SPEC REU R Resources: Basic Data Visualization with ggplot2

Alix Ziff, Gaea Morales, Zacahary Johnson, Claudia Salas Gimenez, Ben Graham

Summer 2024

Introduction

In this walkthrough, we build upon the concepts covered in the first part, focusing on creating more engaging and effective plots by applying custom aesthetics to make them publication-ready.

We'll explore essential tools and techniques in `ggplot2` that will enable you to produce polished, professional-quality visuals. Key topics include adjusting plot labels, customizing appearances, experimenting with color schemes, modifying line types and widths, and incorporating interactive faceting. Using data from the World Development Indicators, we'll analyze energy consumption trends over the past 25 years, and you'll also practice creating bar plots using survey data from the SPEC Lab's LEWIS registry.

Initial Setup

Begin by setting your working directory to the Training Data folder on your computer. Next, load the `tidyverse` and ggplot2 packages. Finally, load the dataset named `wdi_cleaned_part1.csv`, making sure to read string data as character types, just like in Walkthrough 1.

```
# Set working directory
#setwd("YourFolderPath")

# Load required libraries
library(tidyverse)
library(dplyr)
library(ggplot2)

# Load the data
dat <- read.csv("wdi_cleaned_part1.csv", stringsAsFactors = FALSE)
```

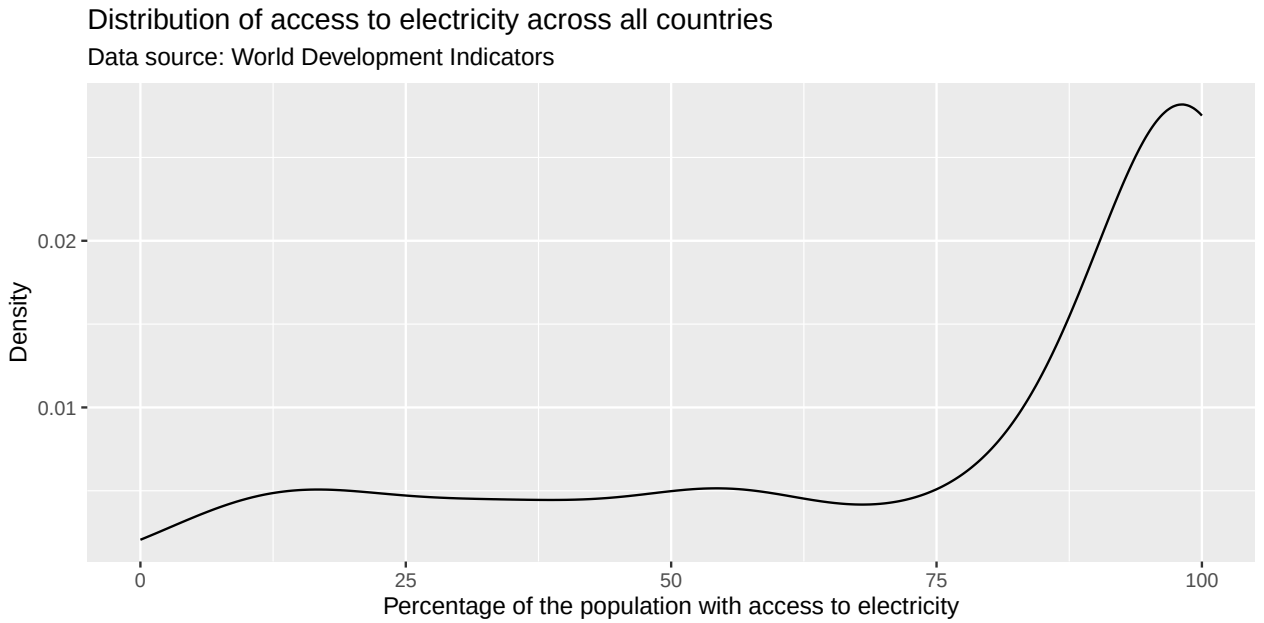
Remember to refer to the World Development Indicators data catalog for detailed descriptions of the indicators. Keep in mind that since these datasets have been modified for this walkthrough, the variable names may differ slightly from those in the original codebooks.

Label Plot

Effective labels are key to making data visualizations clear and understandable. In this section, we'll learn you how to add a clear title, include a subtitle that details the data source or relevant notes, and label the axes to guide the viewers through the data. We can specify titles and axes labels within the `labs()` function.

```
# Label the plot
ggplot(dat, aes(x = electricity_pop)) +
  geom_line(stat = "density") +
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
```

```
       x = "Percentage of the population with access to electricity",
       y = "Density")
```



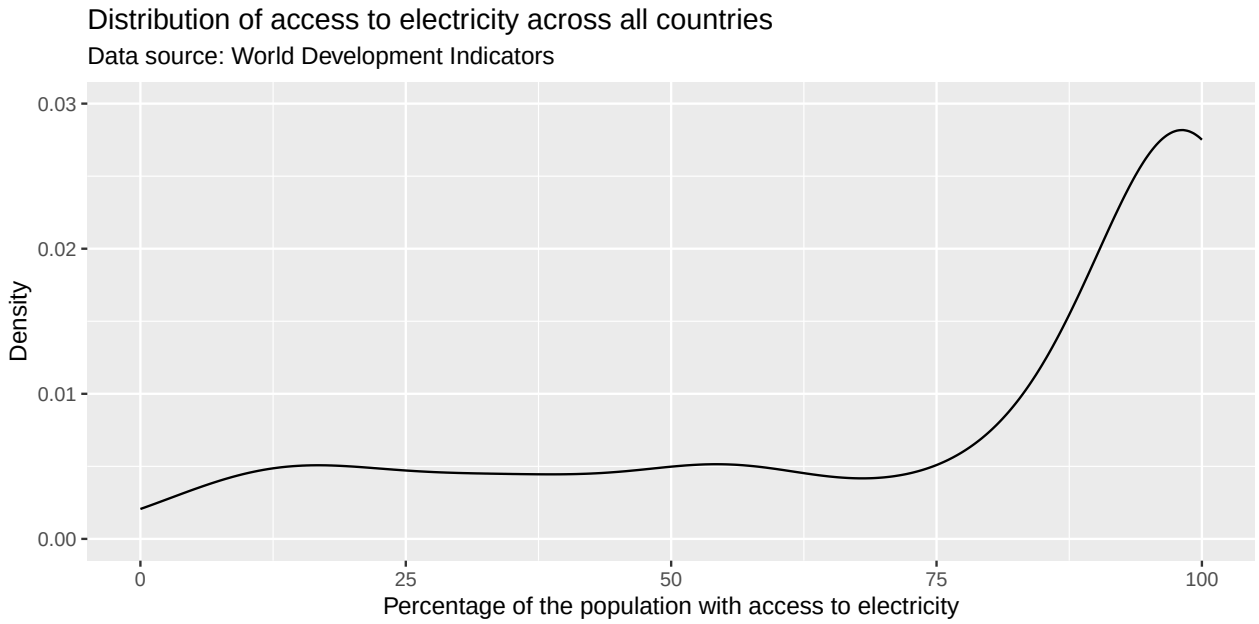

```
       # Add titles and labels to the plot
```

Customize Plot Appearance

Adjusting the Axis

By default, `ggplot()` adjusts the y-axis to start at approximately 0.2 to reduce the amount of empty space in the plot. This might confuse the viewer, as it visually underestimates the height of the density curve – in particular for lower values of the variable. We can manually adjust the range of the axes using the `coord_cartesian()` parameter.

```
# Adjust the axis
ggplot(dat, aes(x = electricity_pop)) +
  geom_line(stat = "density") +
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Percentage of the population with access to electricity",
       y = "Density") +
   # Add titles and labels to the plot
  coord_cartesian(ylim = c(0, 0.03))
```



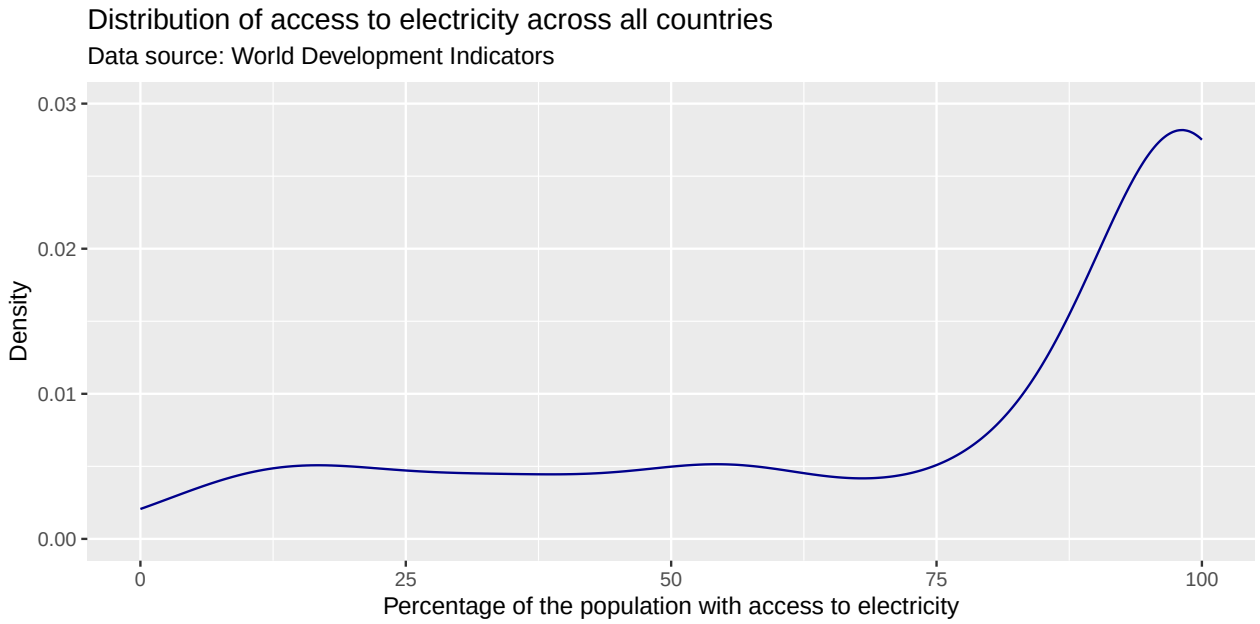

```
  # Adjust the y-axis from 0 to 0.03
```

Caution! You may see code in the Lab that uses `scale_y_continuous(limits = c(0, 0.03))` instead of `coord_cartesian(ylim = c(0, 0.03))`. Note that these are not the same. `coord_cartesian()` only adjusts the range of the axes (it "zooms" in and out), while `scale_y_continuous(limits = c())` subsets the data. For density plots, this does not make a difference. But there are other examples where it alters the actual shape of the graph, rather than just the part of the graph that is visible.

Color me!

We can further enhance our plot by experimenting with different colors and line types. The choice of color and line style can significantly affect the readability and aesthetic appeal of the graphs. Any changes to the appearance of the curve itself are made within the `geom_line()` argument, which specifies the geometric object to be plotted. `R` knows many colors by name; for a great overview see R colors cheatsheet.

```
# Adjust the color
ggplot(dat, aes(x = electricity_pop)) +
  geom_line(stat = "density", color = "darkblue") +
  # Add color to the curve
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Percentage of the population with access to electricity",
       y = "Density") +
   # Add titles and labels to the plot
  coord_cartesian(ylim = c(0, 0.03))
```

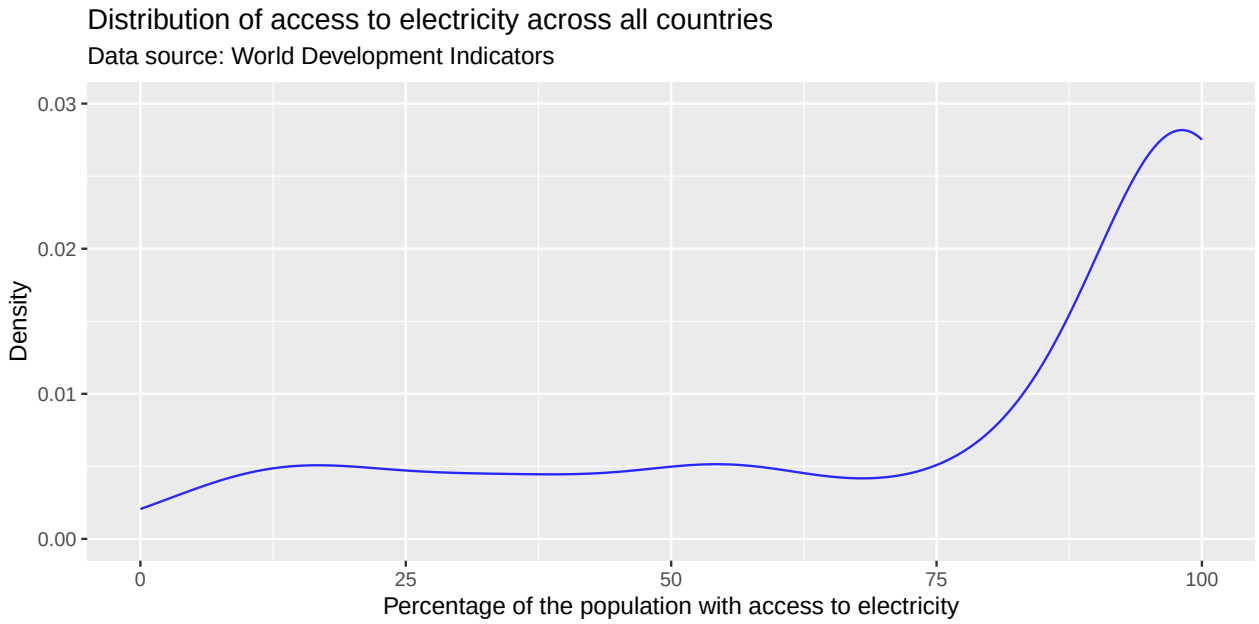



```
  # Adjust the y-axis from 0 to 0.03
```

We can also use hexadecimal or RGB (red, green, blue) strings to specify colors. There are plenty of online tools to pick colors and extract hexadecimal or RBG strings. For instance, ColorHexa is an online tool that allows you to specify a color name, hexadecimal, or RGB string, and returns information on color schemes, complementary colors, as well as alternative shades, tints, and tones. It also offers a color blindness simulator.

Suppose, I like the general tone of the darkblue color above, but am worried that it is a bit too dark for my plot. I could enter the color "darkblue" into the search field at ColorHexa and look for a brighter alternative. Now, suppose I really like the color displayed in the second tile from the left on the tints scale. I can extract this color's hexadecimal value of `#2727ff` by hovering over the tile of that color.

```
# Adjust the color using hexadecimal value
ggplot(dat, aes(x = electricity_pop)) +
  geom_line(stat = "density", color = "#2727ff") +
  # Add color to the curve
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Percentage of the population with access to electricity",
       y = "Density") +
  # Add titles and labels to the plot
  coord_cartesian(ylim = c(0, 0.03))
```



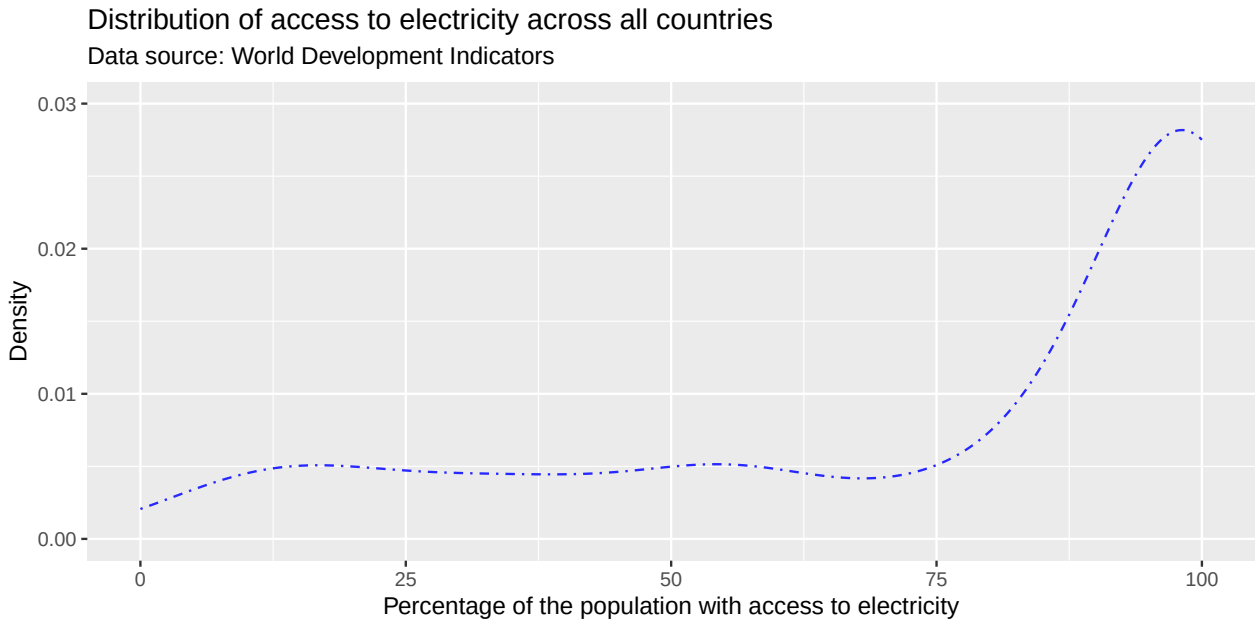

```
  # Adjust the y-axis from 0 to 0.03
```

Customize the Lines

We can also adjust the characteristics of the lines.

- **Line Type**:
  - Adjust the line type using the `linetype` parameter within `geom_line()`.
  - Different line types (e.g., solid, dashed, dotdash) can be specified to enhance visual distinction between multiple lines. For an overview of line types see Linetype Quick Reference.
  - Many academic journals will only accept graphs on a gray scale. This means that color will not be enough to differentiate the three lines. Ajusting the line type will help us with this and will also make the graph more color blind friendly.

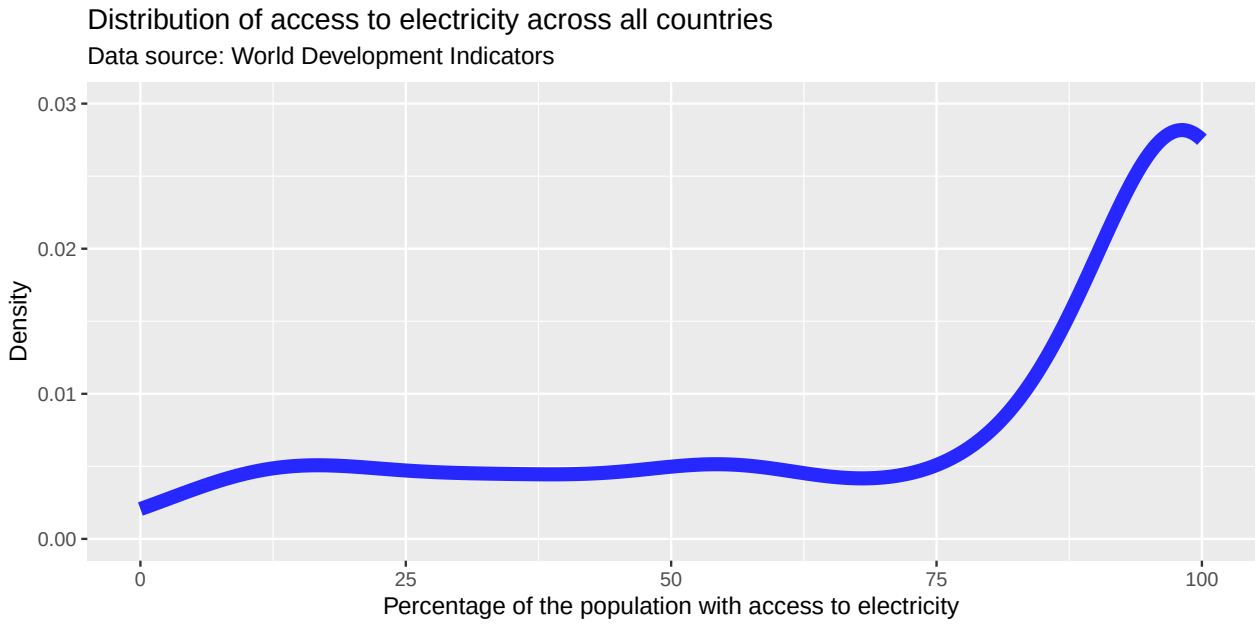
```
# Adjust the line type
ggplot(dat, aes(x = electricity_pop)) +
  geom_line(stat = "density", color = "#2727ff", linetype = "dotdash") +
  # Add color to the curve
  # Customize the line type
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Percentage of the population with access to electricity",
       y = "Density") +
  # Add titles and labels to the plot
  coord_cartesian(ylim = c(0, 0.03))
```




```
  # Adjust the y-axis from 0 to 0.03
```

- **Line Width**:
  - The width of the line can be controlled via the `size` parameter within `geom_line()`.
  - The `size` parameter is versatile, affecting line width in line plots and point size in scatter plots. Increasing the size value results in thicker lines, making them more prominent on the plot.

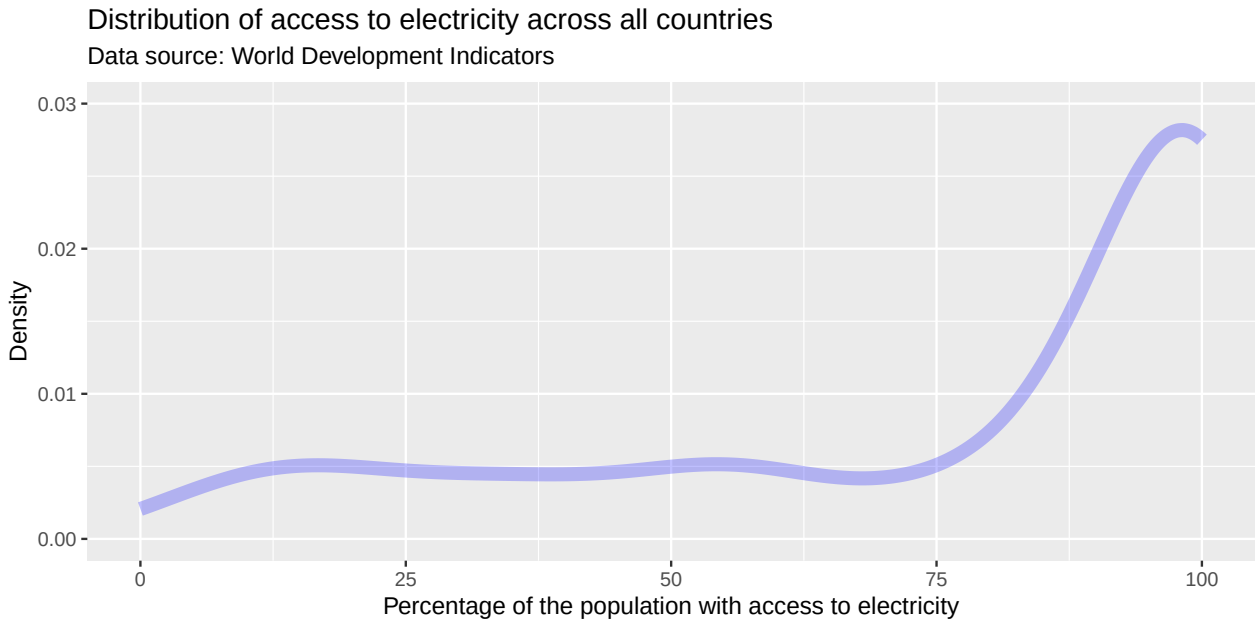
```
# Adjust the line width
ggplot(dat, aes(x = electricity_pop)) +
  geom_line(stat = "density", color = "#2727ff", size = 3) +
  # Add color to the curve
  # Customize the line size
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Percentage of the population with access to electricity",
       y = "Density") +
  # Add titles and labels to the plot
  coord_cartesian(ylim = c(0, 0.03))
```




```
  # Adjust the y-axis from 0 to 0.03
```

- **Line Opacity**:
  - Opacity of the lines can be adjusted using the `alpha` parameter, which is applicable to any geometric object.
  - The `alpha` value ranges from 0 (completely transparent) to 1 (completely opaque). Adjusting opacity is particularly useful when plotting multiple lines or objects on the same graph to reduce overplotting.

```
# Adjust the line opacity
ggplot(dat, aes(x = electricity_pop)) +
  geom_line(stat = "density", color = "#2727ff", size = 3, alpha = 0.3) +
  # Add color to the curve
  # Customize the line size and opecity
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Percentage of the population with access to electricity",
       y = "Density") +
  # Add titles and labels to the plot
  coord_cartesian(ylim = c(0, 0.03))
```

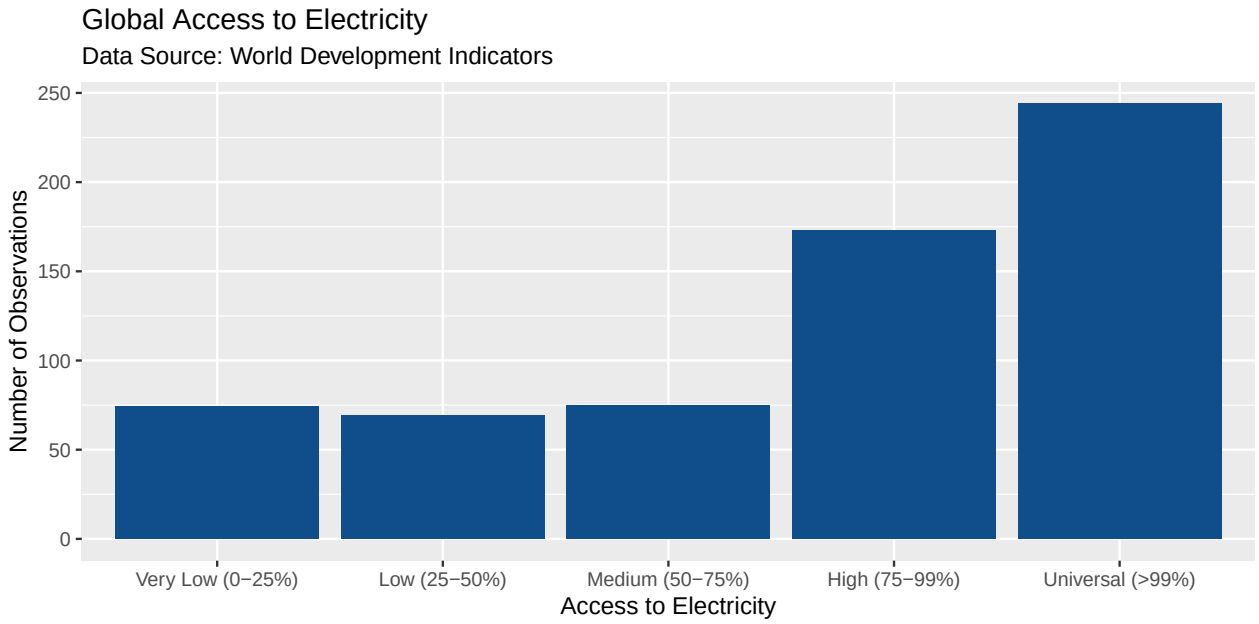



```
  # Adjust the y-axis from 0 to 0.03
```

Let's practice!

Let's revisit the bar plot from Walkthrough I in M4 Data Visualization I and enhance it using the commands we've learned.

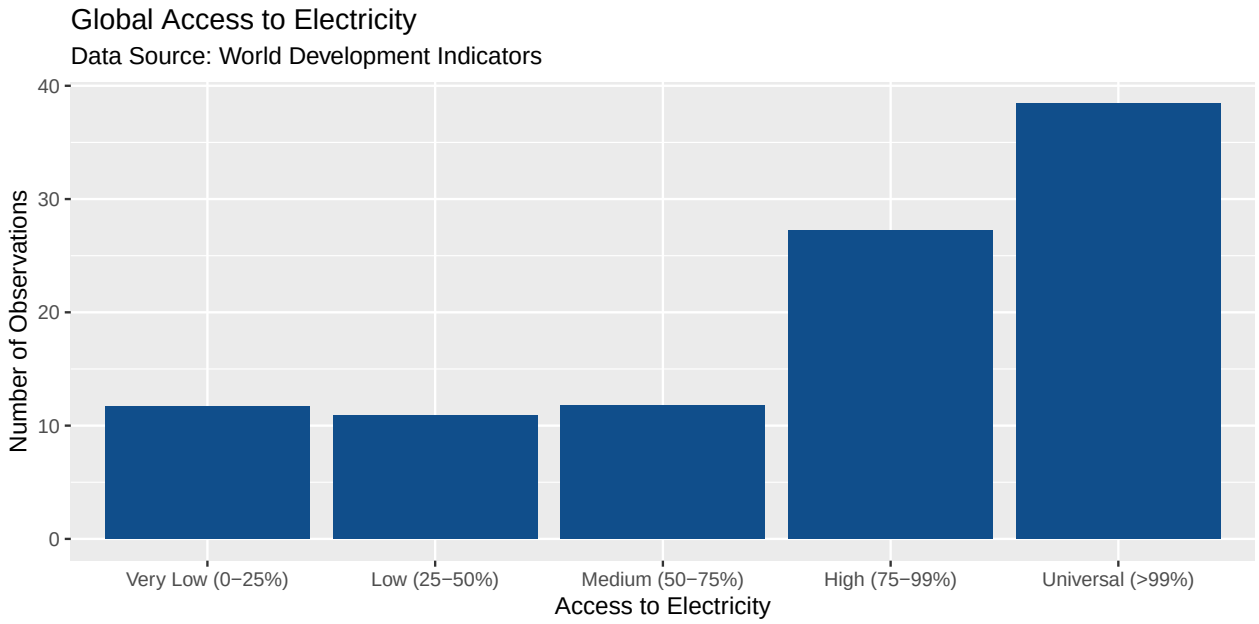
```
# Plot a bar plot
## Transform 'electricity_pop' from a continuous to an ordinal variable with specific
## breaks being (0, 25, 50, 75, 99, 100)
dat2 <- dat %>%
  mutate(electricity_pop_ord = cut(electricity_pop, breaks = c(0,25, 50, 75, 99, 100))) %>%
  filter(electricity_pop_ord != "NA")
  ## Remove any instances where 'electricity_pop_ord' is NA to clean up the data before
  ## plotting.

# Plot the graph
ggplot(dat2, aes(x = electricity_pop_ord)) +
  geom_bar(fill = "dodgerblue4") +
  # Add color to the bars
  labs(title = "Global Access to Electricity",
       subtitle = "Data Source: World Development Indicators",
       y = "Number of Observations",
       x = ("Access to Electricity")) +
    # Add titles and labels to the plot
      scale_x_discrete(labels = c("Very Low (0-25%)",
                                  "Low (25-50%)",
                                  "Medium (50-75%)",
                                  "High (75-99%)",
                                  "Universal (>99%)"))
```




```
  # Customize the labels on the x-axis for categorical bins
```

To further enhance the visual clarity of the bar plot, consider adjusting the y-axis to reflect the percentage of cases instead of number of observations. This involves modifying the y-axis measure from a simple count (the default setting in `geom_bar()`) to a proportion of the total observations. This can be achieved by altering the `aes()` function to include a calculation that divides each count by the total number of observations. Although there are several ways to accomplish this, utilizing `..count..` provides a straightforward method to express the data as percentages.

```
# Adjust the y-axis to reflect the percentage of cases
ggplot(dat2, aes(x = electricity_pop_ord)) +
  geom_bar(aes(y =(100*(..count..)/sum(..count..))), fill = "dodgerblue4") +
  # Adjust y-values to represent the percentage of total observations
  # Add color to the bars
  labs(title = "Global Access to Electricity",
       subtitle = "Data Source: World Development Indicators",
       y = "Number of Observations",
       x = ("Access to Electricity")) +
    # Add titles and labels to the plot
      scale_x_discrete(labels = c("Very Low (0-25%)",
                                  "Low (25-50%)",
                                  "Medium (50-75%)",
                                  "High (75-99%)",
                                  "Universal (>99%)"))
```




```
  # Customize the labels on the x-axis for categorical bins
```

Graphing Distributions Across Groups

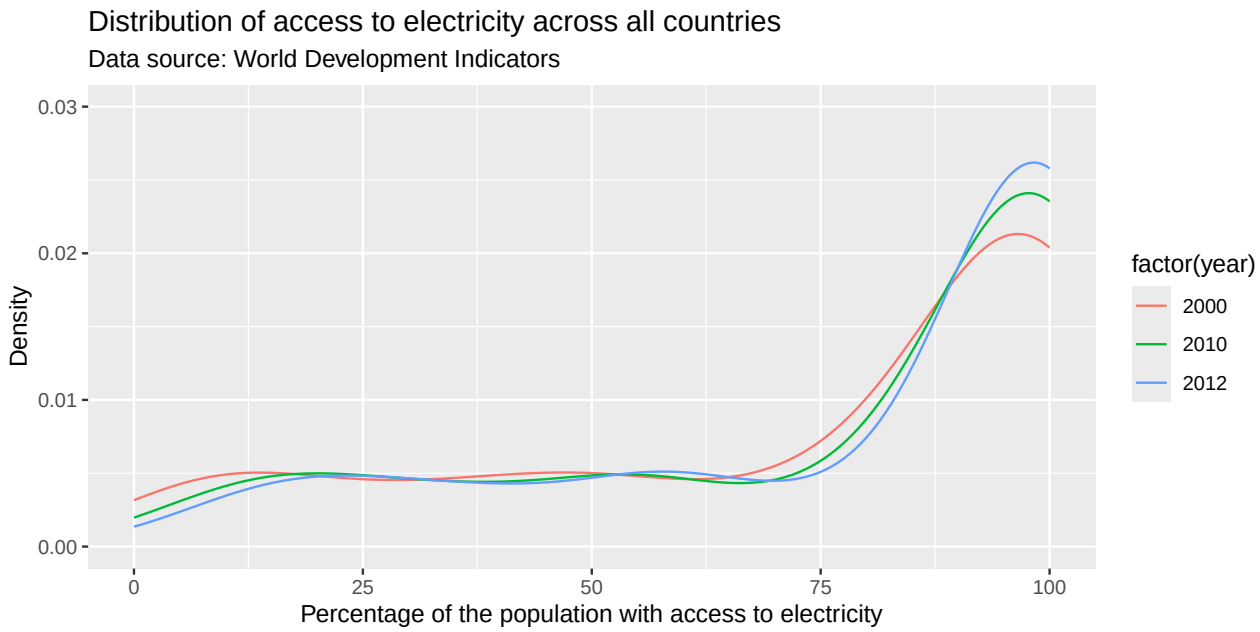
Color Coordination

Sometimes, we want to compare distributions across different groups in our data set. Suppose, we wanted to compare how access to electricity distribution changes over time. Let's examine the three years of observations for our variable `electricity_pop`: 2000, 2010, and 2012.

```
# Display the count of data points per year for electricity_pop
table(dat$year[!is.na(dat$electricity_pop)])
```

We visually distinguish between years using different colors by converting the `year` into a categorical variable and assigning unique colors to each category, specifying them in the `color` parameter within the aesthetics.

```
# Adjust the color 'year'
ggplot(dat, aes(x = electricity_pop, color = factor(year))) +
  geom_line(stat = "density") +
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Percentage of the population with access to electricity",
       y = "Density")  +
  # Add titles and labels to the plot
  coord_cartesian(ylim = c(0, 0.03))
```



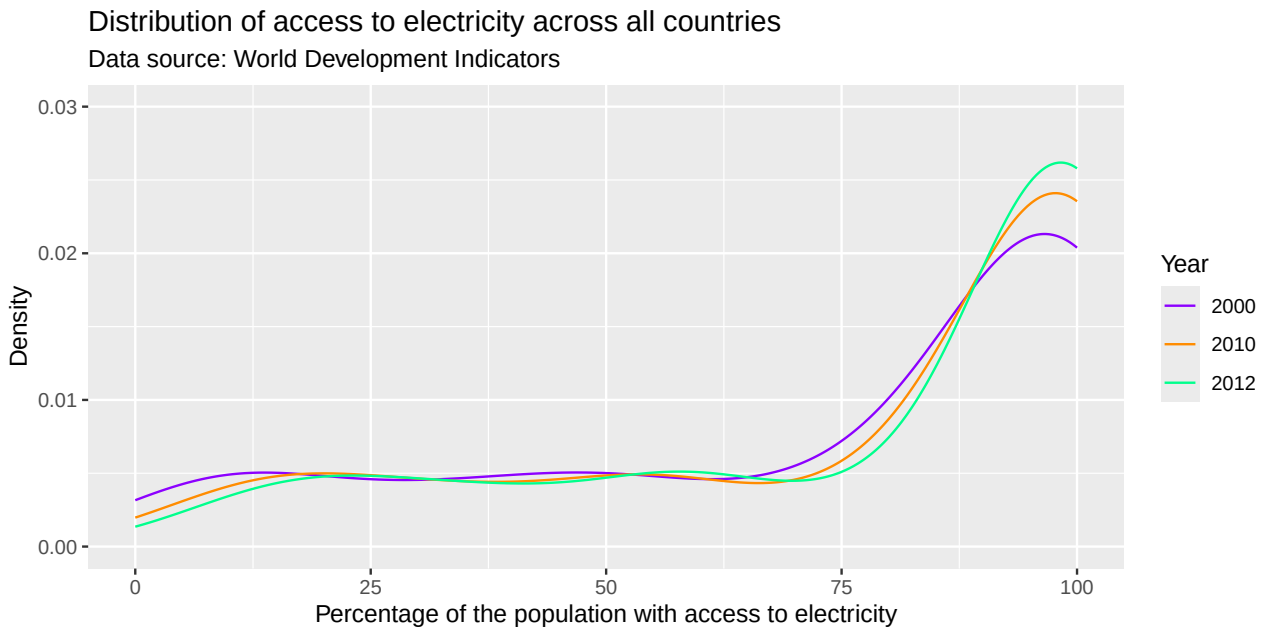

```
  # Adjust the y-axis from 0 to 0.03
```

Question 1: What is the difference between specifying the `color` parameter outside the `aes()` argument versus within the `aes()` argument? (See answer at the bottom of the walkthrough)

Question 2: How would you interpret this plot? How did the access to electricity change over time? (See answer at the bottom of the walkthrough)

We can also customize the colors to distinguish between the three years using the `scale_color_manual()` function. This will change the colors in both the plot and the legend. Let us choose triadic colors to "darkorange" from ColorHexa.

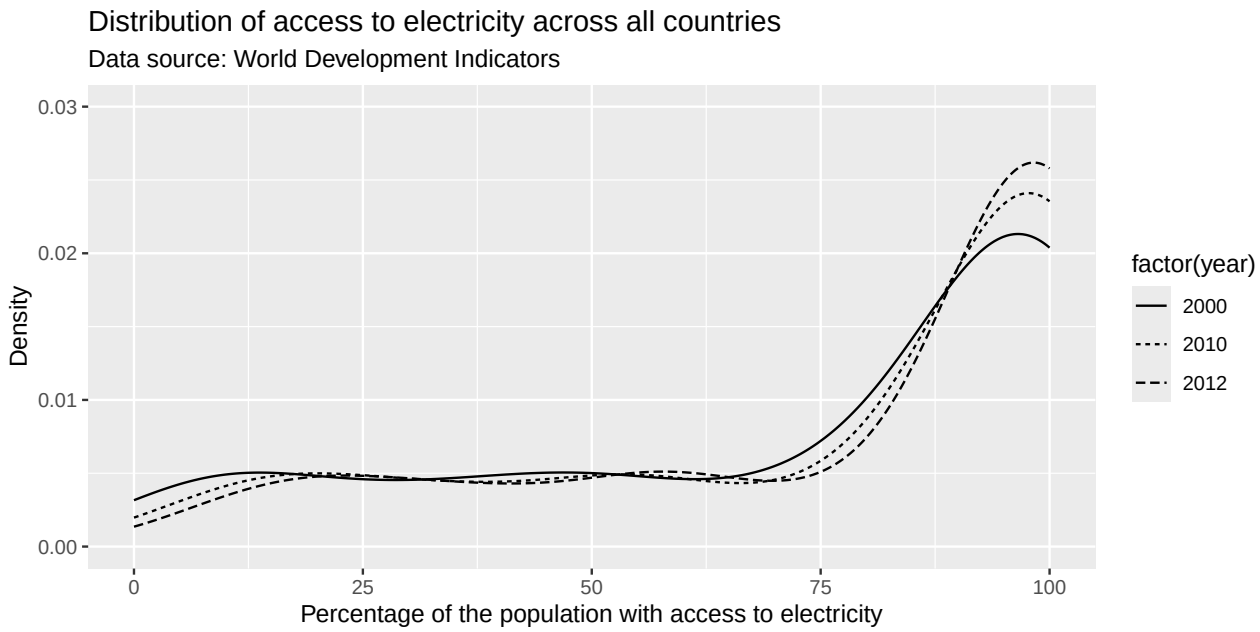
```
# Adjust the color for different years
ggplot(dat, aes(x = electricity_pop, color = factor(year))) +
  geom_line(stat = "density") +
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Percentage of the population with access to electricity",
       y = "Density")  +
  # Add titles and labels to the plot
  coord_cartesian(ylim = c(0, 0.03)) +
  # Adjust the y-axis from 0 to 0.03
  scale_color_manual(values = c("#8c00ff",
                                "#ff8c00",
                                "#00ff8c"),
                     name = "Year")
```




```
  # Customize the colors to distinguish between the three years
```

Lastly, many academic journals will only accept graphs on a gray scale. This means that color will not be enough to differentiate the three lines. We can use different line types instead by specifying the `linetype` parameter within the `aes()` argument.. This also makes the graph more color blind friendly

```
# Adjust the color for different years
ggplot(dat, aes(x = electricity_pop, linetype = factor(year))) +
  geom_line(stat = "density") +
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Percentage of the population with access to electricity",
       y = "Density")  +
  # Add titles and labels to the plot
  coord_cartesian(ylim = c(0, 0.03))
```

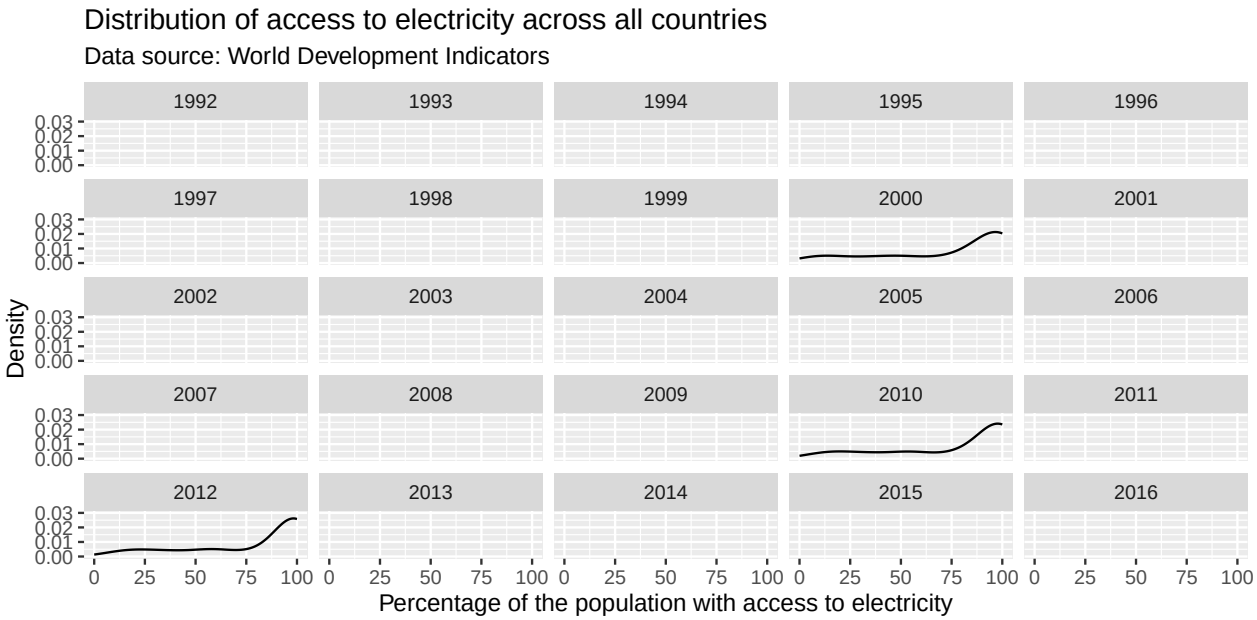



```
  # Adjust the y-axis from 0 to 0.03
```

Interactive Faceting

Another option to graph different groups is to use faceting. Instead of static plots, we can make your comparisons more interactive with `facet_wrap()` to handle multiple groups (years), allowing detailed exploration of each distribution. This means displaying each category of the variable we're analyzing in separate panels within a single plot. We could also use the `facet_grid()` which allows faceting across more than one variable.

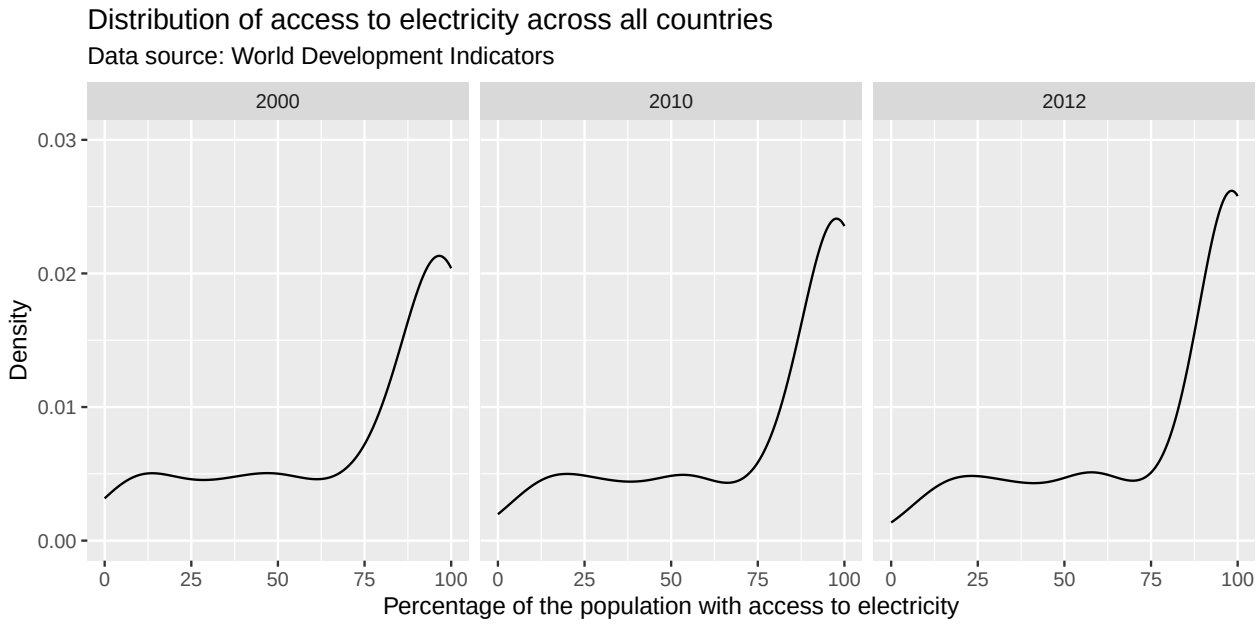
```
# Display multiple years in separate panels within a single plot
ggplot(dat, aes(x = electricity_pop)) +
  geom_line(stat = "density") +
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Percentage of the population with access to electricity",
       y = "Density")  +
  # Add titles and labels to the plot
  coord_cartesian(ylim = c(0, 0.03)) +
  # Adjust the y-axis from 0 to 0.03
  facet_wrap(~ year)
```




```
  # Facet the plot by 'year' to show plots for each year in one graph
```

The `facet_wrap()` function creates individual plots for each year (2000, 2010, and 2012), so we subset these years to avoid empty panels. This can be done by either creating a subsample of the data frame or using the `subset()` function directly within `ggplot()`. For this example, we specifically select the years 2000, 2010, and 2012 by creating a vector.

```
# Display individual panels for 2000, 2010, and 2012 within a single plot
ggplot(subset(dat, year %in% c(2000, 2010, 2012)), aes(x = electricity_pop)) +
  # Create a subset of the data for the years 2000, 2010, and 2012
  geom_line(stat = "density") +
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Percentage of the population with access to electricity",
       y = "Density")  +
  # Add titles and labels to the plot
  coord_cartesian(ylim = c(0, 0.03)) +
  # Adjust the y-axis from 0 to 0.03
  facet_wrap(~ year)
```




```
# Facet the plot by 'year' to show plots for each year in one graph
```

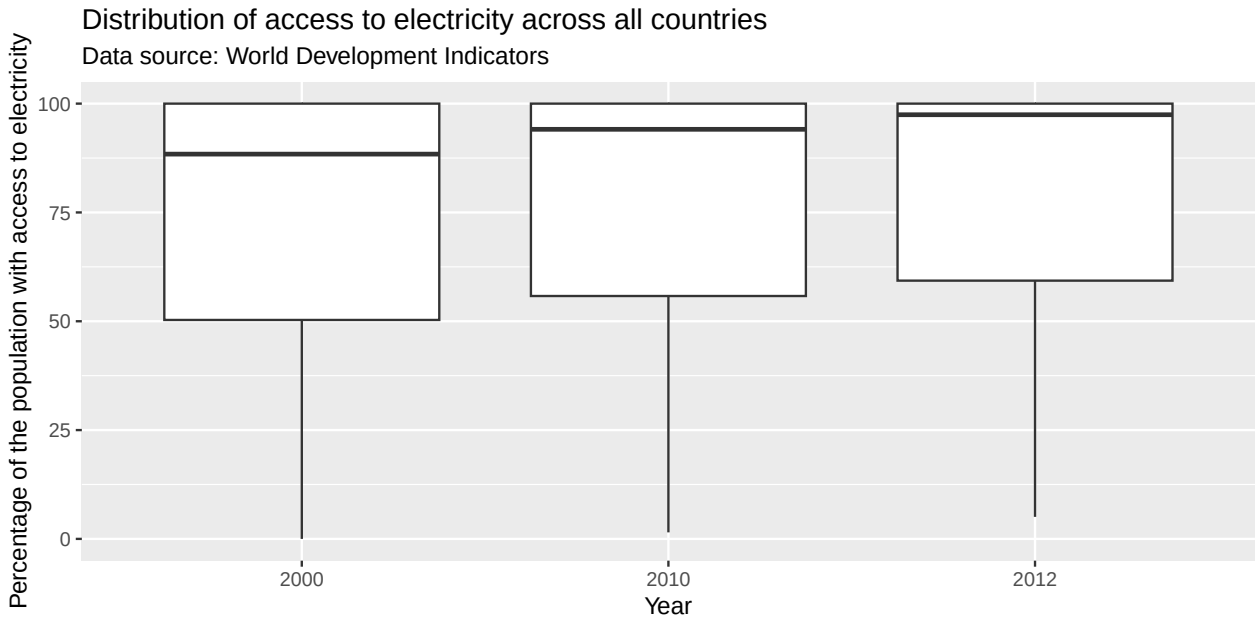
Boxplots Across Time

Another way to show the distribution of variables across groups are boxplots. Boxplots graph different properties of a distribution:

- The box's edges show the 25th and 75th percentiles.
- The median is marked by a line within the box.
- The whiskers extend from the box to represent the range, typically 1.5 times the interquartile range from the quartiles.
- Outliers, if present, are displayed as dots beyond the whiskers.

In `ggplot2` we can graph boxplots across multiple variables using the `geom_boxplot()` geometric object.

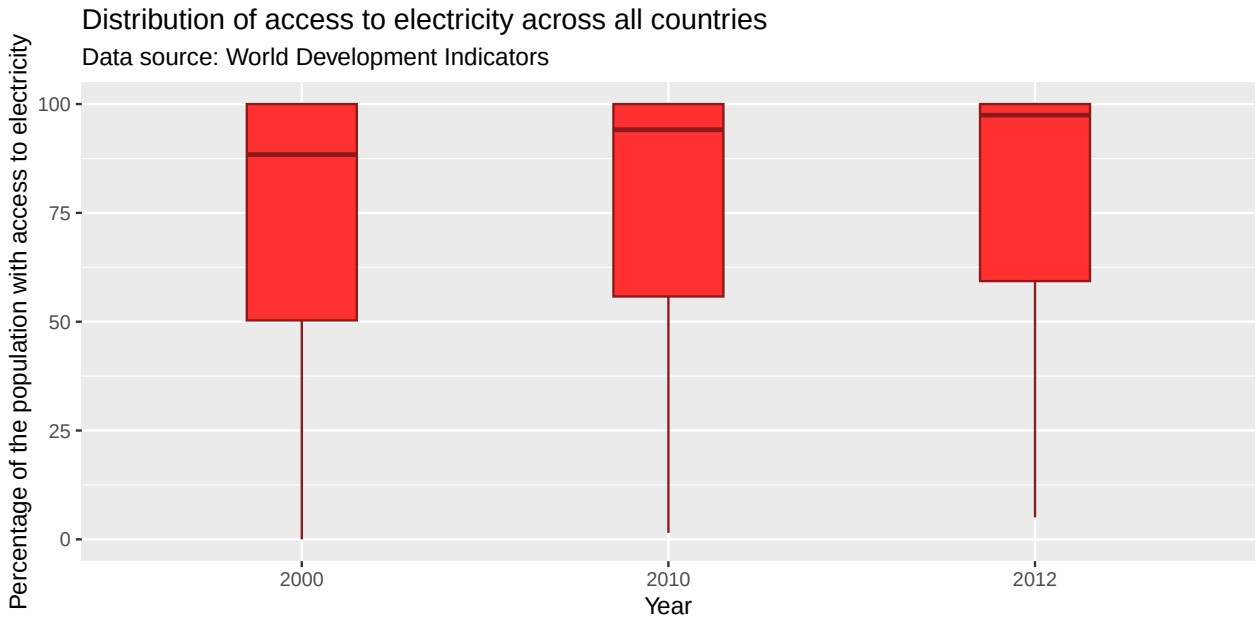
```
# Plot boxplot
ggplot(subset(dat, year %in% c(2000, 2010, 2012)),
       aes(x = factor(year), y = electricity_pop)) +
  geom_boxplot() +
  # Draw boxplots
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Year",
       y = "Percentage of the population with access to electricity")
```




```
  # Add titles and labels to the plot
```

We can customize the appearance of the boxplots by adjusting `color`, `fill`, and `width`.

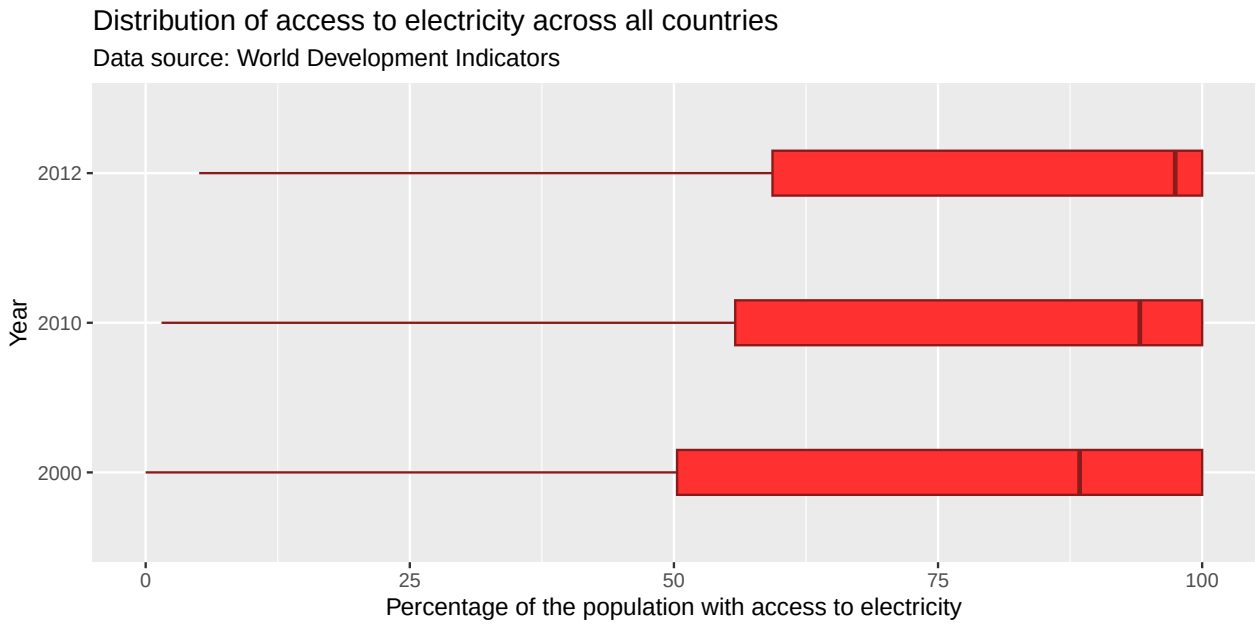
```
# Plot boxplot adjusting color and width
ggplot(subset(dat, year %in% c(2000, 2010, 2012)),
       aes(x = factor(year), y = electricity_pop)) +
  geom_boxplot(width = 0.3, color = "firebrick4", fill = "firebrick1") +
  # Customizing the width, border color, and fill color of boxplots
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Year",
       y = "Percentage of the population with access to electricity")
```




```
  # Add titles and labels to the plot
```

We can change the orientation of the plot with the `coord_flip()` command, which will flip the axes.

```
# Plot horizontal boxplot for 2000, 2010, and 2012 adjusting color and width
ggplot(subset(dat, year %in% c(2000, 2010, 2012)),
       aes(x = factor(year), y = electricity_pop)) +
  geom_boxplot(width = 0.3, color = "firebrick4", fill = "firebrick1") +
  # Customizing the width, border color, and fill color of boxplots
  labs(title = "Distribution of access to electricity across all countries",
       subtitle = "Data source: World Development Indicators",
       x = "Year",
       y = "Percentage of the population with access to electricity") +
  # Add titles and labels to the plot
  coord_flip()
```




```
  # Change the orientation of the plot
```

Saving plots

We can output your plots to many different formats using the `ggsave()` function, including but not limited to `.pdf`, `.jpeg`, `.bmp`, `.tiff`, or `.eps`. In this example, we'll save the plot as `.png` file. We can specify the size of the output graph as well as the resolution in dots per inch (`dpi`). If no specific plot is indicated, `ggsave()` will save the last plot created, which in this case is the horizontally oriented boxplot. By default, if the file path is not fully specified, the plot will be saved to your working directory.

```
# Save last boxplot
ggsave("boxplot_horizontal.png", width = 6, height = 3, dpi = 400)
```

Answers

Question 4: Answer

If the color parameter is specified outside the `aes()` argument, one color is passed all geometric objects of the same type. If the color parameter is specified within the `aes()` argument, different colors are passed to each value of the variable that is passed to the `color` parameter. A separate geometric object will be plotted

for value–each in a different color.

Question 5: Answer

Over time, the amount of country-years with high levels of access to electricity (90% to 100% of the population) has increased. In later years, there are fewer observations at very low levels of access to electricity and there are fewer country-years in which approximately 75% to 90% of the population had access to electricity.

Conclusion

In this walkthrough, we've explored how to create and enhance ggplot2 visualizations with advanced features like custom colors, line types, and interactive elements. You've also learned how to effectively label your plots, control axis limits, and visualize distributions across multiple groups. Mastering these ggplot2 techniques will enable you to create professional, publication-ready visualizations that effectively communicate your findings.

We'll put these concepts into practice through group work and homework assignments.