Data Visualization with ggplot2

Computer Applications for the Psychological Sciencees

PSYC470

Contents

Chapter Goal

To build a strong foundational understanding of the "Grammar of Graphics" and empower you to create beautiful, informative, and highly customized data visualizations using the ggplot2 package in R.

Why ggplot2? The Grammar of Graphics

Introduction: When you first start plotting in R, you often use "base R" functions like plot() or hist(). This approach is like painting on a canvas; you issue a command, and something immediately appears. If you want to add a title or change a color, you issue another command to modify the existing plot. While quick for simple tasks, this "imperative" method can become complex and unwieldy as your visualizations grow more sophisticated. ggplot2 introduces a different, more powerful philosophy. Instead of telling R how to draw a plot step-by-step, you declare what you want the plot to represent. This declarative approach, based on a system called the Grammar of Graphics, allows you to build complex plots in a more structured, reproducible, and intuitive way.

The power of ggplot2 comes from the fact that it is not just a collection of pre-packaged chart types. Instead, it provides a set of grammatical rules and components that you can combine to create a vast, almost infinite, variety of visualizations. Just as English grammar allows you to combine nouns, verbs, and adjectives to form meaningful sentences, the Grammar of Graphics allows you to combine components like data, aesthetic mappings, and geometric objects to build meaningful plots. The key is to think in layers. You start by defining your dataset and the core aesthetic mappings, and then you add layers on top: a layer of points, perhaps a layer for a line of best fit, a layer for labels, and so on. This modular approach is what makes ggplot2 so flexible and powerful.

The Three Essential Components

Every plot you create with ggplot2 is built upon a foundational triad of components. Understanding these three parts is the key to unlocking the full power of the package.

1) data:

The first ingredient is always your data. ggplot2 is designed to work with data frames, where your data is organized in a "tidy" format: each column represents a variable, and each row represents an observation. This structure is essential because ggplot2 needs to know exactly where to find the variables you want to plot.

2) aes() (Aesthetic Mappings):

This is the most crucial concept in the Grammar of Graphics. The `aes()` function is where you define how variables from your data frame are mapped to the visual properties (i.e., the aesthetics) of your plot. This is where you connect your data to what you see. The most common aesthetics are `x` and `y` for position on the axes, but there are many others, such as `color`, `fill`, `shape`, `size`, and `alpha` (transparency). By mapping a variable like gender to the color aesthetic, for example, you are telling ggplot2 to assign a different color to each gender group automatically.

3) geom_...() (Geometric Objects):

The geoms are the "verbs" of your plot. They determine what is actually drawn to represent the data. Each geom function adds a new layer to your plot. If you want a scatter plot, you use `geom_point()`. If you want a line chart, you use `geom_line()`. If you want a bar chart, you use `geom_bar()`. Because you add geoms as layers, you can easily combine them. For instance, you can create a scatter plot with a line of best fit by simply adding a `geom_point()` layer followed by a `geom_smooth()` layer.

Basic Plots: Visualizing Relationships and Distributions

This section will provide hands-on examples for creating the most common plot types. We'll use the starwars dataset, which is built into the dplyr package, for our examples. Make sure you have ggplot2 and dplyr installed and loaded.

```r
# Make sure the necessary libraries are loaded
library(ggplot2)
library(dplyr)
```

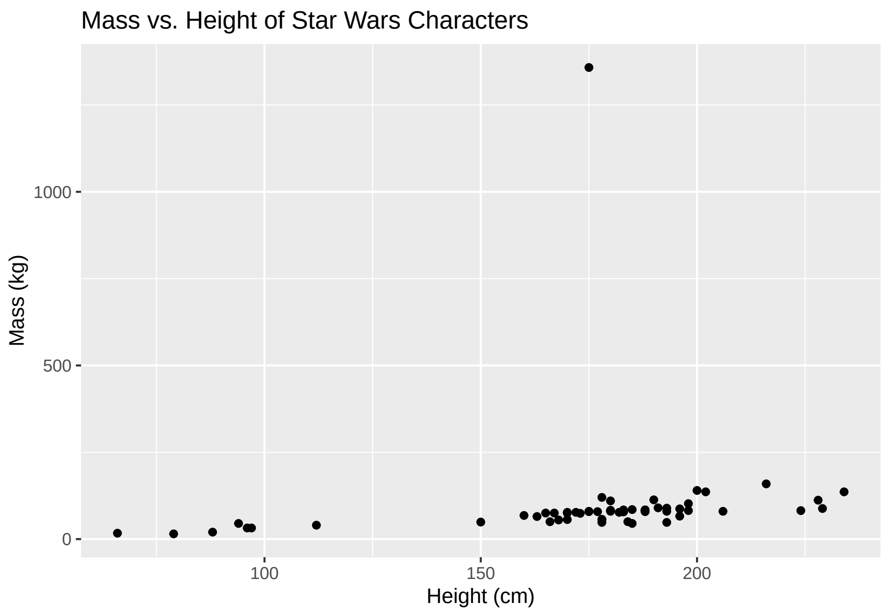
```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Scatter Plots (geom_point):

Scatterplots are the primary tool for visualizing the relationship between two continuous variables. Let's ask a simple question: In the Star Wars universe, do taller characters tend to have more mass?

Hands-on Example: Create a simple scatter plot of height vs. mass.

```r
# We use filter() to remove characters with unknown height or mass
starwars_filtered <- filter(starwars, !is.na(height), !is.na(mass))

ggplot(data = starwars_filtered, aes(x = height, y = mass)) +
  geom_point() +
  labs(title = "Mass vs. Height of Star Wars Characters",
       x = "Height (cm)", y = "Mass (kg)")
```



This plot appears to show a positive relationship, but there is a massive outlier. Who could that be?

```
starwars %>%
  filter(mass>500) %>%
  select(name)
```

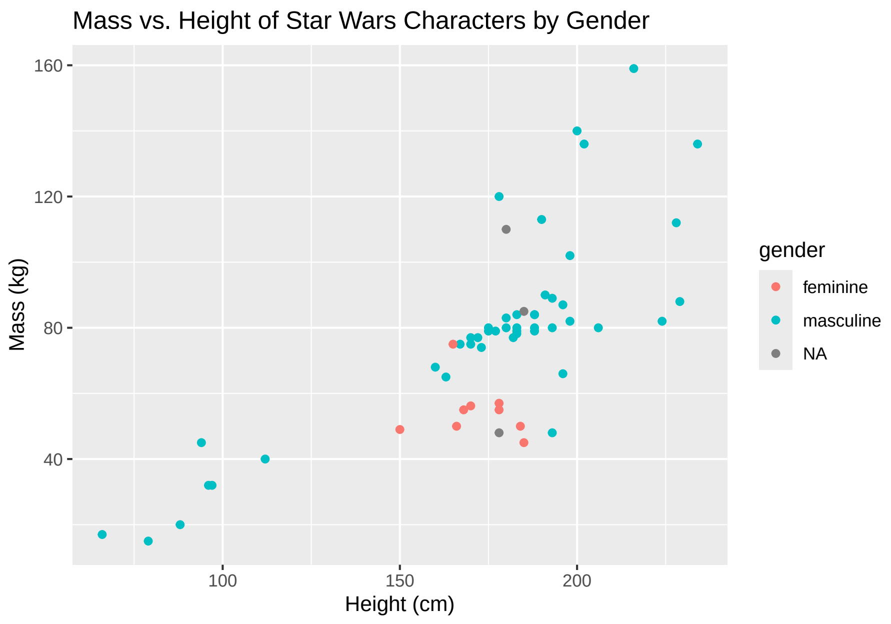
```
## # A tibble: 1 x 1
##   name
##   <chr>
## 1 Jabba Desilijic Tiure
```

It's Jabba the Hutt!!!

Adding Layers

We can add a third variable to reveal deeper patterns. Let's see if the height/mass relationship differs by gender (excluding Jabba). We can map the gender variable to the color aesthetic. In this example, we will use the pipe operator (%>%) with ggplot.

```
starwars_filtered %>%
  filter(mass<500) %>% #Sorry Jabba
  ggplot(aes(x = height, y = mass, color = gender)) + #Note the data argument is omitted when piped
    geom_point() +
    labs(title = "Mass vs. Height of Star Wars Characters by Gender",
      x = "Height (cm)", y = "Mass (kg)")
```



Now we can see the data broken down by gender, with a legend automatically created for us.

Bar Charts (geom_bar and geom_col):

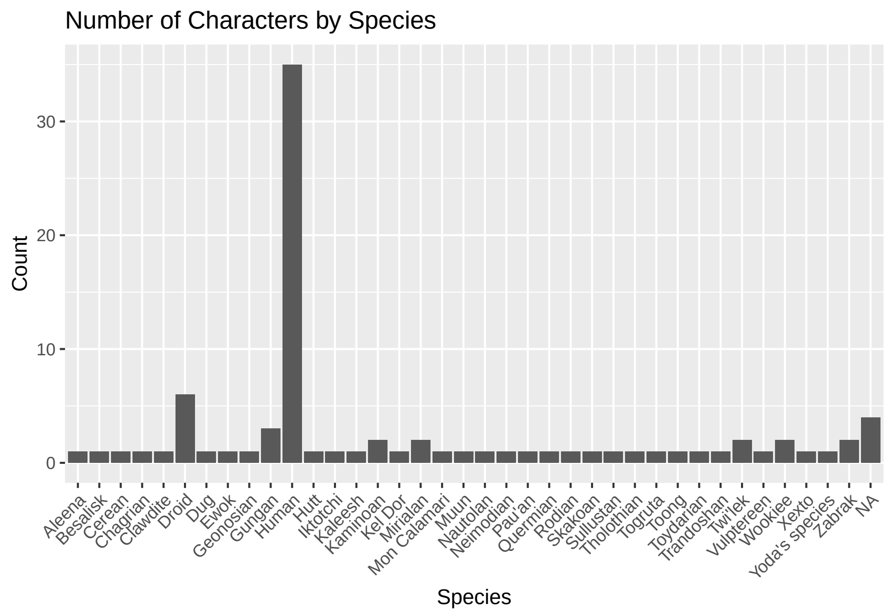
The best tool for comparing quantities across different categories is the bar plot. The two main geoms for bar charts serve different purposes.

- `geom_bar()`: Use this when you want ggplot2 to count the number of rows for each category. It works directly on the raw data.
- `geom_col()`: Use this when you have already calculated the value you want to plot (e.g., a mean or a sum) and have a column in your data that represents that value.

Example (geom_bar)

Let's find out which species are most common in our dataset.

```r
ggplot(data = starwars, aes(x = species)) +
  geom_bar() +
  labs(title = "Number of Characters by Species", x = "Species", y = "Count") +
  # theme() helps make axis labels readable
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



This plot quickly shows us that Humans are, by far, the most common species, followed by Droids.

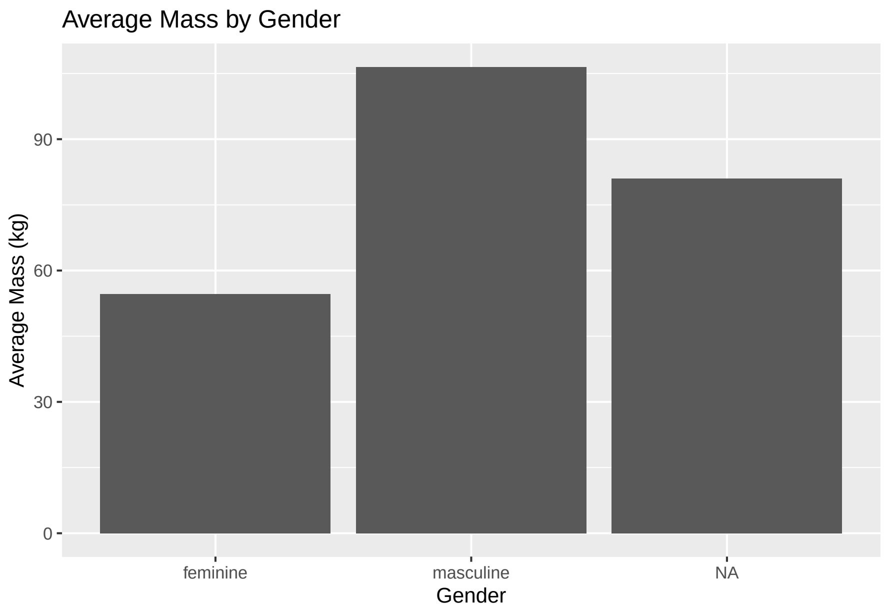
You may have noticed that we added a `theme()` line during the construction of this ggplot. We will discuss plot themes in more detail later, but here is a rundown of how that line works and why it was needed: * `theme()`: This is the main ggplot2 function for controlling all the non-data elements of your plot. This includes things like titles, axis labels, legends, the plot background, and grid lines. If you want to change the look and feel (not the data), you use theme().

- `axis.text.x`: This is the specific theme element you want to change. It targets the text labels on the x-axis (the "tick labels" like "First Quarter", "Second Quarter", etc.). `axis.text.y` would target the y-axis text, and `plot.title` would target the main title.
- `element_text()`: This is a helper function that you must use whenever you are modifying a text element. It tells `theme()` that you are about to provide styling instructions specifically for text, such as size, color, font, and in this case, angle and justification.
  - `angle = 45`: This is an argument inside element_text(). It sets the rotation angle of the text in degrees. angle = 45 rotates the labels 45 degrees counter-clockwise.
  - `hjust = 1`: This is the most subtle but crucial part. hjust stands for horizontal justification. It controls the horizontal alignment of the text relative to its anchor point on the axis. It takes a value from 0 (left-justified) to 1 (right justified):
    * Why it's important: When you rotate text by 45 degrees, the text's anchor point is its left edge. If you don't adjust the justification, the labels will be rotated but misaligned. By setting hjust = 1, you are telling ggplot to align the right end of each rotated label with the tick mark on the axis, which is the visually correct alignment.

Example (geom_col)

Let's calculate the average mass for each gender and plot that result.

```r
# First, create a summary data frame
gender_mass_summary <- starwars_filtered %>%
  group_by(gender) %>%
  summarise(average_mass = mean(mass))

# Now, use geom_col() to plot the pre-calculated 'average_mass'
ggplot(data = gender_mass_summary, aes(x = gender, y = average_mass)) +
  geom_col() +
  labs(title = "Average Mass by Gender", x = "Gender", y = "Average Mass (kg)")
```



Histograms and Density Plots (geom_histogram and geom_density):

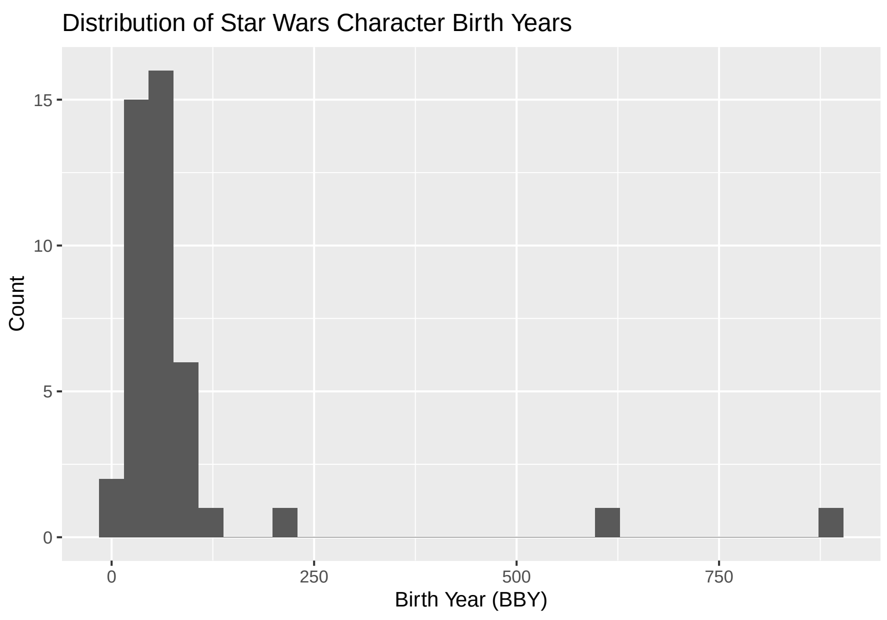
Histogram and Density plots are essential for understanding the distribution of a single continuous variable. Let's examine the distribution of character birth years.

Example: Plot the distribution of birth_year.

```r
ggplot(data = starwars, aes(x = birth_year)) +
  geom_histogram() +
  labs(title = "Distribution of Star Wars Character Birth Years",
       x = "Birth Year (BBY)", y = "Count")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 44 rows containing non-finite outside the scale range
## (`stat_bin()`).
```



Note that the units of the birth_year variable are BBY, an acronym for Before Battle of Yavin. The histogram shows most characters were born in a relatively recent period, with a few ancient outliers.

```
starwars_filtered %>%
  filter(birth_year>750) %>%
  select(name,birth_year)
```

```
## # A tibble: 1 x 2
##   name  birth_year
##   <chr>      <dbl>
## 1 Yoda         896
```

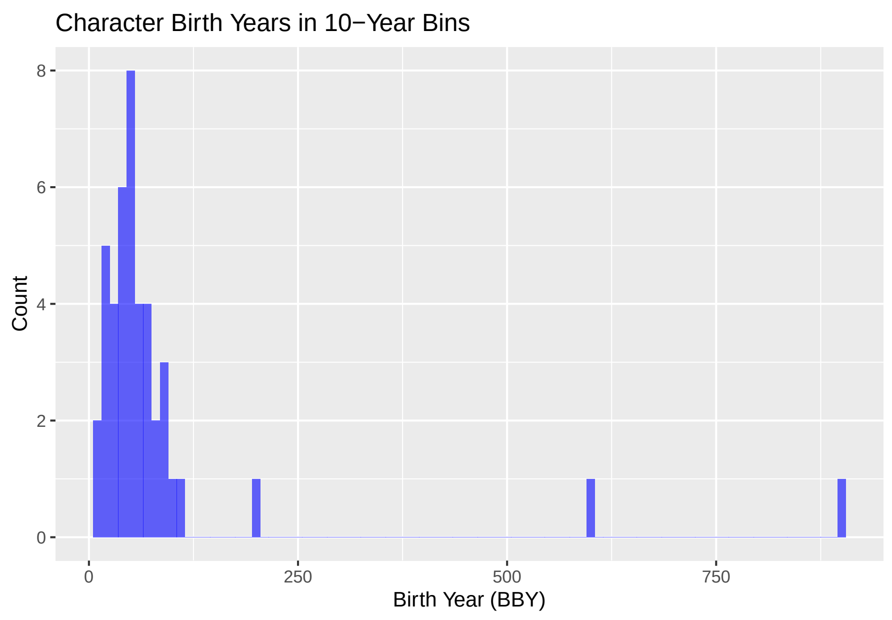
It looks like Yoda was born 896 years BBY!

You may have noticed that ggplot also provided a couple of messages with its output. The first is simply a suggestion, and the second is a warning. The Suggestion: "stat_bin() using bins = 30. Pick better value with binwidth." is ggplot2's friendly way of saying, "I had to guess how to draw your histogram, and you might not like my guess." By default, `geom_histogram()` divides your data into 30 vertical bars (bins) of equal width. This is often not the best choice, as it can hide or over-emphasize patterns in the data.

The Solution: The message itself tells you the solution: use the binwidth argument to take control. This is considered better practice because you are making a conscious choice based on your data. Here is an example of how to do this:

```
ggplot(data = starwars, aes(x = birth_year)) +
  geom_histogram(binwidth = 10, fill = "blue", alpha = 0.6) + # 10-year bins
  labs(title = "Character Birth Years in 10-Year Bins",
       x = "Birth Year (BBY)", y = "Count")
```

```
## Warning: Removed 44 rows containing non-finite outside the scale range
## (`stat_bin()`).
```



By setting binwidth = 10, you are explicitly telling ggplot2 that each bar should represent a 10-year span, which is much more interpretable than the default 30 bins.

The Warning: “Removed 44 rows containing non-finite values. . . ” is telling you that 44 characters in the starwars dataset have no value (an NA or “Not Available”) for their birth_year. A histogram can only be drawn with actual numbers, so ggplot2 automatically removes these rows before creating the plot.

Is it a problem? Not usually. It’s good that it removes them, but it’s also good that it tells you it did. The best practice is to be explicit and remove these missing values yourself before you plot. This makes your code clearer and shows that you are aware of the missing data.

Boxplots (geom_boxplot):

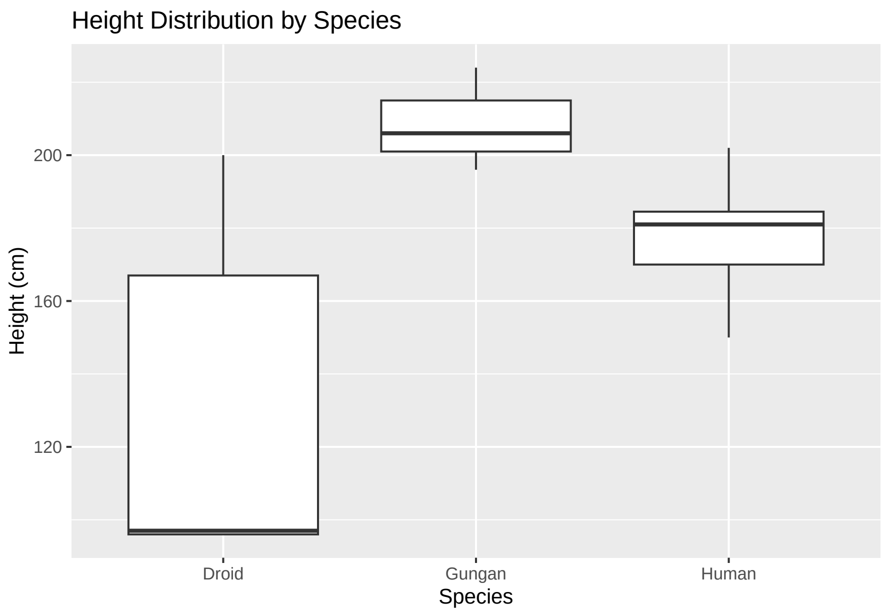
Boxplots are ideal for comparing the distributions of a continuous variable across multiple groups. They pack a lot of statistical information into a compact visual summary. Each boxplot consists of several key elements:

- The thick line in the middle of the box represents the median (the 50th percentile), which is the midpoint of the data.
- The bottom and top edges of the box represent the first quartile (Q1) (the 25th percentile) and the third quartile (Q3) (the 75th percentile), respectively.
  - Thus, the height of the box itself is the Interquartile Range (IQR), which contains the middle 50% of the data and gives a sense of the data’s spread.
- The “whiskers” extend from the box to show the range of the data. By default, they extend to the furthest data point that is within 1.5 times the IQR from the edge of the box.
- Any data points that fall outside the whiskers are plotted as individual points and are considered potential outliers.

This structure makes it very easy to compare the central tendency, spread, and skewness of different groups at a glanceLet’s compare the height distributions of the three most common species: Humans, Droids, and Gungans.

Example: Compare the height of different species.

```r
# Filter for the top 3 species and remove NAs to keep the plot clean
top_species <- starwars %>%
  filter(species %in% c("Human", "Droid", "Gungan"), !is.na(height))

ggplot(data = top_species, aes(x = species, y = height)) +
  geom_boxplot() +
  labs(title = "Height Distribution by Species",
       x = "Species", y = "Height (cm)")
```



Interesting, It would appear that humans in the Star Wars universe have a median height around 180cm! Droids are suprisingly short by comparison, with a median height of only ~110cm, but the distribution has a clear positive skew.

Enhancing and Customizing Your Plots

Creating plots is an essential first step, but the true power of ggplot2 is revealed in its deep customization options. This is where you move from a basic, functional chart to a polished, publication-ready visualization that tells a clear and compelling story. The layered, grammatical approach means you are never limited to a pre-defined set of options; you can add context with labels, break down complex data with facets, and control every aesthetic detail to perfect your final product. In this section, we'll explore the key functions that allow you to take full control of your plots.

Labels, Titles, and Annotations (labs()):

A plot without clear labels is just a pretty picture. To make your visualizations informative and ready for a report or presentation, you need to add context. The labs() function is your primary tool for this, allowing you to control almost all the text on your plot.

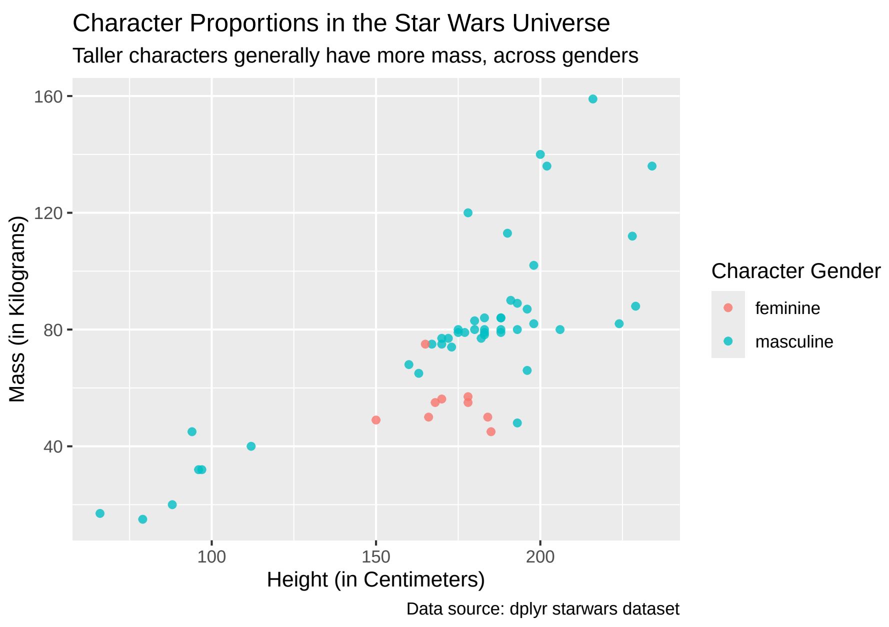
The most common arguments you'll use are:

- title: Adds a main title at the top of the plot.
- subtitle: Adds a smaller subtitle just below the main title.
- caption: Adds text at the bottom-right of the plot, perfect for citing data sources.

- x and y: Replaces the default axis titles with more descriptive text.
- color, fill, shape, size: Changes the title of the legend associated with that specific aesthetic.

Example: Let's take our height vs. mass scatter plot and make it more publication-ready.

```
starwars_filtered %>%
  filter(mass<500, !is.na(gender)) %>% #Sorry Jabba
  ggplot(aes(x = height, y = mass, color = gender)) +
    geom_point(alpha = 0.8) +
    labs(
      title = "Character Proportions in the Star Wars Universe",
      subtitle = "Taller characters generally have more mass, across genders",
      caption = "Data source: dplyr starwars dataset",
      x = "Height (in Centimeters)",
      y = "Mass (in Kilograms)",
      color = "Character Gender" # This changes the legend title
    )
```

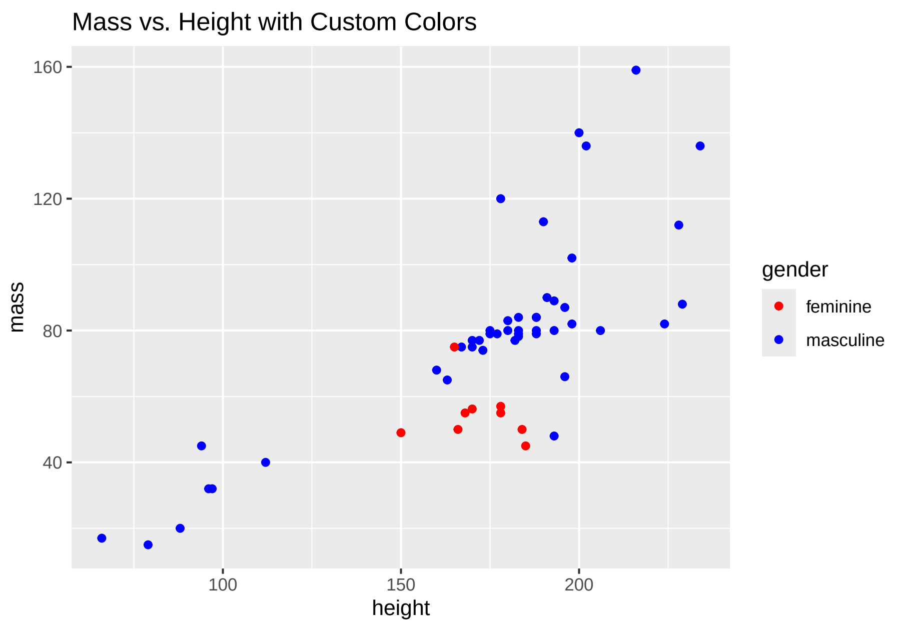


Adjusting Scales and Axes (scale_...):

The `scale_*` family of functions is your control panel for fine-tuning the details of your aesthetic mappings. While `aes()` maps a variable to an aesthetic, `scale_*` controls how that mapping is performed. This includes specifying the exact colors, breaks, and labels you want to use.

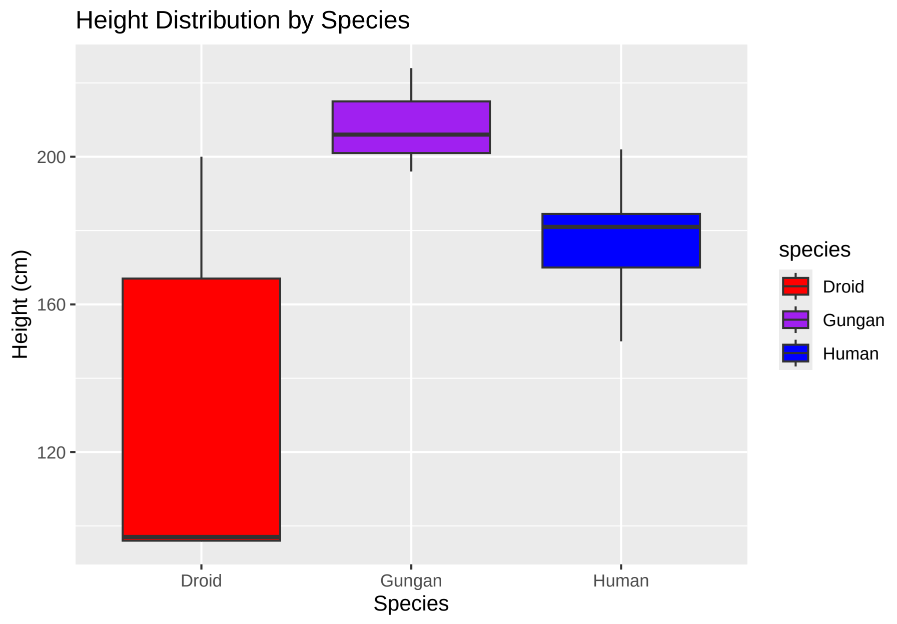
Specifying Manual Colors: To override the default colors, you use `scale_color_manual()` (for points and lines) or `scale_fill_manual()` (for areas like bars and boxes). The key is to provide a named vector to the values argument.

```
# Let's use specific colors for feminine and masculine characters in the scatterplot
starwars_filtered %>%
  filter(mass<500, !is.na(gender)) %>% #Sorry Jabba
  ggplot(aes(x = height, y = mass, color = gender)) +
    geom_point() +
    scale_color_manual(values = c("feminine" = "red", "masculine" = "blue")) +
    labs(title = "Mass vs. Height with Custom Colors")
```



When scaling the colors of bars in a bar chart, or boxes in the boxplot, use the `scale_fill_manual` function.

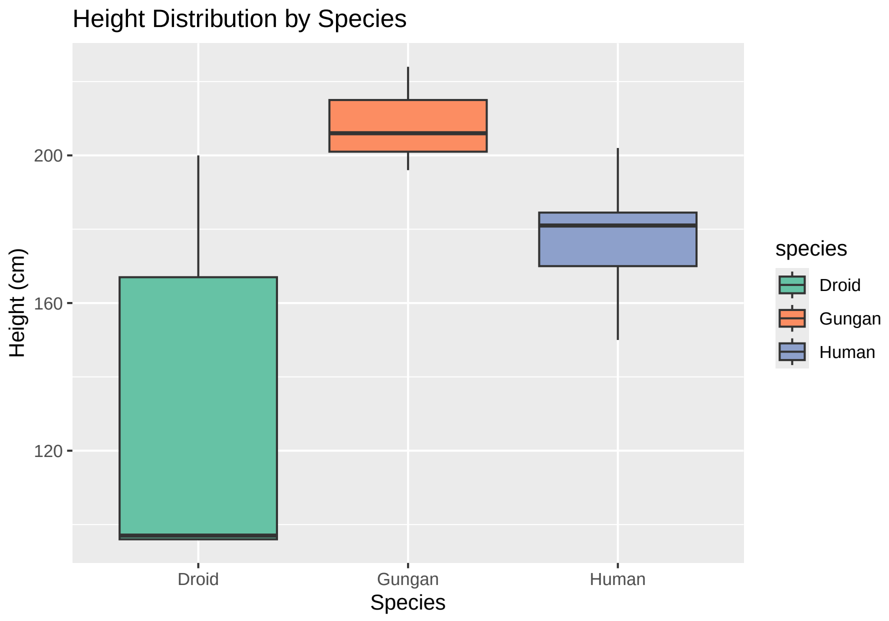
```r
# Filter for the top 3 species and remove NAs to keep the plot clean
top_species <- starwars %>%
  filter(species %in% c("Human", "Droid", "Gungan"), !is.na(height))

ggplot(data = top_species, aes(x = species, y = height, fill=species)) +
  geom_boxplot() +
  scale_fill_manual(values=c('red','purple','blue')) +
  labs(title = "Height Distribution by Species",
       x = "Species", y = "Height (cm)")
```



Using Pre-Built Palettes

Manually picking colors can be difficult. ggplot2 has built-in support for the excellent ColorBrewer palettes, which are designed for clear data visualization. You can use them with scale_color_brewer() or scale_fill_brewer().

```r
### Using a pre-built, colorblind-safe palette for the species boxplot
ggplot(data = top_species, aes(x = species, y = height, fill = species)) +
  geom_boxplot() +
  scale_fill_brewer(palette = "Set2") + # Use the "Set2" palette
  labs(title = "Height Distribution by Species",
       x = "Species", y = "Height (cm)")
```



Here is an overview of the ColorBrewer palettes:

Palette Type	Description	Palette Names
Sequential	Best for continuous data that goes from low to high. Light colors for low values, dark colors for high values.	Blues, Greens, Reds, Purples, Oranges, Greys
Diverging	Best for data with a meaningful midpoint (like 0). Emphasizes values at both extremes with different colors.	RdBu, BrBG, PiYG, PRGn, RdYlBu, RdGy
Qualitative	Best for categorical data where there is no order. Uses distinct colors that are easy to tell apart.	Set1, Set2, Set3, Pastel1, Pastel2, Accent

Themes (theme_...)

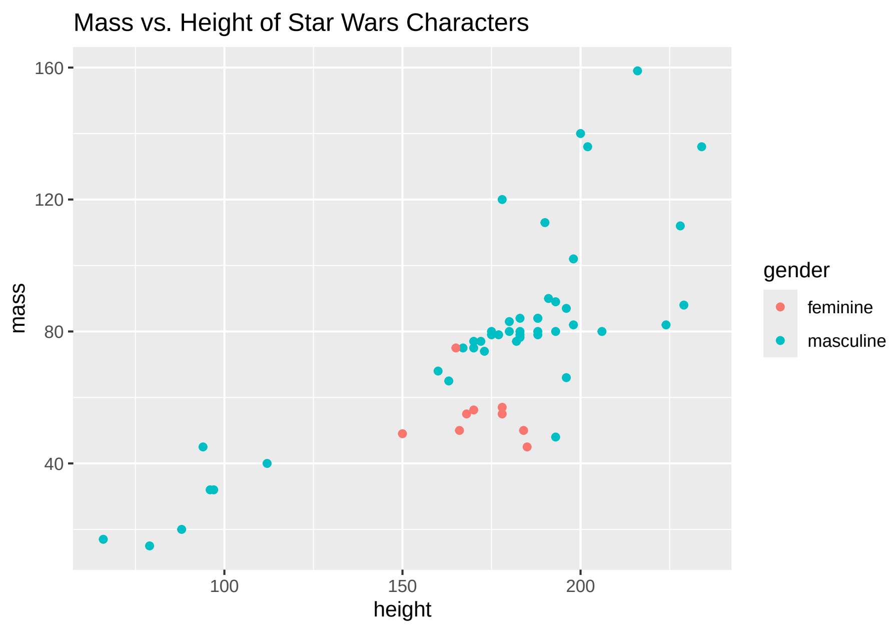
While `geom_*` and `scale_*` functions control the data elements of your plot, themes control the non-data elements. This includes things like the background color, gridlines, font sizes, and legend position. ggplot2's theming system allows you to change the overall look and feel of your plot with a single line of code.

Complete Themes: The easiest way to apply a theme is to add a "complete theme" layer. These functions change all the major display parameters at once.

Example:

Let's create a base plot and see how different themes affect its appearance.

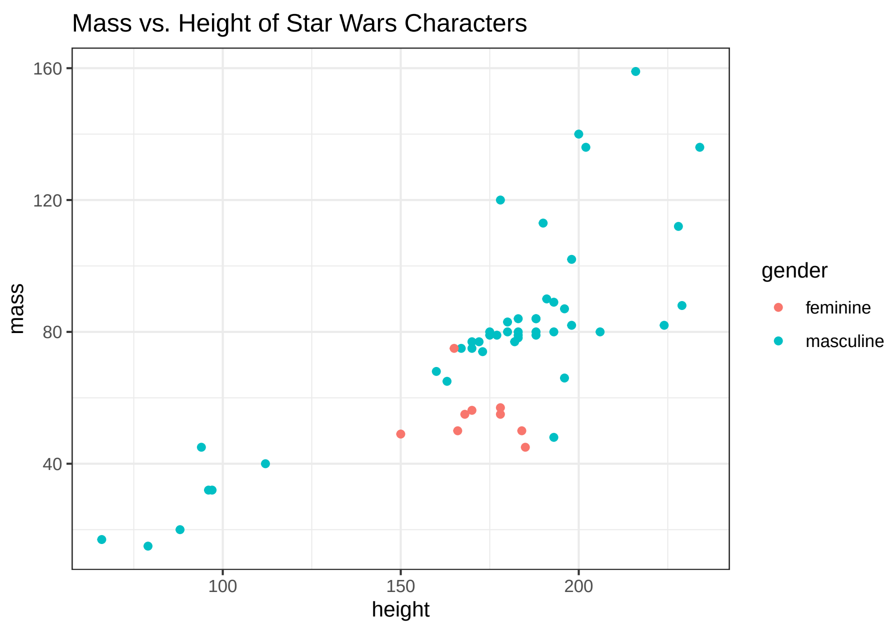
```
# First, get your data ready
starwars_clean <- starwars_filtered %>%
  filter(mass<500, !is.na(gender), !is.na(mass)) #Sorry Jabba

#Create a scatterplot object
p <- ggplot(data = starwars_clean, aes(x = height, y = mass, color = gender)) +
  geom_point() +
  labs(title = "Mass vs. Height of Star Wars Characters")
# The default theme is theme_gray()
  p + theme_gray()
```



Now, let's add different theme layers to our base plot p.

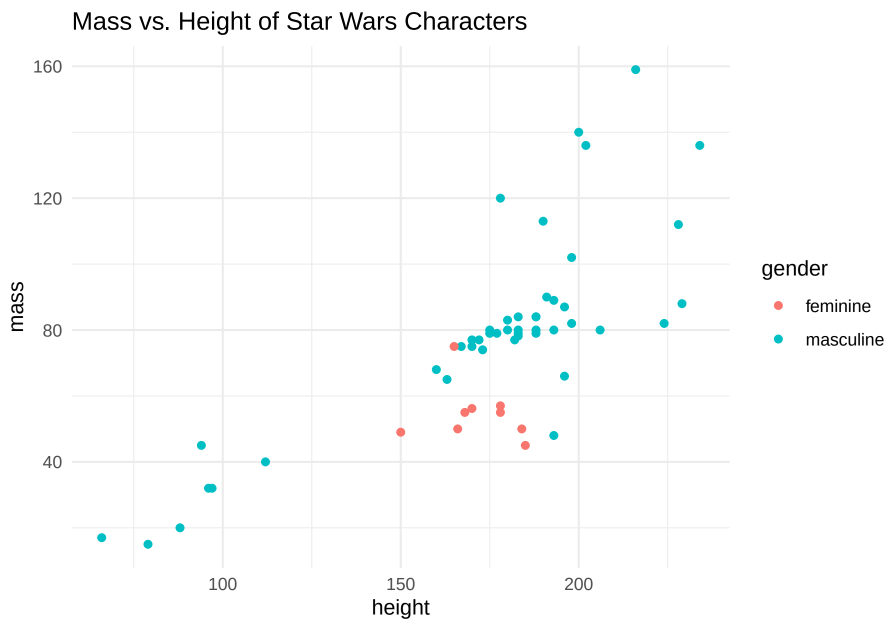
theme_bw() is popular theme that features a white background with black gridlines. It's often preferred for publications.

```
p + theme_bw()
```



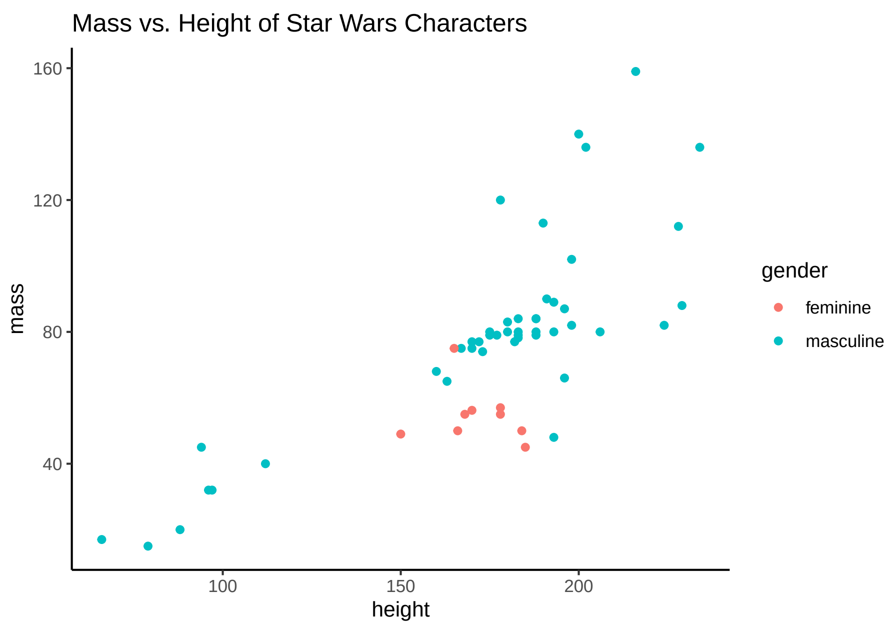
theme_minimal() is a clean theme that removes the background color and most of the axis lines, leaving just the gridlines. It's great for a modern, uncluttered look.

```r
p + theme_minimal()
```



The theme_classic() theme gives a more traditional scientific look, with x and y axis lines but no background or gridlines.

```r
p + theme_classic()
```



Creating Your Own Theme (theme()):

While complete themes are great, the real power comes from using the theme() function to fine-tune individual elements. This allows you to create a custom, reusable theme that matches a specific style guide, like APA format.

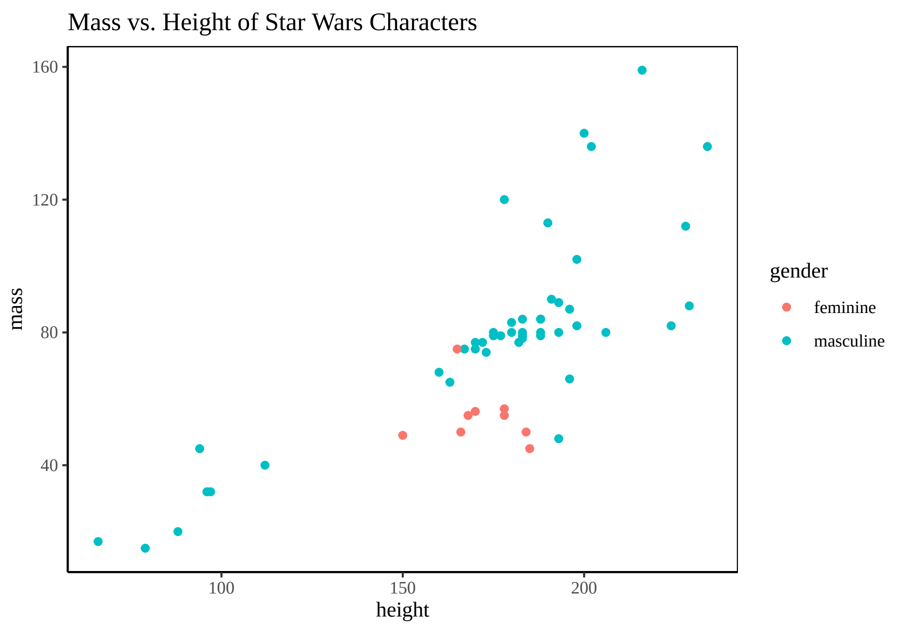
Example: Building theme_apa()

An APA-style plot is clean and simple: it has no background color, no gridlines, and uses a serif font. Let's build a function that creates this theme.

```
# Define our custom APA theme function
theme_apa <- function() {
  theme_classic() +  # Start with theme_classic as a base
  theme(
    panel.background = element_blank(), # Remove panel background
    panel.border = element_rect(color = "black", fill = NA), # Add a black border around the plot
    axis.line = element_line(color = "black"), # Make axis lines black
    text = element_text(family = "serif") # Use a serif font for all text
  )
}
```

Now, let's apply our new theme to the plot

```
p + theme_apa()
```



Below is a list of the most popular themes:

Theme	Description	Use Case
theme_gray()	The default theme. Gray background with white gridlines.	Good for general use and quick exploration.
theme_bw()	Black and white. A white background with thin, black gridlines.	Excellent for publications and reports where clarity is key.
theme_minimal()	A very clean look that removes the background and most axis lines.	Great for a modern, uncluttered, web-friendly aesthetic.
theme_classic()	Resembles a traditional scientific plot with x and y axis lines but no gridlines.	Often used in scientific papers; good for a simple, classic look.
theme_light()	Similar to theme_bw() but with light gray lines and axes for a softer look.	A good alternative to theme_bw() for a less stark appearance.
theme_void()	Removes everything but the data, including axes, labels, and background.	Used for non-standard plots like word clouds or when layering on top of an image.

Faceting: Creating Subplots (facet_wrap and facet_grid):

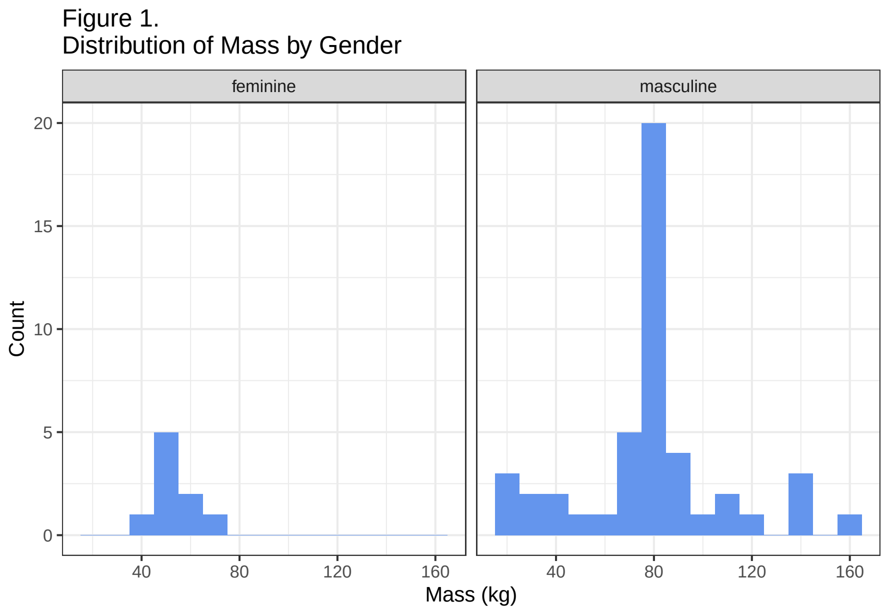
While using aesthetics like color and shape is great for adding variables, it can sometimes lead to a cluttered plot. Faceting is a powerful alternative that lets you split your main plot into a grid of smaller subplots (or "facets"). Each subplot displays the same type of graph for a different subset of your data. This is an incredibly effective way to compare patterns across different groups.

`facet_wrap(~ variable)` is the most common faceting function. You provide a formula with a tilde (~) followed by the name of a categorical variable. ggplot2 will create a separate plot for each level of that variable and arrange them in a sensible grid.

Example

Let's compare the distribution of character mass for each gender. We could try to overlay density plots, but this can get messy. Faceting provides a much clearer view.

```r
# First, let's filter out the extreme outlier (Jabba) for a more informative plot
starwars_mass_filtered <- starwars %>%
  filter(mass < 1000, !is.na(mass), !is.na(gender))

ggplot(data = starwars_mass_filtered, aes(x = mass)) +
  geom_histogram(binwidth = 10, fill = "cornflowerblue") +
  facet_wrap(~ gender) + # Create subplots based on the 'gender' variable
  theme_bw() +
  labs(
    title = "Figure 1.\nDistribution of Mass by Gender",
    x = "Mass (kg)",
    y = "Count"
  )
```



Beyond the Basics

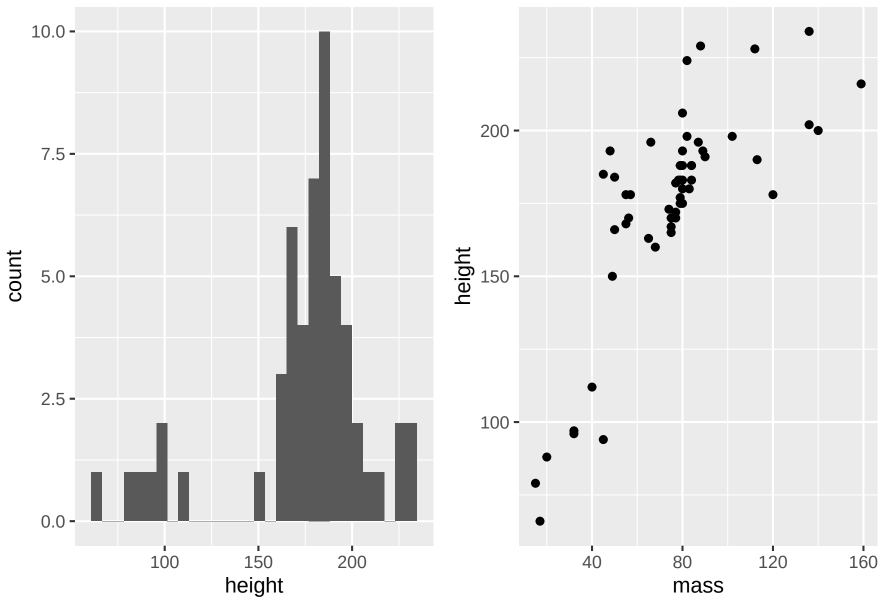
Combining Plots of Different Types

While faceting is a powerful way of combining multiple plots of the same type, you can't tell faceting to apply a different post-processing function to each panel. However, there is a very powerful and popular way to achieve the construction of complex Figures, consisting of multiple plots using the 'patchwork' package. It allows you to combine separate ggplot objects into a single figure with incredibly simple syntax. You just use arithmetic operators like + and / to arrange your plots in columns and rows respectively.

```r
# First, make sure you have patchwork installed
#install.packages("patchwork")
library(patchwork)

# Create two separate plots
p1 <- ggplot(starwars_clean, aes(x = height)) + geom_histogram()
p2 <- ggplot(starwars_clean, aes(x = mass, y=height)) + geom_point()
```

```r
# Combine them side-by-side
p1 + p2
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

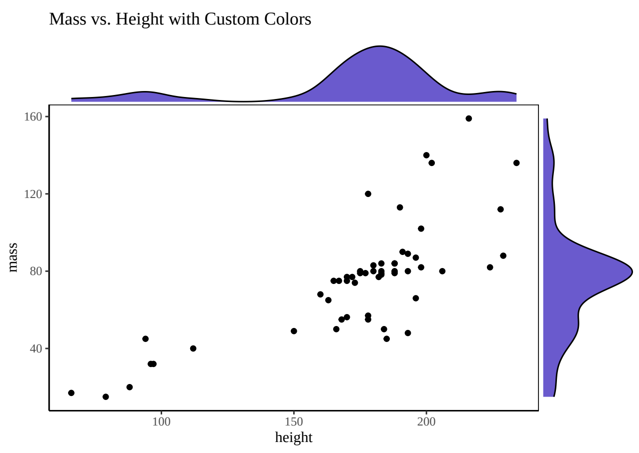


ggExtra()

The ggplot2 ecosystem is vast, and many add-on packages provide powerful new functionalities. The ggextra package is a great example. Its main function, ggMarginal(), allows you to add marginal plots (like histograms, density plots, or boxplots) to the top and right sides of a scatter plot. This is incredibly useful for seeing both the relationship between two variables and their individual distributions in the same figure.

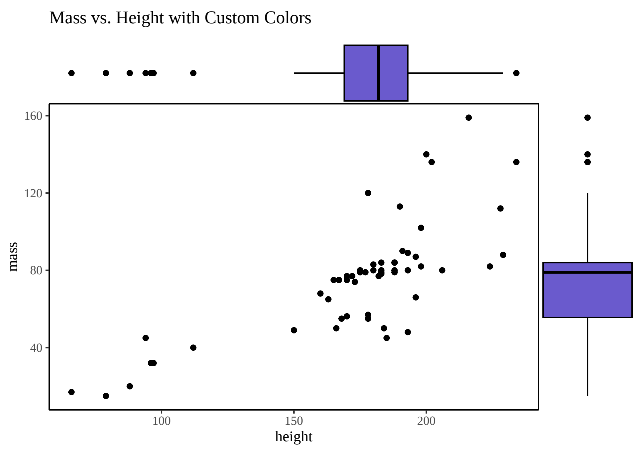
```r
# First, make sure you have ggextra installed and loaded
#install.packages("ggExtra")
library(ggExtra)

# Next, construct your scatterplot
p<-ggplot(data = starwars_clean, aes(x = height, y = mass)) +
  geom_point() +
  scale_color_manual(values = c("feminine" = "black", "masculine" = "darkgray")) +
  labs(title = "Mass vs. Height with Custom Colors") +
  theme_apa()

# Use the scatter plot 'p' as input to ggMarginal
ggMarginal(p, type = "density", fill = "slateblue")
```

```
## Warning: No shared levels found between `names(values)` of the manual scale and the
## data's colour values.
## No shared levels found between `names(values)` of the manual scale and the
## data's colour values.
## No shared levels found between `names(values)` of the manual scale and the
## data's colour values.
```



```r
ggMarginal(p, type = "boxplot", fill = "slateblue")
```

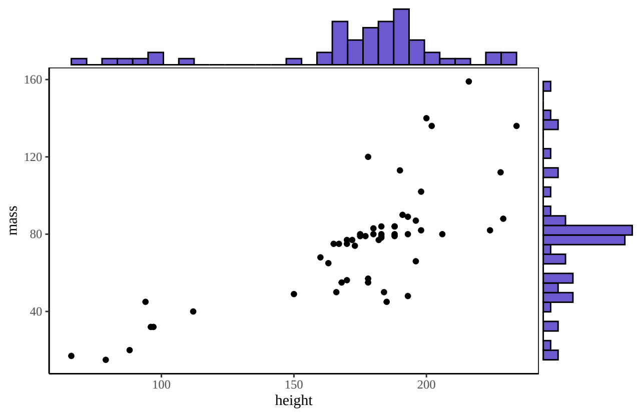
```
## Warning: No shared levels found between `names(values)` of the manual scale and the
## data's colour values.
## No shared levels found between `names(values)` of the manual scale and the
## data's colour values.
## No shared levels found between `names(values)` of the manual scale and the
## data's colour values.
```



```r
ggMarginal(p, type = "histogram", fill = "slateblue")
```

```
## Warning: No shared levels found between `names(values)` of the manual scale and the
## data's colour values.
## No shared levels found between `names(values)` of the manual scale and the
## data's colour values.
## No shared levels found between `names(values)` of the manual scale and the
## data's colour values.
```

Mass vs. Height with Custom Colors



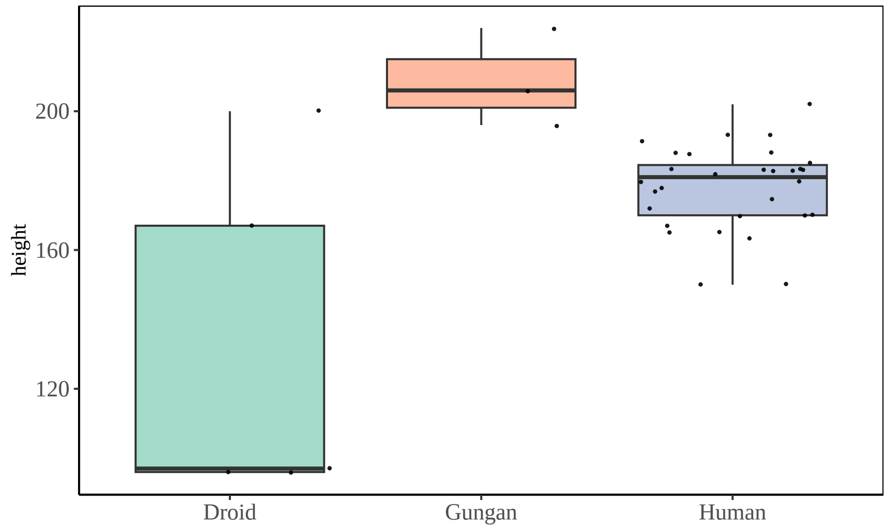
Better Boxplots

Boxplots are great at compressing a distribution into a few landmarks (median, quartiles, whiskers), but that very efficiency can sometimes be misleading. With small or unequal group sizes, whiskers and notches can look very different due to factors like multimodality, skew, and clustering. Also, many distinct shapes of the data distribution can yield identical boxes. Overlaying the raw data (e.g., geom_boxplot() plus a semi-transparent geom_jitter(width = . . . , alpha = . . .)) puts sample size, density, and gaps back on the page, helping you judge whether the box's story matches the data's shape. This pairs a compact summary with honest visibility of the observations—very much in the spirit of tidy, reproducible analyses.

Jitter

```r
starwars_topspecies<-starwars %>%
  filter(species %in% c("Human", "Droid", "Gungan"), !is.na(height))
# Plot
  ggplot(data=starwars_topspecies, aes(x=species, y=height, fill=species)) +
    geom_boxplot(alpha=.6) +
    #scale_fill_viridis(discrete = TRUE, alpha=0.6) +
    scale_fill_brewer(palette = "Set2") +
    geom_jitter(color="black", size=0.4, alpha=0.9) +
    theme_apa() +
    theme(
      legend.position="none",
      plot.title = element_text(size=12),
      axis.text = element_text(size=12)
    ) +
    ggtitle("Figure 1\nA boxplot with jitter") +
    xlab("")
```

Figure 1
A boxplot with jitter

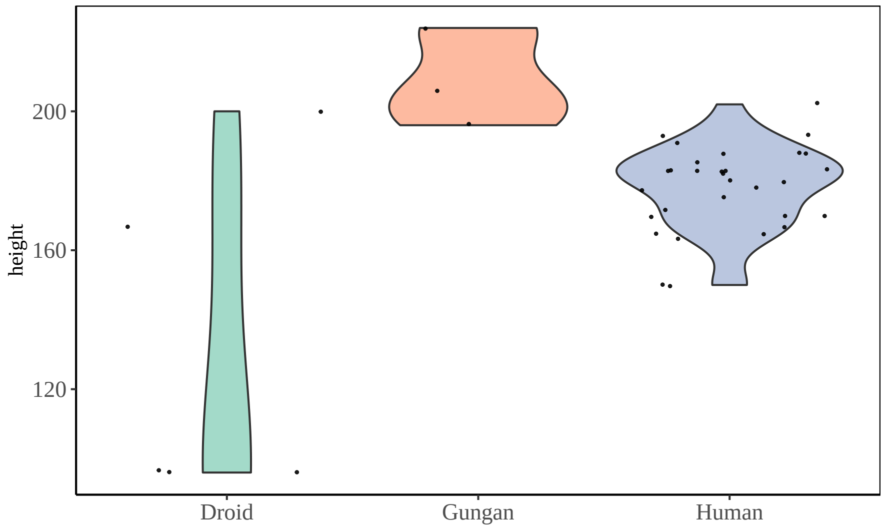


Violin plots

A violin plot is a hybrid between a boxplot and a kernel-density plot—it shows the familiar summary statistics of a boxplot (median and interquartile range) while also revealing the full shape of the distribution. Each "violin" is a smoothed estimate of the data's density, mirrored to create a symmetrical, violin-like outline. This makes it easy to see whether values are skewed, multimodal, or tightly clustered—patterns that a boxplot alone can hide. In R's ggplot2, you can make one with geom_violin(), and many analysts combine it with a jittered scatter layer (geom_jitter(alpha=0.3, width=0.1)) to display the individual observations on top of the density shape. The result gives both a statistical summary and a sense of the raw data's texture, helping you judge variation and overlap at a glance—perfect for the "gold-in, gold-out" spirit of tidy visualization.

```
# Plot
  ggplot(data=starwars_topspecies, aes(x=species, y=height, fill=species)) +
    geom_violin(alpha=.6) +
    #scale_fill_viridis(discrete = TRUE, alpha=0.6) +
    scale_fill_brewer(palette = "Set2") +
    geom_jitter(color="black", size=0.4, alpha=0.9) +
    theme_apa() +
    theme(
      legend.position="none",
      plot.title = element_text(size=12),
      axis.text = element_text(size=12)
    ) +
    ggtitle("Figure 1\nA boxplot with jitter") +
    xlab("")
```

Figure 1
A boxplot with jitter



Grouped Bar/Column Charts

When we compare data across two categorical variables, a simple boxplot or bar chart isn't enough—we need to show how one variable's effect depends on the levels of another. Grouped (or clustered) plots do exactly that: they organize multiple boxes or bars side by side for easy comparison within each group.

In the grammar of graphics (which underlies ggplot2), there's a conceptual difference between bar charts and column charts, even though they look almost identical:

Plot type	What it shows	Typical geom in ggplot2	Data source
Bar chart	Counts (frequencies) — how many cases fall into each category	geom_bar()	Raw (unaggregated) data
Column chart	Summary statistics — like means, proportions, or totals you've already computed	geom_col()	Pre-summarized data

Why this matters

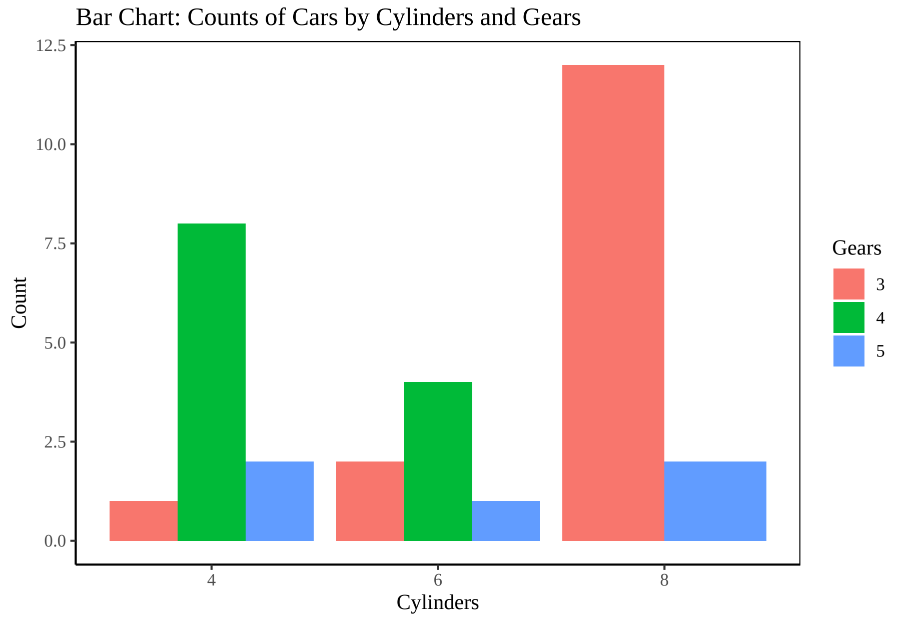
geom_bar() counts observations automatically using stat = "count" — it's designed for showing frequencies (e.g., number of cars per cylinder type).

geom_col() assumes the data already contains the summary values (e.g., mean horsepower per cylinder) and plots those directly with stat = "identity".

A grouped bar chart displays frequencies as a function of category

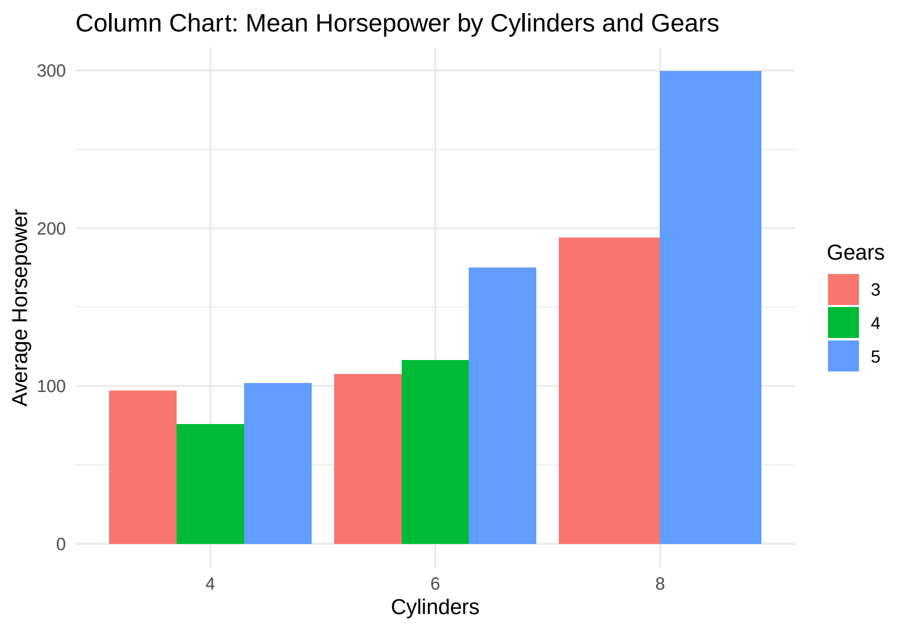
For example, let's say we want a count of the number of cars as a function of the number of cylinders and the number of gears. . .

```r
# Bar chart: counts of cars by cylinders and gears
ggplot(mtcars, aes(x = factor(cyl), fill = factor(gear))) +
  geom_bar(position = "dodge") +
  labs(title = "Bar Chart: Counts of Cars by Cylinders and Gears",
       x = "Cylinders", y = "Count", fill = "Gears") +
  theme_apa()
```



A grouped column plot (bar chart) uses the same idea, but instead of counting frequencies, it displays summary statistics like means or counts.

```
# Summarize mtcars by cylinder and gear
mtcars_summary <- mtcars %>%
  mutate(cyl = factor(cyl), gear = factor(gear)) %>%
  group_by(cyl, gear) %>%
  summarise(
    mean_hp = mean(hp),
    .groups = "drop"
  )
# Column chart: precomputed mean horsepower
ggplot(mtcars_summary, aes(x = cyl, y = mean_hp, fill = gear)) +
  geom_col(position = "dodge") +
  labs(title = "Column Chart: Mean Horsepower by Cylinders and Gears",
       x = "Cylinders", y = "Average Horsepower", fill = "Gears") +
  theme_minimal()
```



For example, Let's say we want to compare average horsepower by the number of cylinders (cyl) and number of gears (gear). Note, the example below also illustrates how to compute a 95% CI for the error bars.

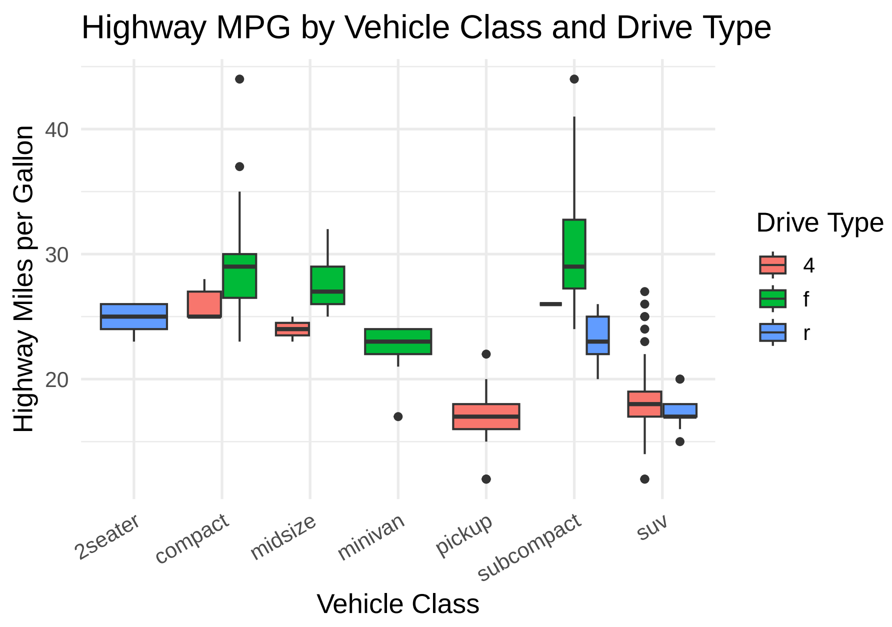
Bars are grouped by cylinder count, with each bar inside a group representing a gear category. This makes it easy to see, for instance, that cars with 8 cylinders and 5 gears have the highest average horsepower.

In both cases, the key idea is that x defines the primary grouping, and fill defines the secondary grouping within each x-category. The position = "dodge" argument tells ggplot2 to place the elements side by side instead of stacking them. Grouped plots are very powerful for visualizing interactions—situations where the relationship between two variables changes depending on a third. They turn abstract summaries into something intuitive: patterns you can see at a glance.

Note that you can group other kinds of plots as well. A grouped boxplot displays the distribution of a numeric variable for every combination of two factors.

For example, let's say we wanted to compare the highway gas mileage (MPG) across various classes of cars and drivetrains using the `mpg` dataset containing fuel economy data from 1999 to 2008 for 38 popular models of cars.

```r
ggplot(mpg, aes(x = class, y = hwy, fill = drv)) +
  geom_boxplot(position = position_dodge(width = .8)) +
  labs(
    title = "Highway MPG by Vehicle Class and Drive Type",
    x = "Vehicle Class",
    y = "Highway Miles per Gallon",
    fill = "Drive Type"
  ) +
  theme_minimal(base_size = 14) +
  theme(axis.text.x = element_text(angle = 30, hjust = 1))
```



Each group of boxes (by class) contains smaller boxes for different drv values, showing the distribution of highway MPG for front-wheel, rear-wheel, and 4-wheel drive vehicles.

Note about the position argument: In ggplot2, every geometric layer (like geom_col(), geom_bar(), geom_boxplot(), or geom_errorbar()) has a position adjustment that determines how elements that share the same x-value are placed.

By default, bars or boxes with the same x position are stacked on top of each other (position = "stack"). The position_dodge() adjustment instead tells ggplot2 to place them side by side — "dodging" them horizontally.

In the geom_col() and geom_bar() examples, we simply used the argument `position = "dodge"` in order to let ggplot know that we wanted the elements within each group to appear side by side. In the geom_boxplot() example, we took even more control by using the `position = position_dodge(width = .8)` argument, which uses the position_dodge() function to control the spacing between elements within each x-category.

The width argument controls how far apart groups within the same x-category are spread. It's expressed in the same coordinate units as the x-axis, relative to the width of each bar (or box). So…

- position_dodge(width = 1) –> bars or boxes fully separated (no overlap).
- position_dodge(width = 0.5) –> tighter spacing, some overlap.
- position_dodge(width = 0.8) (typical default) –> balanced spacing, tidy grouping.

Error Bars

Error bars are not just decorative elements on a chart, they are visual tools for showing uncertainty around an estimate. Whenever we plot sample statistics—like a mean, proportion, or regression coefficient—we're summarizing data that came from a sample, not an entire population. Error bars remind the reader that our estimate could have varied if we had drawn a different sample.

Each bar (or whisker) typically represents a range such as a standard deviation (SD), standard error (SE), or a confidence interval (CI) around the mean:

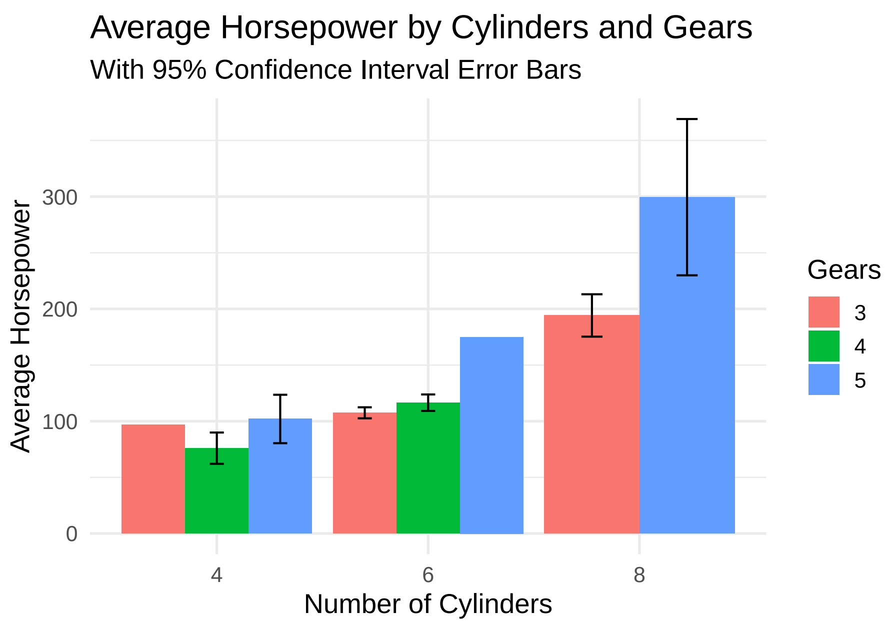
- SD → shows spread of individual data points.
- SE → shows precision of the mean (smaller with larger n).
- 95% CI → shows an interval likely to contain the true population mean.

When error bars are short, we have a more precise estimate; when they are long, there's greater uncertainty. However, overlapping error bars do not automatically mean "no difference" — they simply indicate that the uncertainty ranges overlap, which can happen even with statistically significant effects.

In ggplot2, error bars are often added with geom_errorbar() and paired with grouped bar or point charts to communicate both central tendency and variability:

Let's recompute the summary of mean horsepower as a function of the number of cylinders and gears but this time include measures of uncertainty and then re-plot the column chart including error bars reflecting the 95% CI.

```r
# Summarize mtcars by cylinder and gear
mtcars_summary <- mtcars %>%
  mutate(cyl = factor(cyl), gear = factor(gear)) %>%
  group_by(cyl, gear) %>%
  summarise(
    mean_hp = mean(hp),
    sd_hp   = sd(hp),
    n       = n(),
    se      = sd_hp / sqrt(n),               # standard error
    ci      = 1.96 * se,                     # 95% CI assuming normality
    lower   = mean_hp - ci,                  # Lower bound of CI
    upper   = mean_hp + ci,                  # Upper bound of CI
    .groups = "drop"
  )

# Plot with error bars
ggplot(mtcars_summary, aes(x = cyl, y = mean_hp, fill = gear)) +
  geom_col(position = position_dodge(width = 0.9)) +
  geom_errorbar(
    aes(ymin = lower, ymax = upper),
    position = position_dodge(width = 0.9),
    width = 0.2,
    color = "black"
  ) +
  labs(
    title = "Average Horsepower by Cylinders and Gears",
    subtitle = "With 95% Confidence Interval Error Bars",
    x = "Number of Cylinders",
    y = "Average Horsepower",
    fill = "Gears"
  ) +
  theme_minimal(base_size = 14)
```



Note that our level of uncertainty is different across the grouped factors. We have relatively precise estimates for 6-cylinder cars with 3 and 4 gears, but less precise estimates for 8-cylinder cars with 3 and 5 gears as indicated by the longer error bars.

Bubble Charts

A bubble chart is an extension of the scatterplot that adds a third quantitative variable by mapping it to the size (and sometimes color) of each point.

While scatterplots show how two variables are related (x and y), bubble charts can reveal multivariate relationships—for instance, how test performance changes with both stress and study time, or how reaction time depends on age and accuracy.

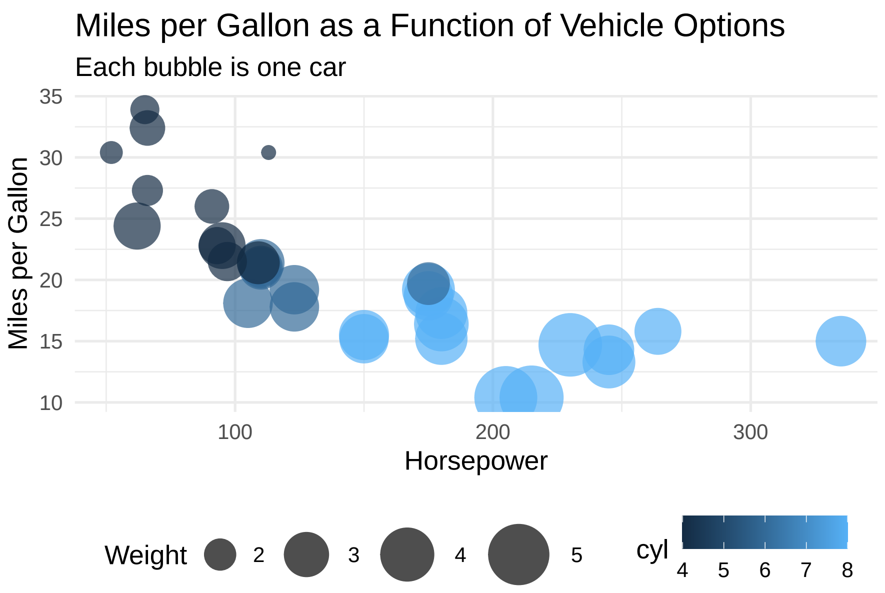
In bubble charts, each bubble's:

- x-position → represents one variable (predictor)
- y-position → represents another variable (outcome)
- size → encodes a third continuous measure (e.g., sample size, variability, or confidence)
- color (optional) → can encode a fourth, categorical grouping

Bubble charts are especially helpful in psychology or social-science contexts, where you often want to show how individual differences (age, stress level, motivation, etc.) jointly influence an outcome variable. Used carefully—with clear legends and non-overlapping points—they can turn a flat scatterplot into a richer multivariate story.

Let's consider another example from the mtcars dataset. The objective is to understand the relationships between horsepower (hp), miles per gallon (mpg), the weight of the car (wt), and the number of cylinders (cyl).

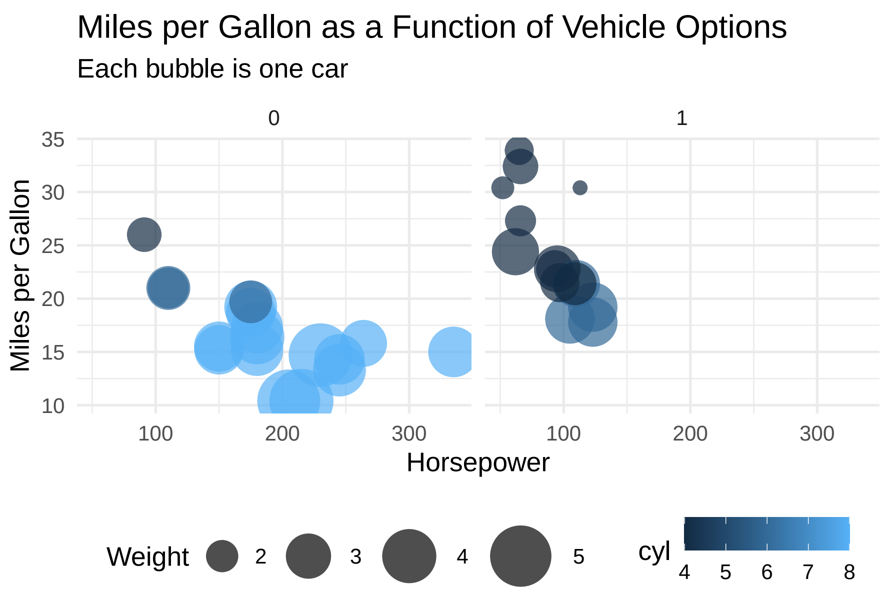
```r
# Create the bubble chart
ggplot(mtcars, aes(
  x = hp,
  y = mpg,
  size = wt,
  color = cyl
)) +
  geom_point(alpha = 0.7) +
  scale_size_continuous(range = c(3, 15), name = "Weight") +
  labs(
    title = "Miles per Gallon as a Function of Vehicle Options",
    subtitle = "Each bubble is one car",
    x = "Horsepower",
    y = "Miles per Gallon"
  ) +
  theme_minimal(base_size = 14) +
  theme(legend.position = "bottom") #Note that you can change the position of the legend
```



It looks like a vehicle's horsepower is negatively correlated with its fuel efficiency, but positively correlated with the car's weight and the number of cylinders.

We can even add additional complexity by faceting a 5th, categorical variable like the organization of the cylinders. For example, let's facet the plot by `vs`, which indicates whether or not the cylinders are arranged in a v-shape (vs = 0) or in a straight line (vs = 1).

```
# Create the bubble chart
ggplot(mtcars, aes(
  x = hp,
  y = mpg,
  size = wt,
  color = cyl
)) +
  geom_point(alpha = 0.7) +
  scale_size_continuous(range = c(3, 15), name = "Weight") +
  labs(
    title = "Miles per Gallon as a Function of Vehicle Options",
    subtitle = "Each bubble is one car",
    x = "Horsepower",
    y = "Miles per Gallon"
  ) +
  theme_minimal(base_size = 14) +
  theme(legend.position = "bottom") + #Note that you can change the position of the legend
  facet_wrap(~vs)
```



Correlograms

A correlogram is a visual map of the relationships between several quantitative variables, displaying the correlation coefficients (typically Pearson's r) in a grid. Each cell represents the strength and direction of the relationship between two variables—positive correlations appear in one color (often blue), negatives in another (often red), and the intensity or size of the symbol indicates magnitude.

Correlograms make it easy to spot patterns: clusters of strongly related variables, potential redundancy among measures, or distinct dimensions emerging from psychological scales. In psychology, they're often used to inspect questionnaire items, trait measures, or test batteries before running factor analysis.

For this example, let's try using a dataset from the `psych` package (by William Revelle). The bfi dataset is particularly good for a psychological-style bubble plot because it contains real questionnaire responses across major personality dimensions, age, and gender.

```r
# Install/load psych and corrplot if needed
if (!require(psych)) install.packages("psych")
```

```
## Loading required package: psych

##
## Attaching package: 'psych'

## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```

```r
if (!require(corrplot)) install.packages("corrplot")
```

```
## Loading required package: corrplot

## corrplot 0.92 loaded
```

```r
library(psych)
library(corrplot
        )
# Inspect the data
?psych::bfi
head(bfi)
```

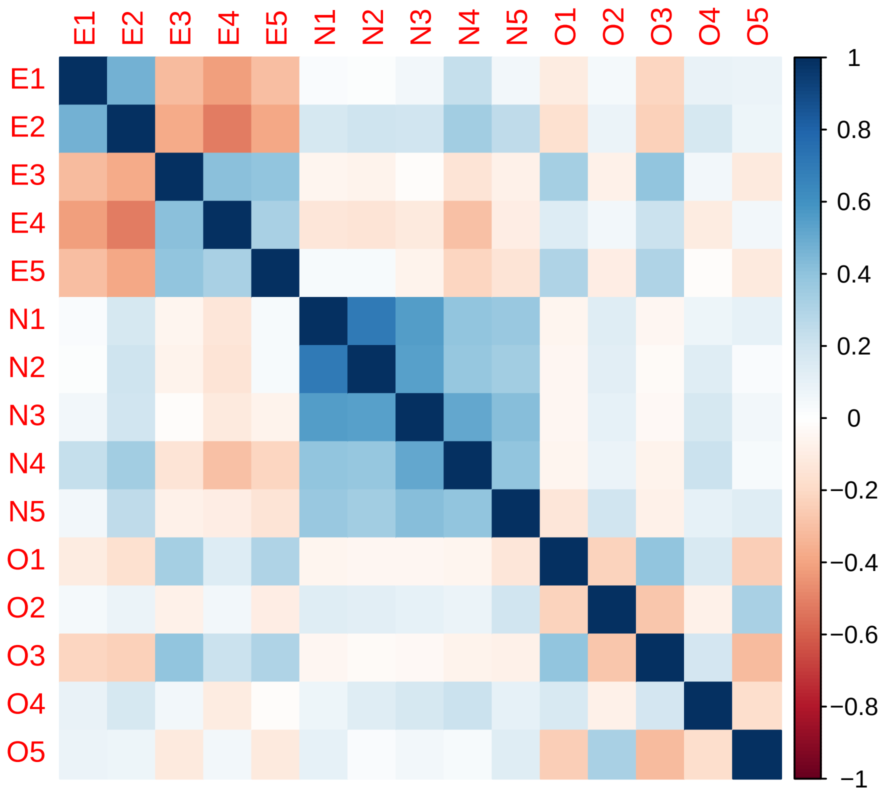
```
##       A1 A2 A3 A4 A5 C1 C2 C3 C4 C5 E1 E2 E3 E4 E5 N1 N2 N3 N4 N5 O1 O2 O3 O4
## 61617  2  4  3  4  4  2  3  3  4  4  3  3  3  4  4  3  4  2  2  3  3  6  3  4
## 61618  2  4  5  2  5  5  4  4  3  4  1  1  6  4  3  3  3  3  5  5  4  2  4  3
## 61620  5  4  5  4  4  4  5  4  2  5  2  4  4  4  5  4  5  4  2  3  4  2  5  5
## 61621  4  4  6  5  5  4  4  3  5  5  5  3  4  4  4  2  5  2  4  1  3  3  4  3
## 61622  2  3  3  4  5  4  4  5  3  2  2  2  5  4  5  2  3  4  4  3  3  3  4  3
## 61623  6  6  5  6  5  6  6  6  1  3  2  1  6  5  6  3  5  2  2  3  4  3  5  6
##       O5 gender education age
## 61617  3      1        NA  16
## 61618  3      2        NA  18
## 61620  2      2        NA  17
## 61621  5      2        NA  17
## 61622  3      1        NA  17
## 61623  1      2         3  21
```

The bfi dataset from the psych package contains 25 self-report items measuring the Big Five personality traits:

- Extraversion (E1–E5)
- Agreeableness (A1–A5)
- Conscientiousness (C1–C5)
- Neuroticism (N1–N5)
- Openness (O1–O5)

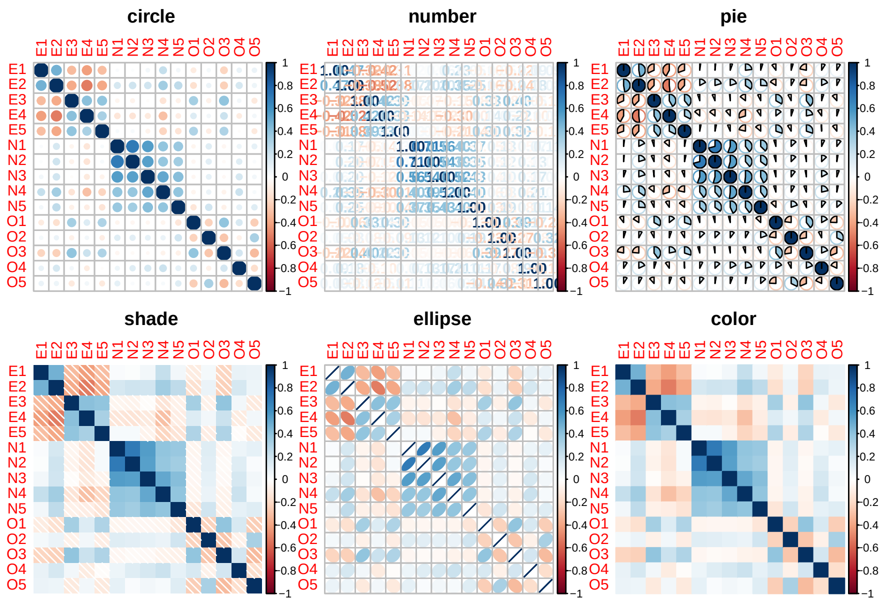
Let's visualize how these traits relate to one another using a correlogram.

```r
# Select the 25 personality items
bfi_items <- bfi %>%
  select(E1:O5) %>%
  na.omit()

# Compute the correlation matrix
bfi_cor <- cor(bfi_items)

# Basic correlogram
corrplot(bfi_cor, method = "color")
```



The corrplot package has lots of customization options that make correlation matrices not only informative but visually engaging. Let's start with the different corrplot "methods". Different methods emphasize correlation magnitude (circle size or ellipse flattening) or direction (color).

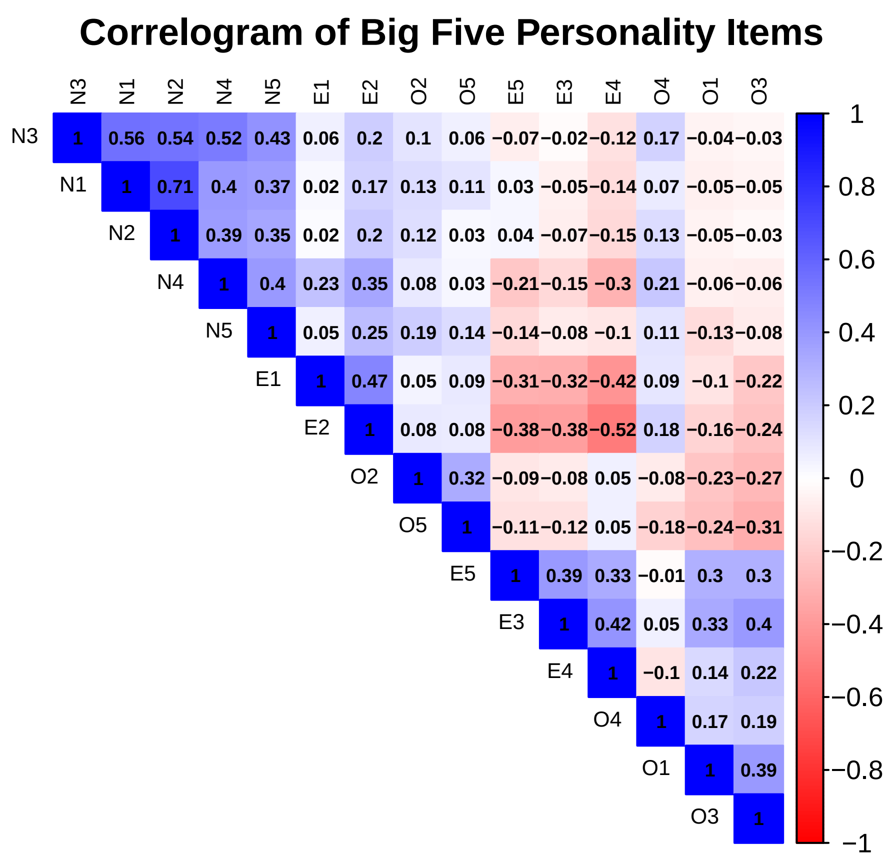
```r
par(mfrow = c(2,3))  # put plots in a 2x3 grid
corrplot(bfi_cor, method = "circle", title = "circle", mar = c(0,0,2,0))
corrplot(bfi_cor, method = "number", title = "number", mar = c(0,0,2,0))
corrplot(bfi_cor, method = "pie", title = "pie", mar = c(0,0,2,0))
corrplot(bfi_cor, method = "shade", title = "shade", mar = c(0,0,2,0))
corrplot(bfi_cor, method = "ellipse", title = "ellipse", mar = c(0,0,2,0))
corrplot(bfi_cor, method = "color", title = "color", mar = c(0,0,2,0))
```



```r
par(mfrow = c(1,1))
```

If you don't have any sense of how the variables are "clustered" together, corrplot will automatically arrange your data based on a

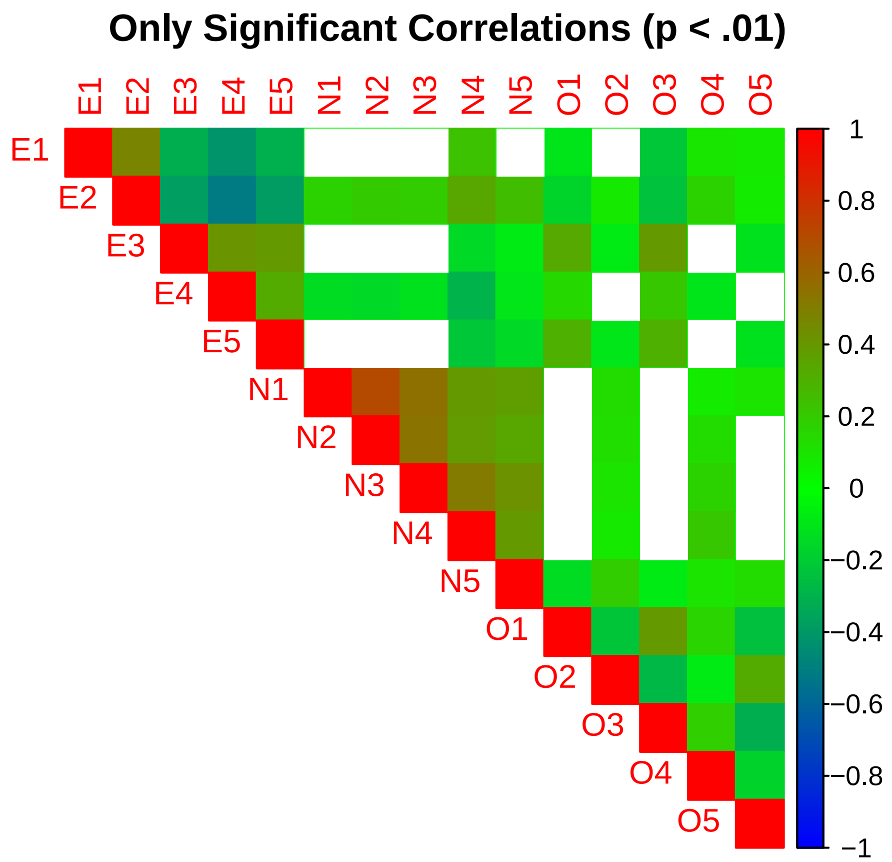
```r
# Basic correlogram
corrplot(bfi_cor,
         method = "color",            # colored tiles
         type = "upper",              # show upper triangle (full or lower)
         order = "hclust",         # groups correlated variables
         addrect = 5,              # draw rectangles around 5 clusters
         tl.col = "black",            # text label color
         tl.cex = 0.7,                # text label size
         col = colorRampPalette(c("red", "white", "blue"))(200), # Change the color pallette
         addCoef.col = "black",       # print r values
         number.cex = 0.6,
         title = "Correlogram of Big Five Personality Items",
         mar = c(0,0,2,0))
```



Highlight correlations with p-values

You can also highlight specific correlations based on their associated p-value.

```r
# Compute correlation test results (r and p)
corr_test <- psych::corr.test(bfi_items) # Use corr.test from psych package
p_mat <- corr_test$p  # extract p-values

# Plot only significant correlations
corrplot(bfi_cor, p.mat = p_mat, sig.level = 0.01, insig = "blank",
         method = "color",
         type = "upper",
         col = colorRampPalette(c("blue", "green", "red"))(200),
         title = "Only Significant Correlations (p < .01)",
         mar = c(0,0,2,0))
```

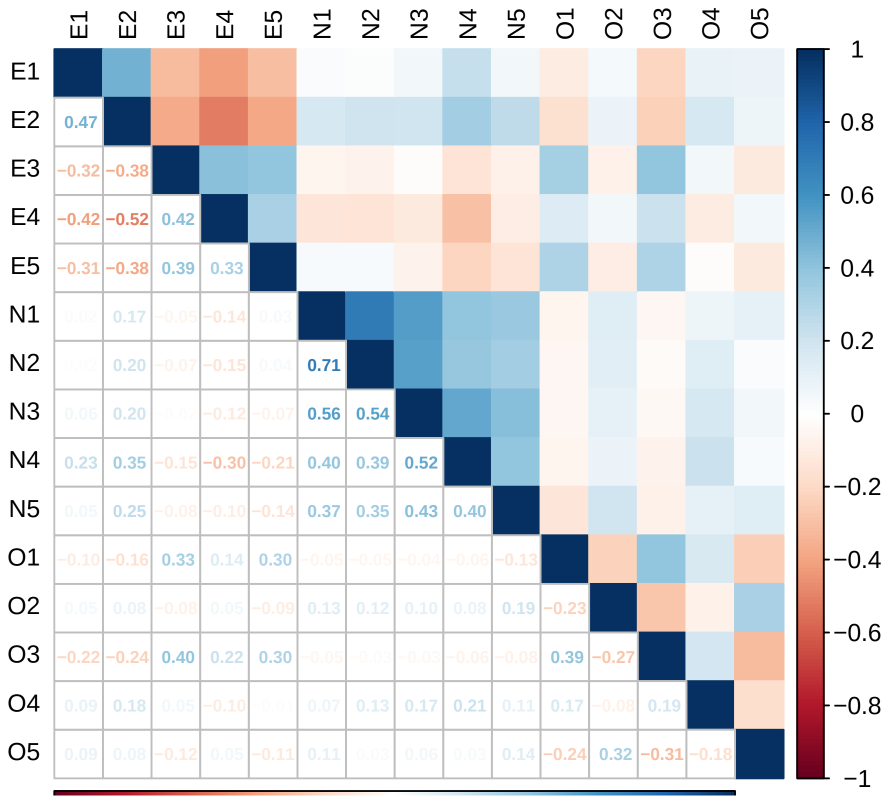


```
# The argument insig = "blank" hides non-significant cells, making meaningful relationships stand out.
```

Combine upper and lower panels

You can visualize different information on each half of the matrix.

```
# Upper: colored circles, Lower: correlation coefficients
corrplot(bfi_cor, method = "color", type = "upper", order = "original",
         tl.pos = "lt", tl.col = "black", tl.cex = 0.8)

corrplot(bfi_cor, method = "number", type = "lower", tl.pos = 'n', add = TRUE, diag = FALSE, number.cex
```



```
# tl.pos='n' --> no labels
```

This combination gives both a colorful overview and exact r values at once.

Heatmaps

A heatmap is a graphical display where values in a data matrix are represented by color intensity. The correlogram is a form of heatmap, however, in traditional heatmaps, rows and columns typically correspond to variables, observations, or experimental conditions. The color of each cell encodes the magnitude of the value it represents (e.g., low = blue, high = red).

Heatmaps are great for identifying clusters or patterns in correlations or responses, differences across subjects or conditions, and groups of variables that behave similarly.

In psychology, heatmaps are often used to visualize:

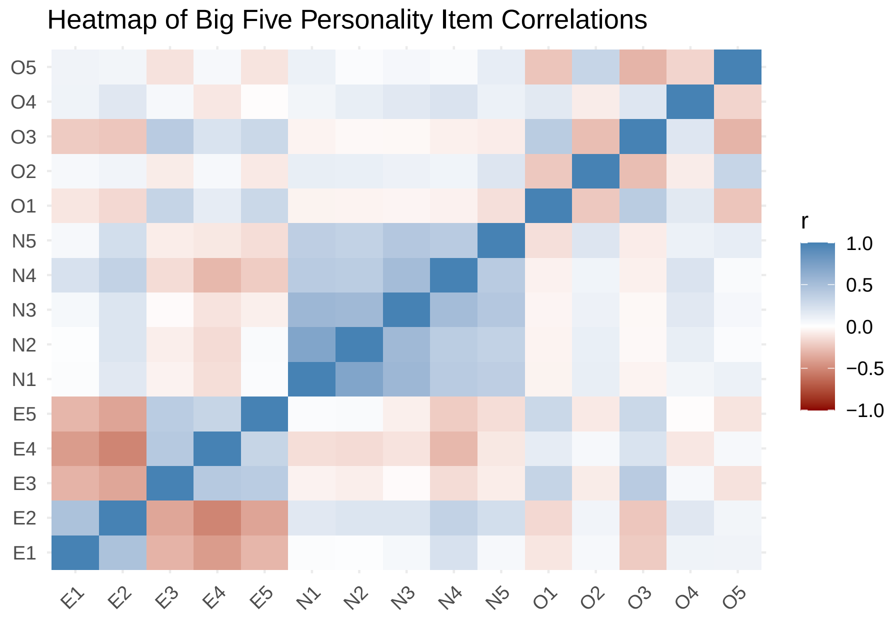
- Correlation matrices (relationships among traits, brain regions, questionnaire items)
- Response patterns (e.g., item × participant matrices)
- Time-series data (e.g., activation over time × region)

Simple Heatmap of Correlations

Let's visualize how the 25 Big Five personality items from psych::bfi relate to one another.

```
# Convert to long (tidy) format for ggplot
cor_long <- as.data.frame(as.table(bfi_cor))

# Plot as heatmap
ggplot(cor_long, aes(Var1, Var2, fill = Freq)) +
  geom_tile() +
```

```r
  scale_fill_gradient2(low = "darkred", mid = "white", high = "steelblue",
                       midpoint = 0, limit = c(-1, 1)) +
  labs(
    title = "Heatmap of Big Five Personality Item Correlations",
    x = NULL, y = NULL, fill = "r"
  ) +
  theme_minimal(base_size = 12) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Choropleth Map

A choropleth map displays data values across geographic regions — such as countries, states, or counties — by color-coding each area according to a numeric variable. They're designed to answer questions like "Where is this variable higher or lower?" at a glance.

Each polygon (region) is shaded on a color scale, so darker or more saturated colors typically indicate higher values. In psychology, education, and social science, choropleth maps can reveal geographic patterns in well-being, mental health, education, or demographic factors. For instance, darker regions might show higher stress scores or greater life satisfaction.

Under the hood, a choropleth map combines:

- Spatial data (the map geometry — polygons for regions)
- Attribute data (the variable you want to visualize)
- A join between them by region name or code

In R, we usually build these maps with ggplot2 (using geom_polygon() or geom_sf()) or helper packages like maps, map_data(), or sf.

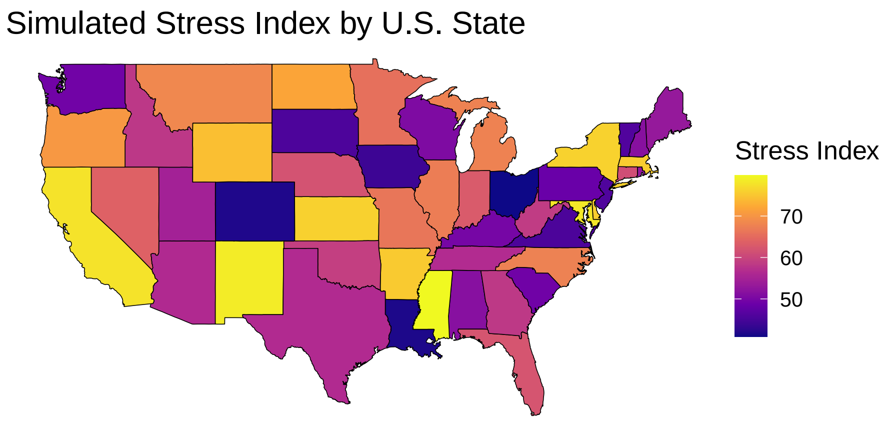
A US Map using ggplot

```r
# Install (if required) and load maps package
if (!require(maps)) install.packages("maps")
```

```
## Loading required package: maps
```

```r
library(maps)

# Get US map data
states_map <- map_data("state")

# Simulate a "stress index" variable for each state
set.seed(123)
state_data <- data.frame(
  region = tolower(state.name), # get state names from the state dataset
  stress_index = runif(50, 40, 80) # get 50 random numbers from uniform dist. between 40 & 80
)

# Merge the numeric variable with the map
us_map <- left_join(states_map, state_data, by = "region")

# Choropleth map
ggplot(us_map, aes(long, lat, group = group, fill = stress_index)) +
  geom_polygon(color = "black", size = 0.2) + # Border between states
  coord_fixed(1.3) +  # sets a fixed aspect ratio between the x and y axes
  scale_fill_viridis_c(option = "plasma", name = "Stress Index") +
  labs(
    title = "Simulated Stress Index by U.S. State",
  ) +
  theme_void(base_size = 14)
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

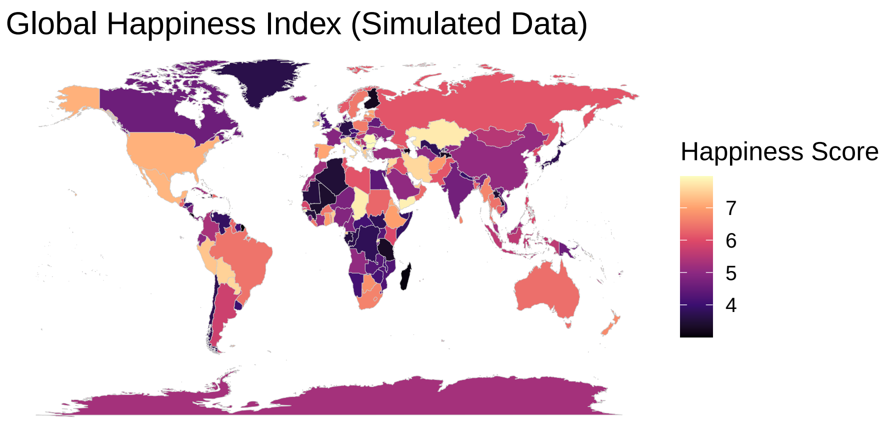


Each state's fill color corresponds to its simulated "stress index," illustrating higher stress concentrated in certain regions.

A World map using ggplot

```r
world_map <- map_data("world")

# Simulated "happiness" score by country
set.seed(123)
```

```
country_data <- data.frame(
  region = unique(world_map$region),
  happiness = runif(length(unique(world_map$region)), 3, 8)
)

world_data <- left_join(world_map, country_data, by = "region")

ggplot(world_data, aes(long, lat, group = group, fill = happiness)) +
  geom_polygon(color = "gray80", size = 0.1) +
  scale_fill_viridis_c(option = "magma", name = "Happiness Score") +
  coord_fixed(1.3) +
  labs(
    title = "Global Happiness Index (Simulated Data)",
  ) +
  theme_void(base_size = 14)
```



Lines of fit

geom_smooth(): Go in-depth with regression models. Explain the different methods (method = "lm", "glm", "loess") and how to interpret the confidence intervals (se = TRUE).

The 2d Density Plot

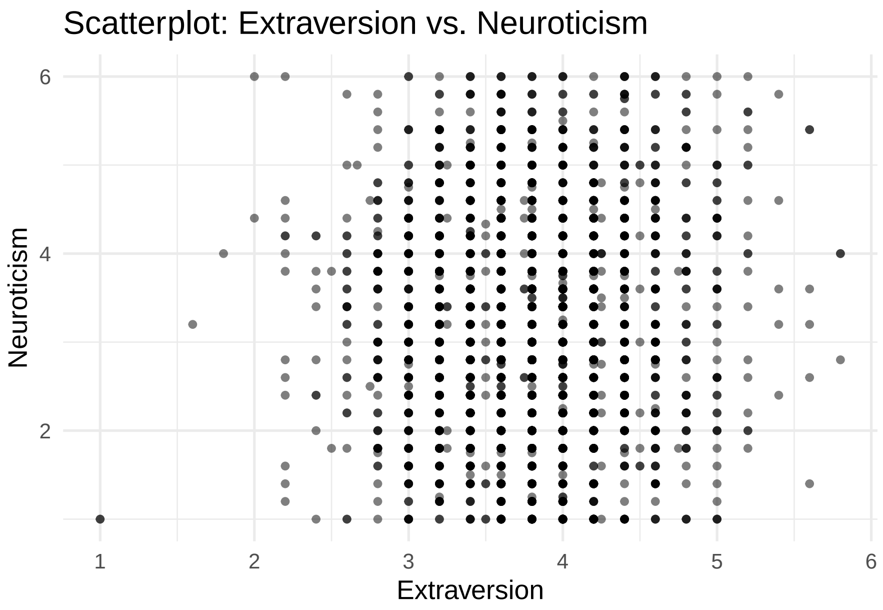
geom_density_2d() and geom_hex(): Introduce 2D density plots and hexbin plots as powerful alternatives to scatter plots for very large datasets where overplotting is a major issue.

A good example can be found in the bfi dataset from the psych package.

```
# Load the bfi data (Big Five Inventory responses)
data(bfi)

# Compute mean scores for each trait per participant
bfi_person <- bfi %>%
  mutate(
    Extraversion = rowMeans(select(., E1:E5), na.rm = TRUE), # Compute extraversion average
    Neuroticism  = rowMeans(select(., N1:N5), na.rm = TRUE), # Compute neuroticism average
    gender = factor(gender,
                    levels = c(1, 2),
                    labels = c("Male", "Female")) # Add labels for gender
  ) %>%
  filter(!is.na(Extraversion), !is.na(Neuroticism), !is.na(age), !is.na(gender))
```

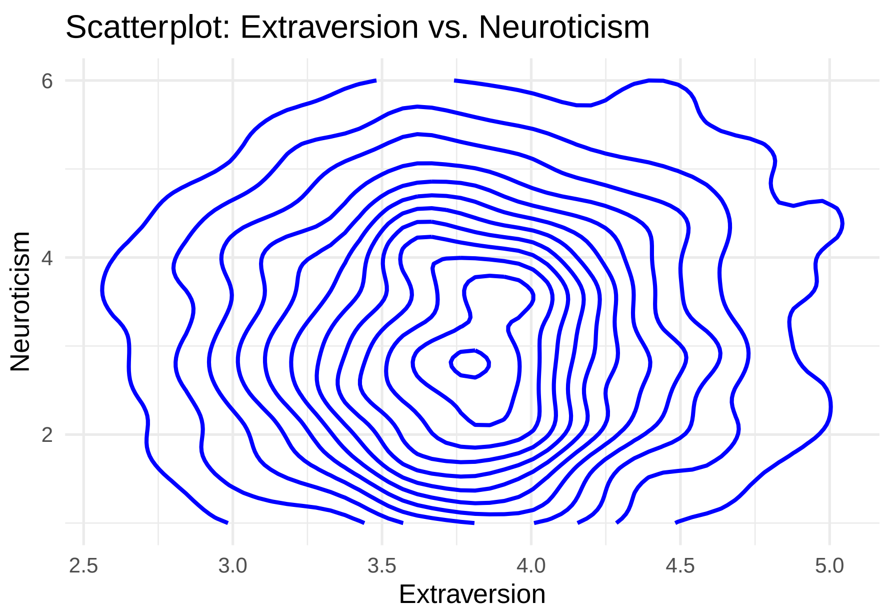
Because there are so many participants, the scatterplot does not give a clear picture of the data.

```
ggplot(bfi_person, aes(x = Extraversion, y = Neuroticism)) +
  geom_point(alpha = 0.5) +                          # raw data points
  labs(
    title = "Scatterplot: Extraversion vs. Neuroticism",
    x = "Extraversion",
    y = "Neuroticism"
  ) +
  theme_minimal(base_size = 14)
```



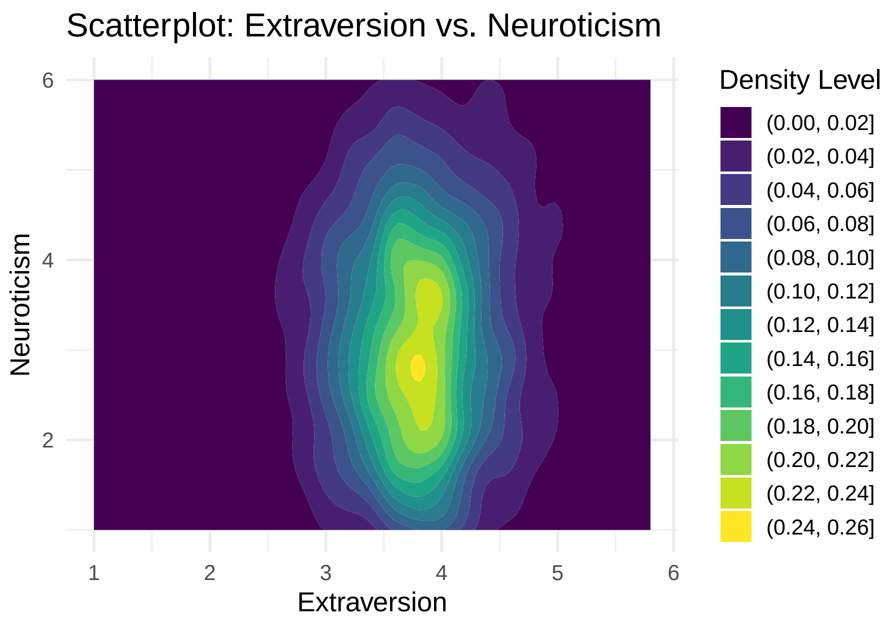
However, the 2D density plot nicely illustrates how data are concentrated over the 2D plane.

```
ggplot(bfi_person, aes(x = Extraversion, y = Neuroticism)) +
  geom_density_2d(color = "blue", size = 1) +        # contour lines
  labs(
    title = "Scatterplot: Extraversion vs. Neuroticism",
    x = "Extraversion",
    y = "Neuroticism"
  ) +
  theme_minimal(base_size = 14)
```

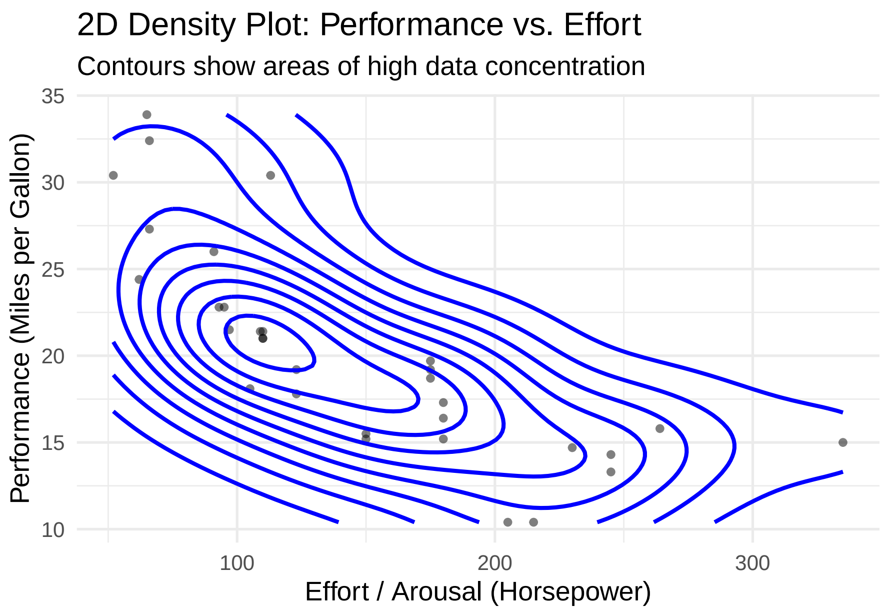


Even better is the filled 2D density.

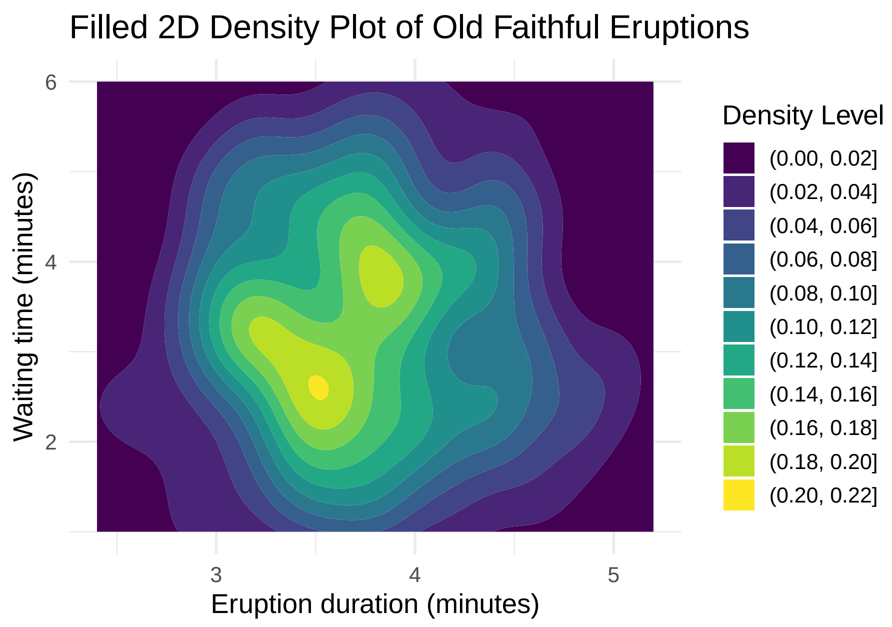
```r
ggplot(bfi_person, aes(x = Extraversion, y = Neuroticism)) +
  geom_density_2d_filled() +         # filled density
  scale_fill_viridis_d(name = "Density Level") +
  labs(
    title = "Scatterplot: Extraversion vs. Neuroticism",
    x = "Extraversion",
    y = "Neuroticism"
  ) +
  theme_minimal(base_size = 14)
```



```r
ggplot(mtcars, aes(x = hp, y = mpg)) +
  geom_point(alpha = 0.5) +                    # raw data points
  geom_density_2d(color = "blue", size = 1) +  # contour lines
  labs(
    title = "2D Density Plot: Performance vs. Effort",
    subtitle = "Contours show areas of high data concentration",
    x = "Effort / Arousal (Horsepower)",
    y = "Performance (Miles per Gallon)"
  ) +
  theme_minimal(base_size = 14)
```



```
# Load the bfi data (Big Five Inventory responses)
data(bfi)

# Compute mean scores for each trait per participant
bfi_person <- bfi %>%
  mutate(
    Extraversion = rowMeans(select(., E1:E5), na.rm = TRUE),
    Neuroticism  = rowMeans(select(., N1:N5), na.rm = TRUE),
    gender = factor(gender,
                    levels = c(1, 2),
                    labels = c("Male", "Female")) # Add labels for gender
  ) %>%
  filter(!is.na(Extraversion), !is.na(Neuroticism), !is.na(age), !is.na(gender), education %in% c(1))

ggplot(bfi_person, aes(Extraversion, Neuroticism)) +
  geom_density_2d_filled() +
  scale_fill_viridis_d(name = "Density Level") +
  labs(
    title = "Filled 2D Density Plot of Old Faithful Eruptions",
    x = "Eruption duration (minutes)",
    y = "Waiting time (minutes)"
  ) +
  theme_minimal(base_size = 14)
```



Saving High-resolution Images

When you've created a figure you want to use in a paper, presentation, or poster, it's important to save it at high resolution so it looks crisp and professional. In R, there are two main ways to do this:

1) ggsave() — designed for ggplot objects.

It automatically saves the last plot (or a specific one you name) and lets you control the size, units, and resolution (dpi).

```
ggsave("my_plot.png", width = 8, height = 6, units = "in", dpi = 300)
```

- dpi = 300 gives publication-quality resolution.
- You can also use formats like "pdf", "tiff", or "svg" for vector graphics.

2) Base graphics method — useful when the plot isn't from ggplot (like the corrplot() example).

With this method, you open a graphics device (e.g., png()), make the plot, and then close the device with dev.off():

```
png("bfi_correlogram_fancy.png", width = 1200, height = 1000, res = 150)
corrplot(bfi_cor, method = "color", order = "hclust", addrect = 5,
         col = colorRampPalette(c("darkred", "white", "navy"))(200),
         tl.col = "black", tl.cex = 0.7,
         title = "Fancy Big Five Correlogram")
dev.off()
```

```
## pdf
##   2
```

- width and height control the output size (in pixels by default).
- res = 150 sets resolution; increase to 300+ for publication-quality images.
- Replace png() with pdf(), tiff(), or jpeg() for other formats.

Animations (Under Construction)

```
# libraries:
#library(ggplot2)
#library(gganimate)
#install.packages('babynames')
#library(babynames)
#library(hrbrthemes)

# Keep only 3 names
#don <- babynames %>%
#  filter(name %in% c("Gabrielle", "Kathryn", "Samantha")) %>%
#  filter(sex=="F")

# Plot
#don %>%
#  ggplot( aes(x=year, y=n, group=name, color=name)) +
#    geom_line() +
#    geom_point() +
    #scale_color_brewer(palette = 'Pastel') +
#    ggtitle("Popularity of American names in the previous 30 years") +
#    theme_ipsum() +
#    ylab("Number of babies born") +
#    transition_reveal(year)



# Save at gif:
#anim_save("~/Desktop/labnames.gif")
```

TO-DO

Continuous Color Scales: Explain scale_color_gradient() (two-color) and scale_color_gradient2() (three-color, for diverging data) to map continuous variables to color.

Manual Scale Control: Reinforce scale_color_manual(), scale_shape_manual(), etc., but with an emphasis on creating named vectors for reproducibility, ensuring the right color/shape is always mapped to the right category.

Axis Transformations: Show how to transform axes using the trans argument within continuous scales (e.g., scale_y_continuous(trans = "log10")) to handle skewed data.

Advanced Labeling: Use the labels argument within scales to format axis text (e.g., labels = scales::percent or labels = scales::dollar).

tidytext: For visualizing text data, such as plotting word frequencies or sentiment scores using ggplot2.