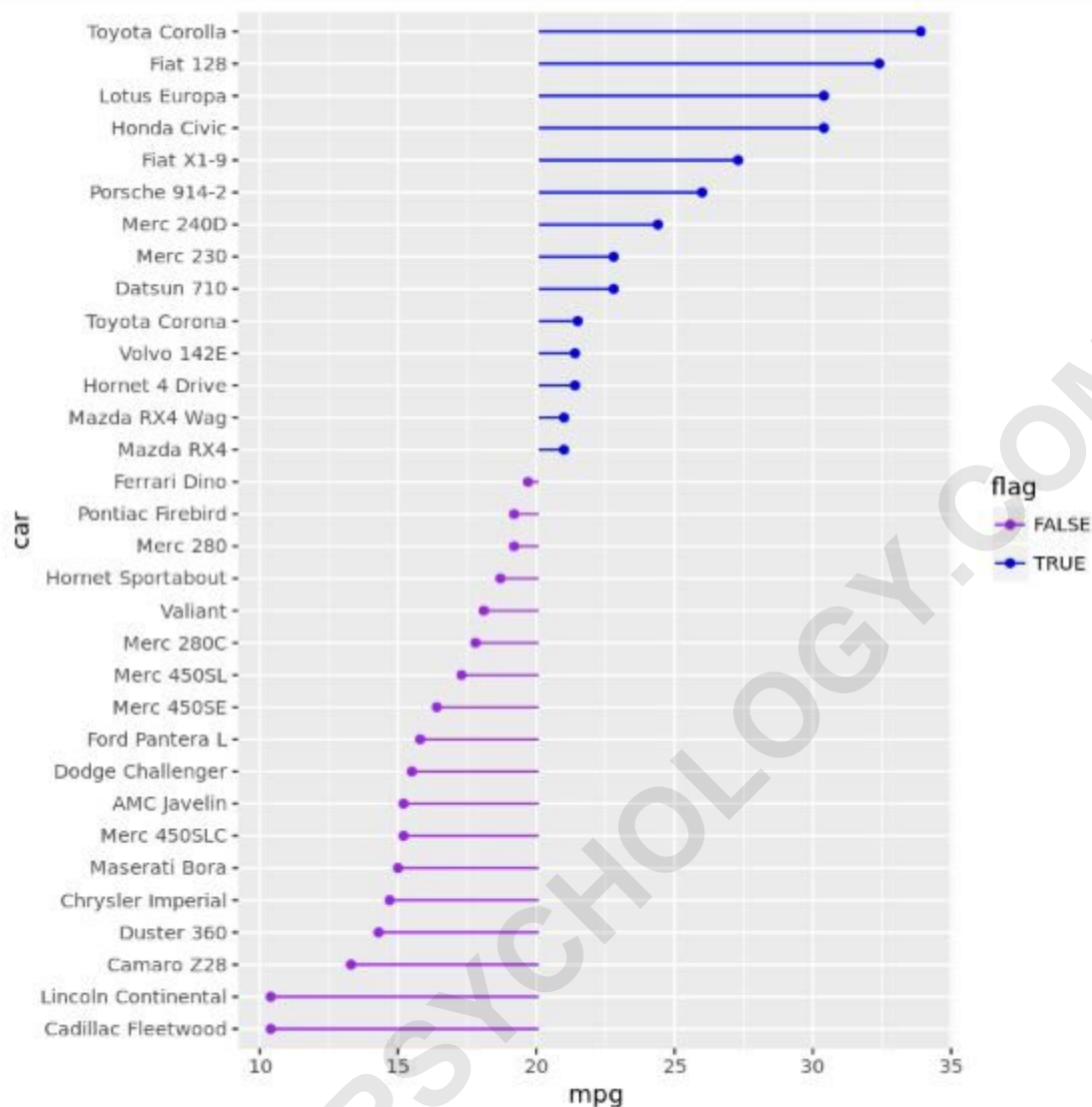
The following **code** provides the implementation for this **diverging lollipop chart**. By combining **logical flagging** with custom **color mapping**, we create a tool that is perfect for **performance reviews** or **benchmarking** studies. The simplicity of the lines combined with the clear color coding makes the **data insights** accessible at a glance, proving that the **lollipop chart** is as functional as it is beautiful.

```
ggplot(mtcars_new, aes(x = mpg, y = car, color = flag)) +  
geom_segment(aes(x = mean_mpg, y = car, xend = mpg, yend = car)) +  
geom_point()
```



To further customize the appearance with specific colors, you can use the following approach:

```
ggplot(mtcars_new, aes(x = mpg, y = car, color = flag)) +
  geom_segment(aes(x = mean_mpg, y = car, xend = mpg, yend = car)) +
  geom_point() +
  scale_colour_manual(values = c("purple", "blue"))
```



## Final Aesthetic Refinements and Professional Theming

The final step in our journey involves transforming a functional plot into a **publication-quality** graphic. **ggplot2** offers a vast array of **theming** options that allow you to control every **aesthetic** detail, from the background grid to the font family. By applying **theme\_minimal()**, we remove unnecessary elements that do not contribute to the **data interpretation**. We also use **annotate** to add descriptive text and arrows, which act as **visual cues** to guide the reader through the **chart's** findings.

Detailed **theming** also involves adjusting the **axis titles**, **legend position**, and **plot margins**. In the final version of our chart, we hide the **legend** because the **annotations** and colors already provide enough context. We also select a specific **typeface**, such as Georgia, to give the chart a sophisticated feel. These final touches might seem minor, but they are essential for creating a **data**

**visualization** that looks deliberate, authoritative, and professional in a **business** or **academic** context.

Below is the complete, high-level **code** required to produce the final, polished **lollipop chart**. This snippet integrates everything we have learned: **data manipulation** with **dplyr**, **layered plotting** with **ggplot2**, and advanced **aesthetic customization**. By studying this **code**, you can gain a deeper understanding of how to leverage **R** for professional-grade **visual communication**.

```
ggplot(mtcars_new, aes(x = mpg, y = car, color = flag)) +  
geom_segment(aes(x = mean_mpg, y = car, xend = mpg, yend = car), color = "grey") +  
geom_point() +  
annotate("text", x = 27, y = 20, label = "Above Average", color = "#00BFC4", size = 3, hjust =  
-0.1, vjust = .75) +  
annotate("text", x = 27, y = 17, label = "Below Average", color = "#F8766D", size = 3, hjust =  
-0.1, vjust = -.1) +  
geom_segment(aes(x = 26.5, xend = 26.5, y = 19, yend = 23),  
arrow = arrow(length = unit(0.2,"cm")), color = "#00BFC4") +  
geom_segment(aes(x = 26.5, xend = 26.5, y = 18, yend = 14),  
arrow = arrow(length = unit(0.2,"cm")), color = "#F8766D") +  
labs(title = "Miles per Gallon by Car") +  
theme_minimal() +  
theme(axis.title = element_blank(),  
panel.grid.minor = element_blank(),  
legend.position = "none",  
text = element_text(family = "Georgia"),  
axis.text.y = element_text(size = 8),  
plot.title = element_text(size = 20, margin = margin(b = 10), hjust = 0),  
plot.subtitle = element_text(size = 12, color = "darkslategrey", margin = margin(b = 25, l =  
-25)),  
plot.caption = element_text(size = 8, margin = margin(t = 10), color = "grey70", hjust = 0))
```

### Miles per Gallon by Car

