Data Visualization in R Workshop

2025-04-15

Introduction

Goal: to learn about data visualization using ggplot2 in R

In this workshop we will focus on different types of data visualizations and customizing those visualizations using the ggplot2 package in R. This workshop will use a subset of records from the “diamonds” dataset which is built into ggplot2. This dataset contains information on 53,940 round-cut diamonds, providing details on their physical attributes and pricing.

Loading the Toolkit

1. Open R Studio on the KU virtual lab or use software installed on your laptop.
2. If you have not previously installed the ggplot2 and dpylr packages, remove the # from the `install.packages()` commands below. Then run the code block to install and load the packages using the `library()` function.

```
# install.packages("ggplot2")
# install.packages("dplyr")

library(ggplot2) # for data visualization
library(dplyr) # for data manipulation
```

Loading the Data

To learn more about the diamonds dataset, you can apply the `help()` function. This will open the help panel in R studio and we can see a description of the data and the variables that are included.

To preview the first few lines of the data set, use the `head()` function. You’ll see that the dataset includes 10 variables: price (ranging from 326 dollars to 18,823) dollars, carat weight (from 0.2 to 5.01), and cut quality, categorized into Fair, Good, Very Good, Premium, and Ideal. The color of each diamond is graded from D (best) to J (worst), while clarity is rated from I1 (least clear) to IF (Internally Flawless).

Additionally, the dataset records the physical dimensions of each diamond, including length (x), width (y), and depth (z) in millimeters. The depth percentage and table width provide further geometric characteristics, useful for analyzing a diamond’s proportions and brilliance.

You’ll also notice that the diamonds data set contains over 50,000 records. For speed and clarity of visualizations, we are going to sample 1500 records from the full data set using the code block below.

```
help("diamonds")
head(diamonds)
```

```
## # A tibble: 6 × 10
##   carat cut       color clarity depth table price     x     y     z
##   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
## 2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
## 3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
## 4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
## 5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
## 6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
```

```
# Sampling a small number of observations from the diamonds data set
diamonds <- diamonds %>% dplyr::sample_n(size = 1500, replace = F)
```

Data Visualization in R: ggplot2

We will be using a popular R package for data visualization called **ggplot2**. This package provides a structured, flexible approach to creating graphics such as scatter plots, box plots and bar charts. It follows the “Grammar of Graphics,” a concept that allows users to build plots layer by layer. Unlike base R plotting functions, ggplot2 makes it easy to create and customize visualizations with a few lines of code.

Plotting Basics

A ggplot2 visualization follows a structured approach, where different components of a plot are added layer by layer. The essential components include:

- `ggplot()` - This function sets up the base of a plot by specifying the **data frame** and aesthetic mappings (`aes()`).
- `aes()` - The `aes()` function specifies **which variables** are mapped to different visual properties (x, y, color, size, etc.).
- `geom_*()` - A set of `geom_*()` functions define **what type of plot** to draw. We will work with three commonly used plots: bar plots (`geom_bar`), scatter plots (`geom_point`) and box plots (`geom_box`). If we have time we’ll talk about `geom_histogram()` .

Function	Description
`geom_bar()`	Bar charts
`geom_point()`	Scatter plots
`geom_boxplot()`	Box plots
`geom_histogram()`	Histograms
`geom_line()`	Line charts

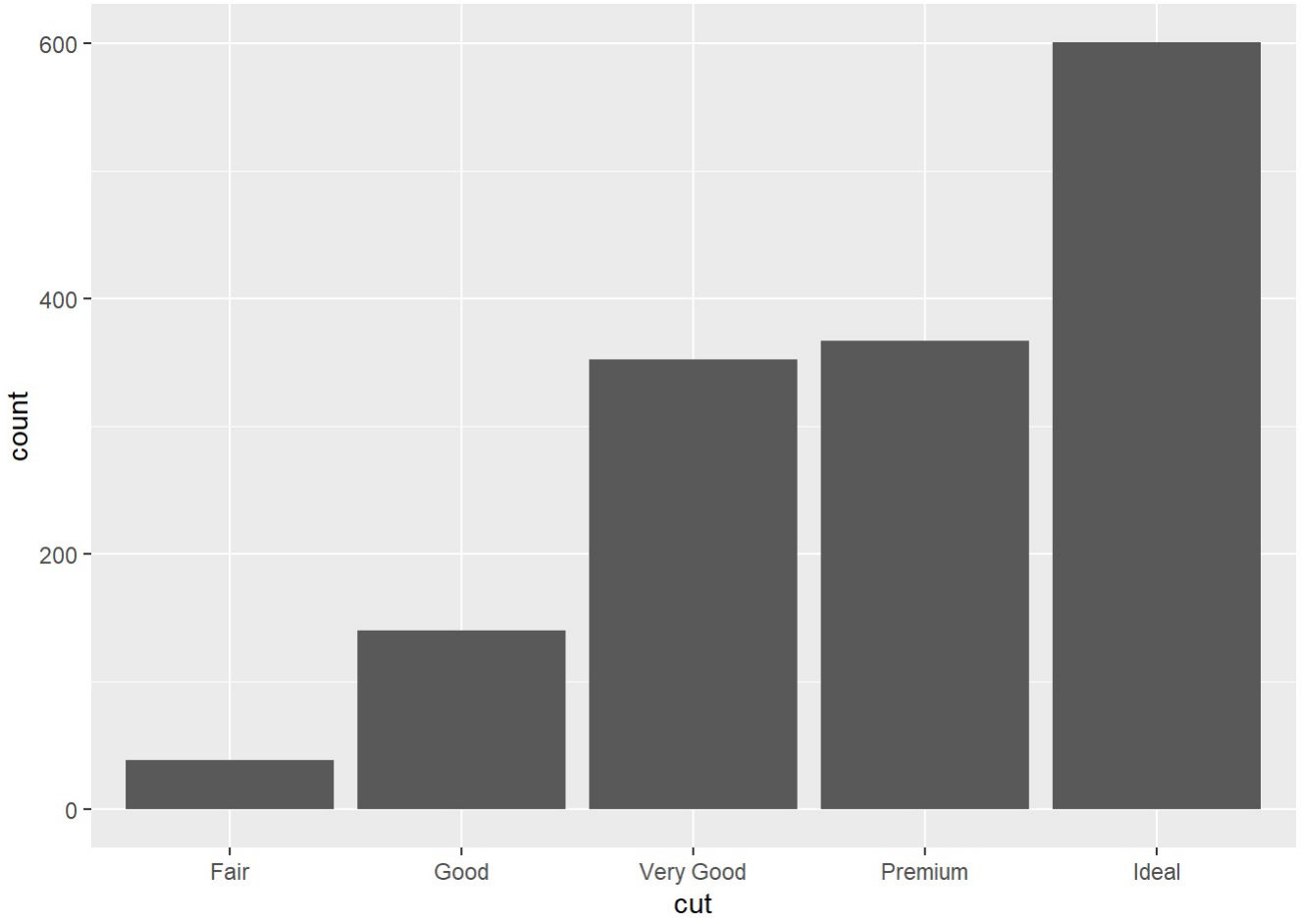
Bar Plots

Creating Basic Bar Charts

Bar Chart of Counts - One Discrete Variable

The first consideration with a bar chart is the statistic we want to display. The command `geom_bar()` is used for bar charts where the y-axis represents a count (i.e., frequency of occurrences of a categorical variable). The example below creates a simple bar chart that counts the number of diamonds in the data set for each **cut category**. A basic bar chart can be created in R using the following command.

```
# Creates a basic bar chart showing the count of each diamond cut category
ggplot(data = diamonds, mapping = aes(x = cut)) +
  geom_bar()
```



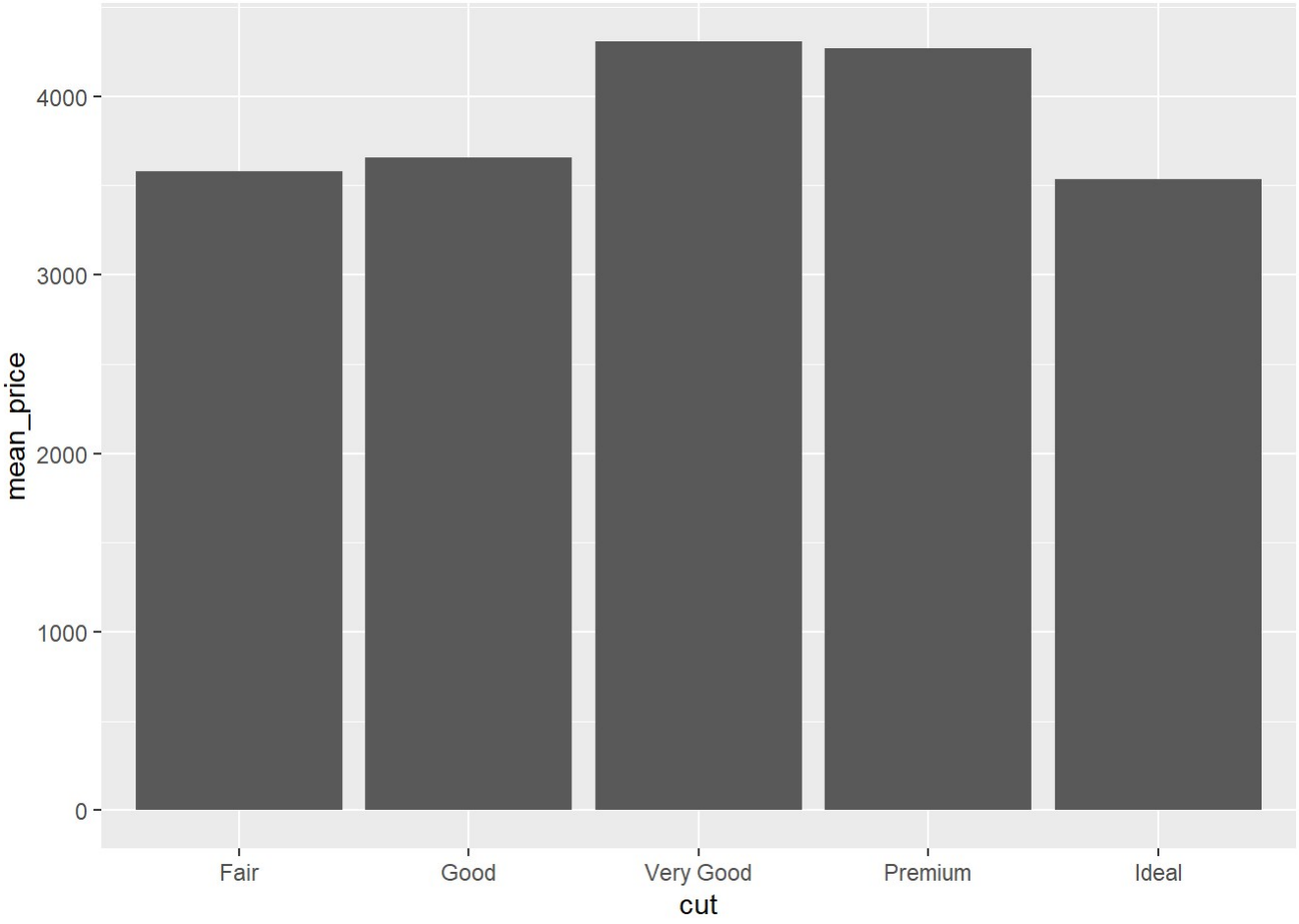

```
# ggplot(data = diamonds, mapping = aes(y = color)) + # flipping the coordinates
#   geom_bar()
```

Bar Chart of Means

Note that basic function `geom_bar` expects only one aesthetic, either an x or y and calculates counts. If you want to plot a mean, for example the **mean price** for each cut, you need to summarize the data first and then use either geom_col() or `geom_bar(stat = "identity")` .

```r
# Summarize data: Compute mean price per cut
diamonds_summary <- diamonds %>%
  group_by(cut) %>%
  summarize(mean_price = mean(price, na.rm = TRUE))

# Create the bar chart with summarized values
ggplot(data = diamonds_summary, aes(x = cut, y = mean_price)) +
  geom_bar(stat = "identity") # ggplot uses the values directly.
```

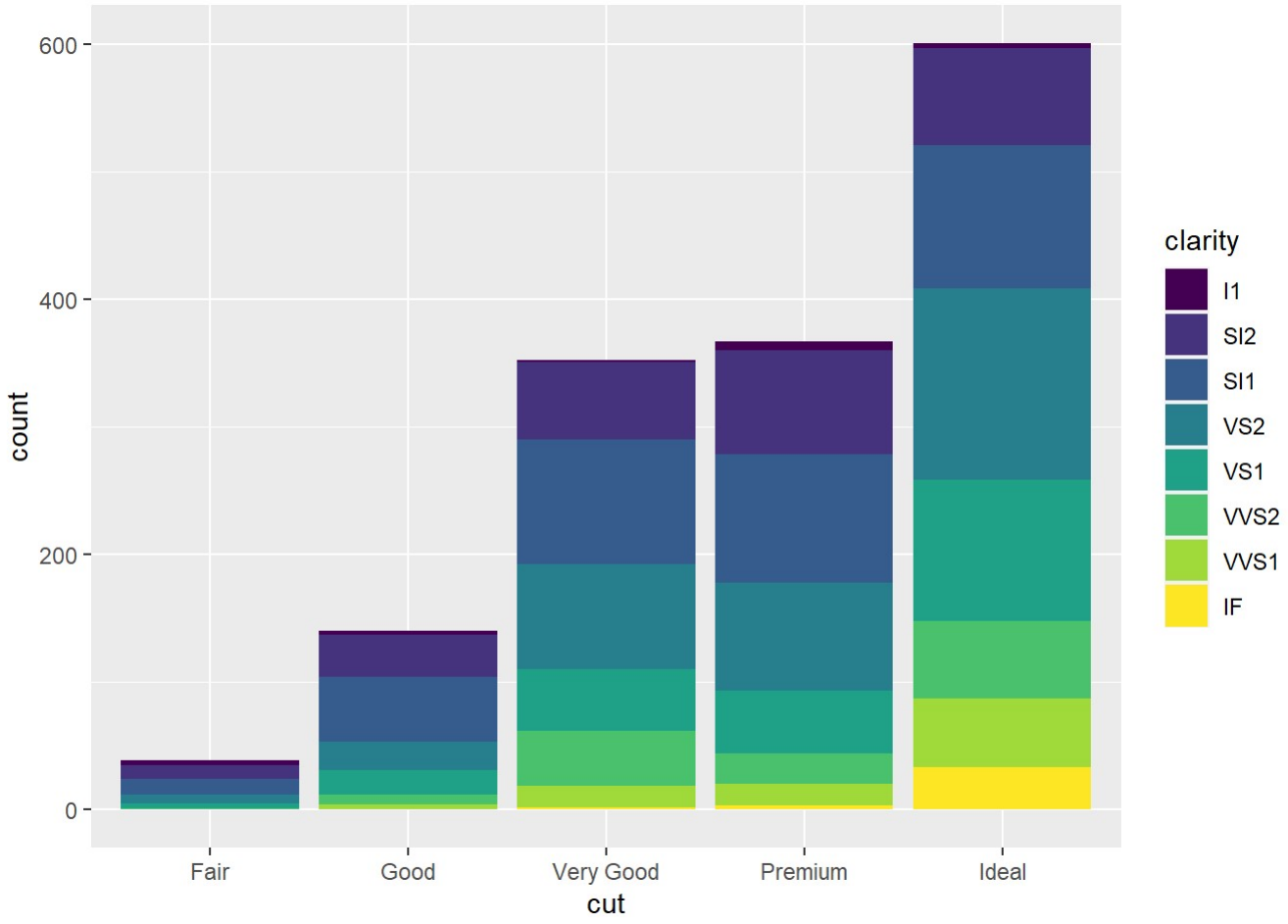



```r
# Alternatively - geom_col uses stat = "identity" automatically - assumes you've summarized t
he data
# ggplot(data = diamonds_summary, aes(x = cut, y = mean_price)) +
#   geom_col()
```

Charting Multiple Variables with Color

Suppose you want to visualize more than one variable at a time. You can chart additional variables using color by adding the `fill =` argument to a basic bar plot. Here, the `fill="clarity"` argument **colors the bars** based on the clarity of the diamonds, creating a stacked bar chart that shows how clarity is distributed within each category of cut.

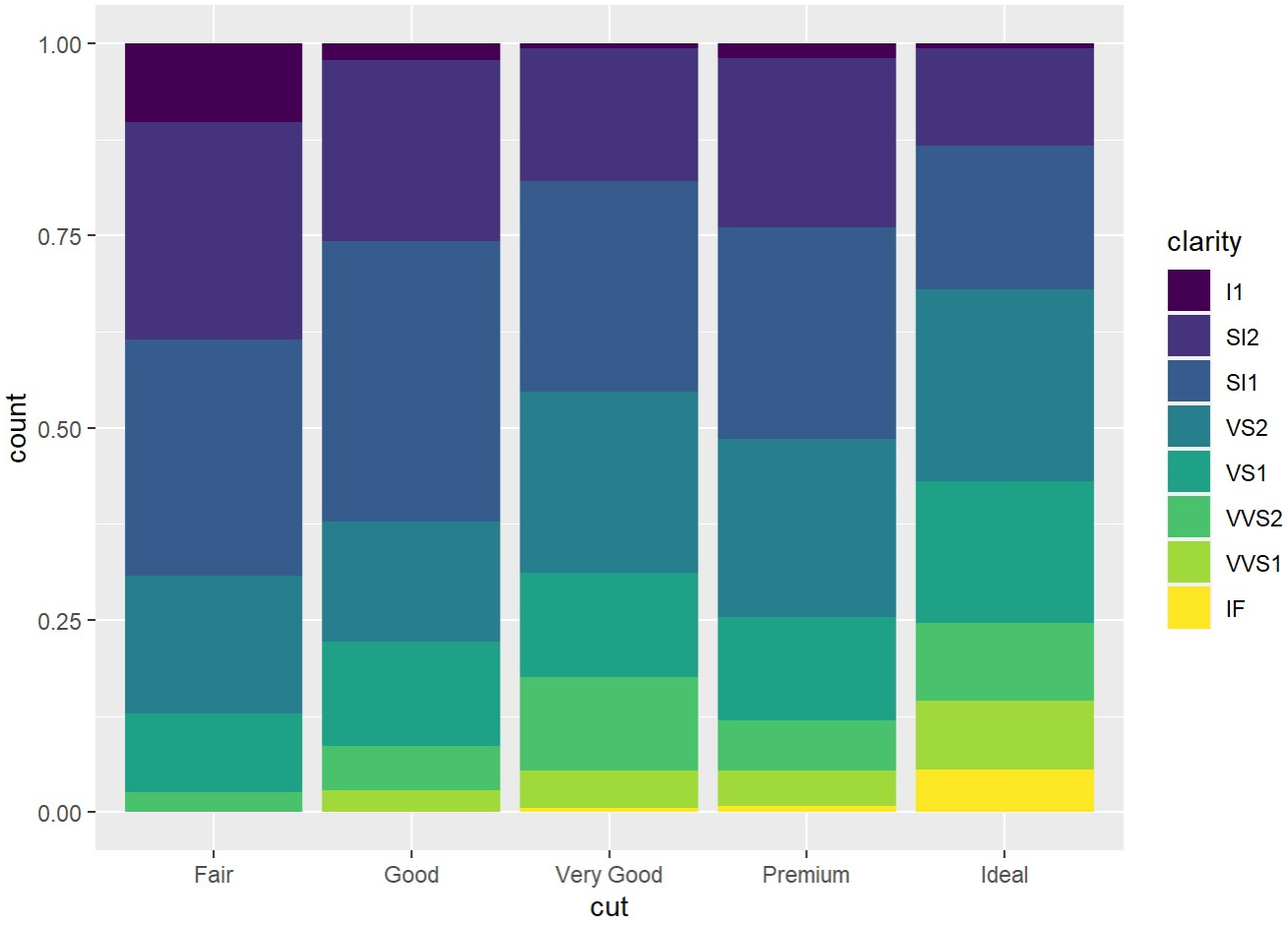
```
# Creates a bar chart where fill colors indicate clarity categories
# added fill argument to aes()
ggplot(data = diamonds, mapping = aes(x = cut, fill = clarity)) +
  geom_bar()
```



Bar Positioning: Stacked Bar Charts

You can modify the bar positioning by using the `position = "fill"` argument, which normalizes the bars to represent **proportions instead of counts**. This makes it easier to compare the relative distribution of clarity across different cuts.

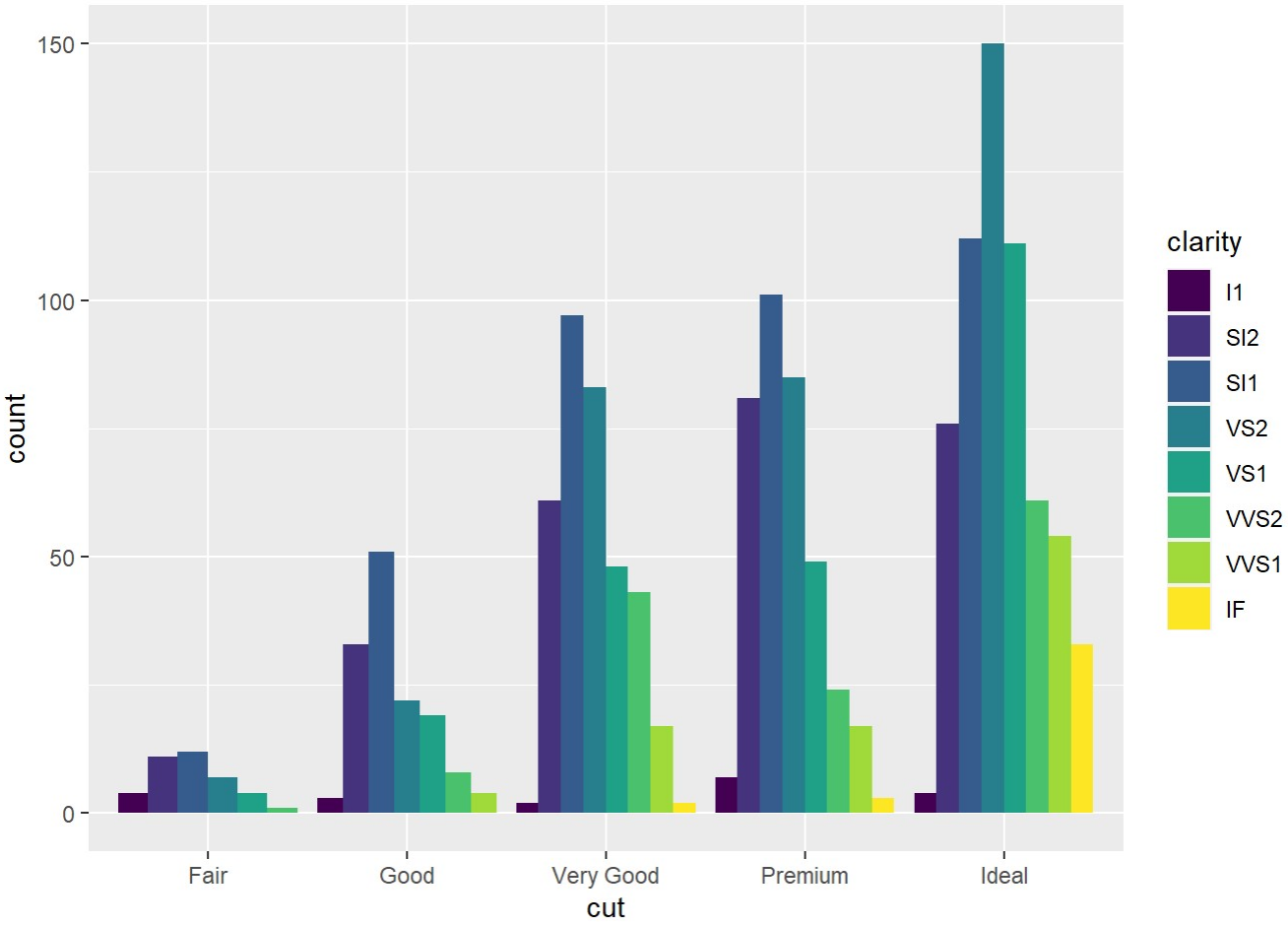
```
# Creates a proportional (100%) stacked bar chart
ggplot(data = diamonds, mapping = aes(x = cut, fill = clarity)) +
  geom_bar(position = "fill")  # specify position argument to geom_bar()
```



Bar Positioning: Grouped Bar Charts

If we modify the the plot code slightly by specifying `position = "dodge"`, instead of `position = "fill"`, ggplot2 places bars for each clarity level **side-by-side** instead of stacking them, allowing for direct comparisons between clarity levels within each cut category.

```
# Creates a grouped bar chart
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity),
           position = "dodge")  # specify dodge argument to geom_bar()
```

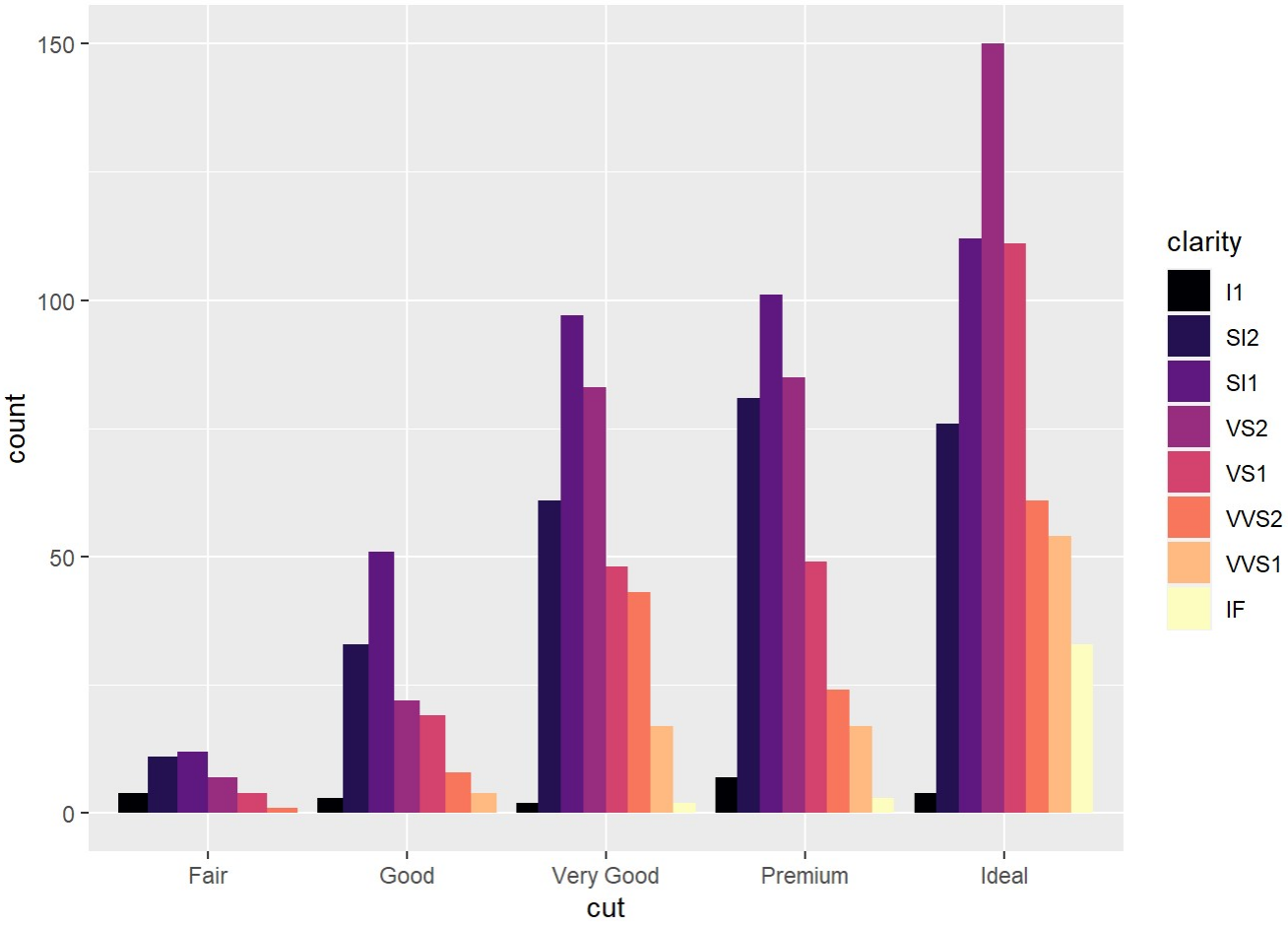


Customizing Colors with `scale_*`

Using `scale_fill_viridis_d()`

The `scale_*()` functions allow you to adjust axes, colors, and legends. Scales modify the appearance of the plot, including axis labels and colors. Here's we're looking at two ways to modify color. The first example below creates a grouped bar chart that visualizes the distribution of diamond cuts while distinguishing clarity levels using custom colors. The `scale_fill_viridis_d()` function along with `option = "magma"` works to **apply a colorblind-friendly palette** to the plot.

```
# Changing the chart color scheme using scale_fill_viridis_d
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity),
           position = "dodge") +
  scale_fill_viridis_d(option = "magma")
```

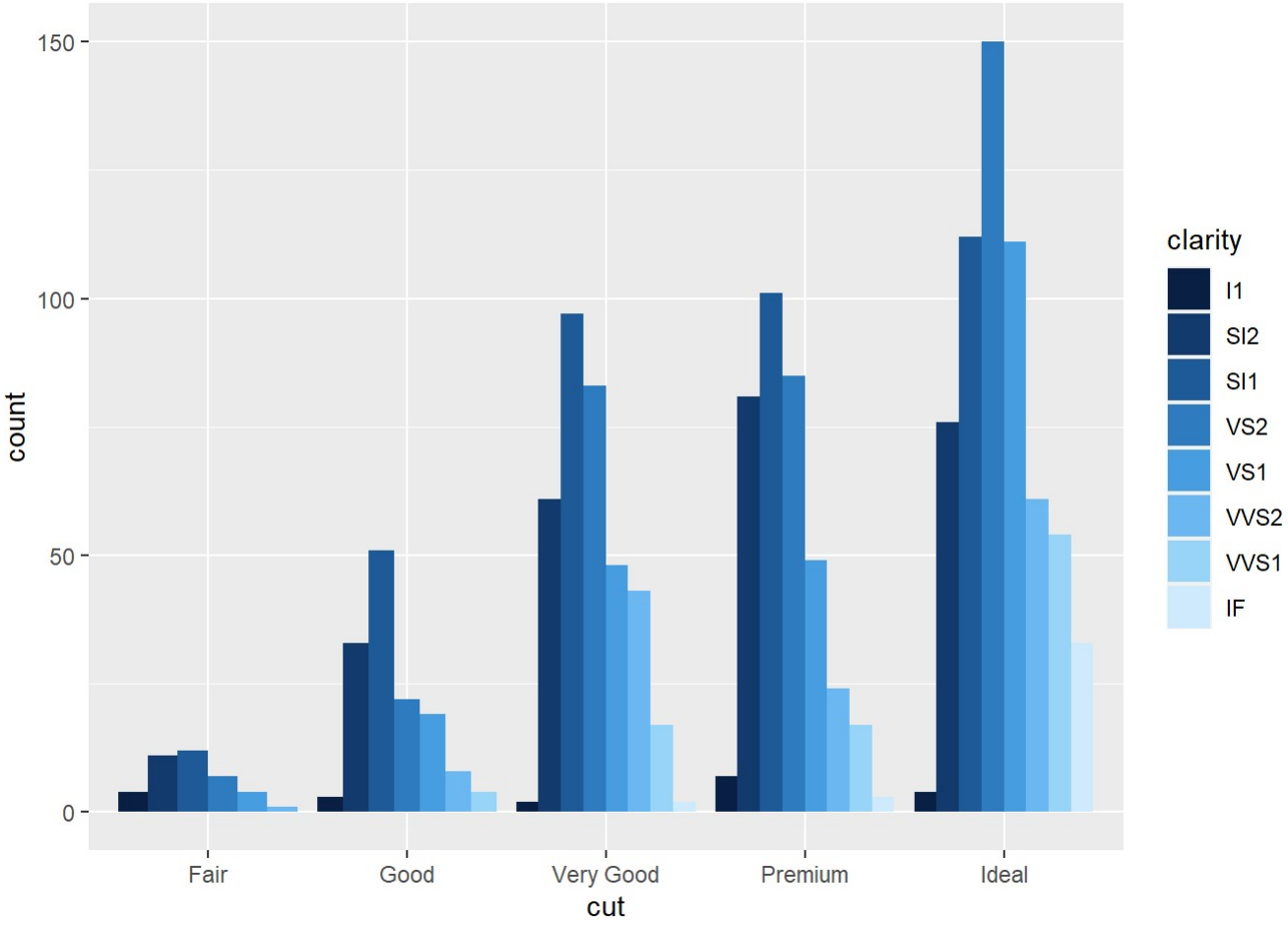


Using `scale_fill_manual()`

The default colors assigned to each clarity level by ggplot2 may not be ideal, especially if you want consistent branding or more control over how the chart looks. That's where `scale_fill_manual()` comes in. This function allows you to **assign custom colors** to each clarity level using specific hex codes. It overrides ggplot2's default color scale and lets you assign your own specific colors to each value of the fill aesthetic.

Here each name like "I1" or "SI2" matches a level of the clarity variable. Each hex code like "#0A1F44" is the color that bar segment will be filled with. These assignments tell ggplot2: "When you see clarity = 'SI2', use this exact shade of blue."

```
# Changing the chart color scheme using scale_fill_manual
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity),
           position = "dodge") +
  scale_fill_manual(
    values = c(
      "I1" = "#0A1F44",
      "SI2" = "#123A6D",
      "SI1" = "#1E5A97",
      "VS2" = "#2E7BC0",
      "VS1" = "#469DE0",
      "VVS2" = "#6BB7F1",
      "VVS1" = "#98D4F7",
      "IF" = "#CFEAFD"
    ))
```



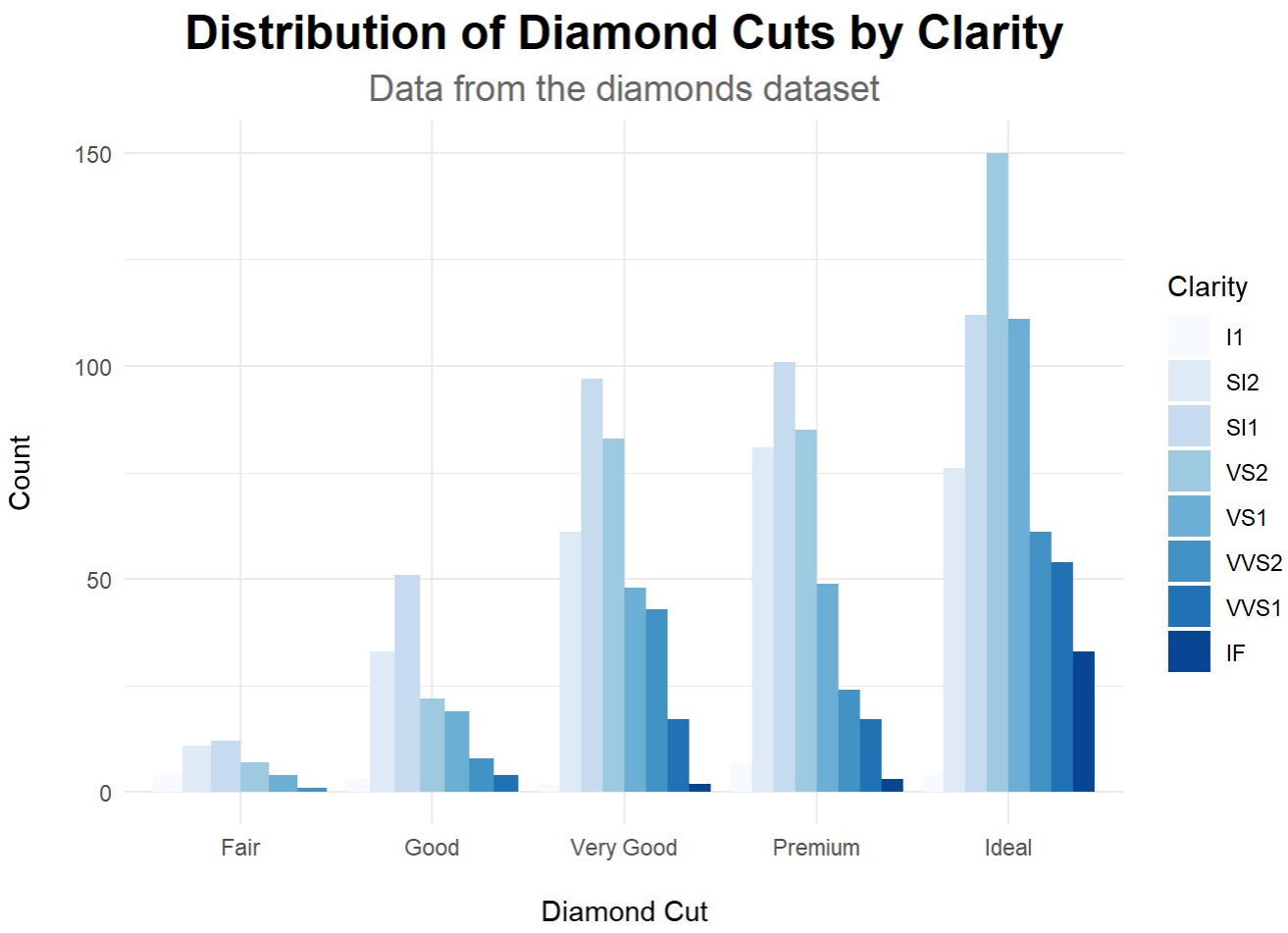

```
# Easier Shortcut to a color gradient chart
#ggplot(data = diamonds) +
#  geom_bar(mapping = aes(x = cut, fill = clarity),
#           position = "dodge") +
#  scale_fill_brewer(palette = "Purples")
```

Customizing Labels with `labs()` and `theme()`

While simple bar charts are useful for data exploration, sometimes we want to make a nicely formatted chart for inclusion in an assignment, report or publication. In the example below, we're building on what we've done previously. We're just enhancing the plot by customizing labels and styling elements for better readability and aesthetics. Here:

- The `labs()` function is used to add a title, subtitle, axis labels, and a legend title, making the plot more informative.
- The `theme_minimal()` function gives the plot a clean, modern appearance by removing unnecessary grid lines.
- The `theme()` function allows us to fine-tune text styling, including font size, typeface, justification, and color. We can also use the `theme()` function to specify the font size, type face, and horizontal justification and color of the labels.

```
ggplot(data = diamonds, mapping = aes(x = cut, fill = clarity)) +
  geom_bar(position = "dodge") +
  scale_fill_brewer(palette = "Blues") +
  labs(
    title = "Distribution of Diamond Cuts by Clarity",
    subtitle = "Data from the diamonds dataset",
    x = "Diamond Cut",
    y = "Count",
    fill = "Clarity"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(face = "bold", size = 18, hjust = 0.5),
    plot.subtitle = element_text(size = 14, hjust = 0.5, color = "gray40"),
    axis.text.x = element_text(angle = 0, hjust = 0.5),
    axis.title.x = element_text(margin = margin(t = 15)),  # Moves x-axis label downward
    axis.title.y = element_text(margin = margin(r = 15)),  # Moves y-axis label to the left
  )
```



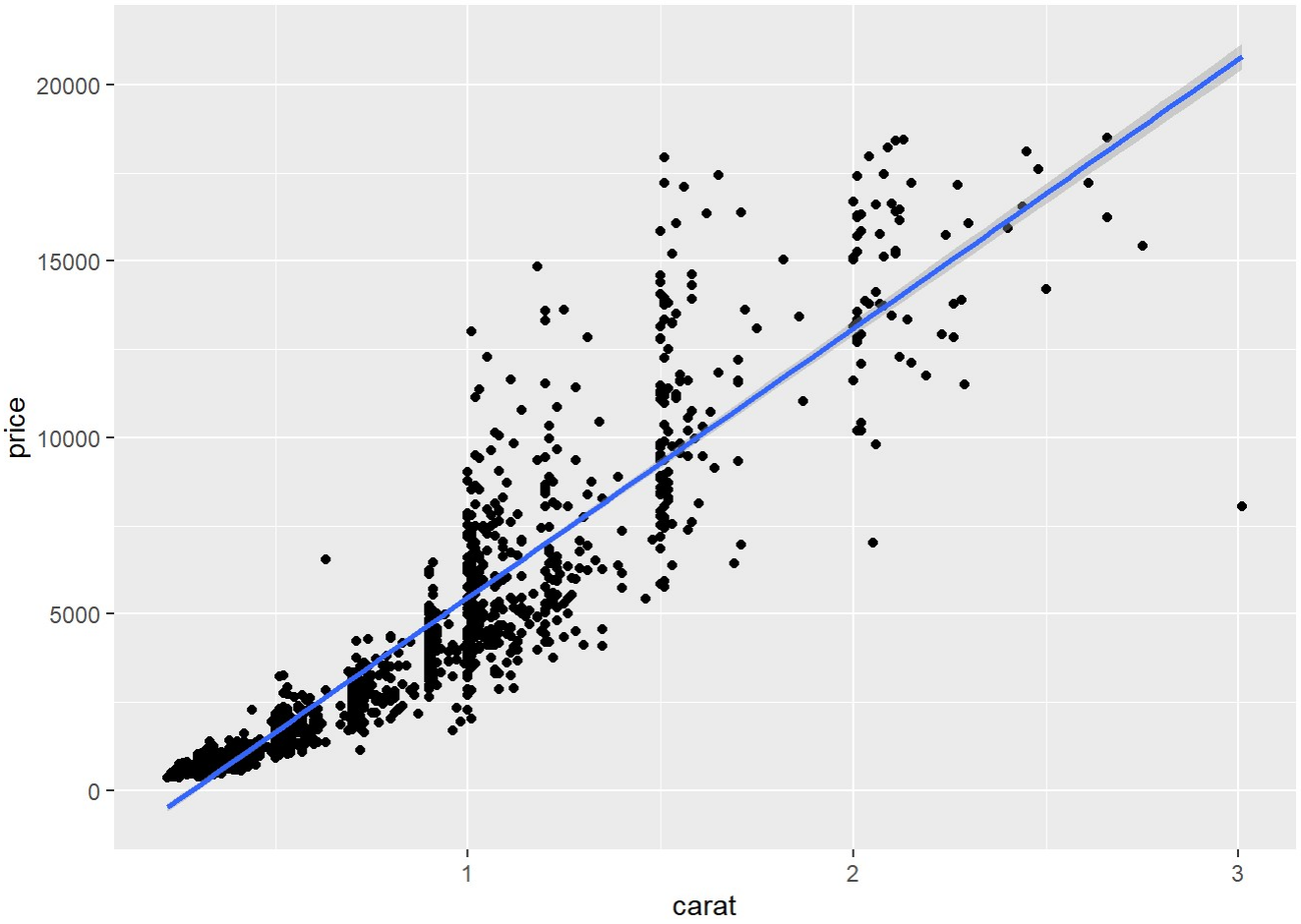
Scatter Plots

Now that you know how to create a customize a bar chart using ggplot2, we'll move on to some other types of charts. First, we'll go over the basic plotting function for scatter plots and then we'll illustrate how to customize the plots using scales and themes.

Creating Basic Scatter Plots

A scatter plot is useful for showing relationships between two numerical variables. In the diamonds data set, a great way to start is by plotting carat (weight of the diamond) vs. price to see how weight affects cost. Once again, `ggplot(data = diamonds, aes(x = carat, y = price))` initializes the plot, specifying that we are using the diamonds data set. The `aes(x = carat, y = price)` sets up the aesthetics mapping: X-axis = carat (the weight of the diamond) and Y-axis = price (the cost of the diamond in US dollars). Then, `geom_point()` adds points to the plot, creating a scatter plot where each dot represents one diamond in the data set.

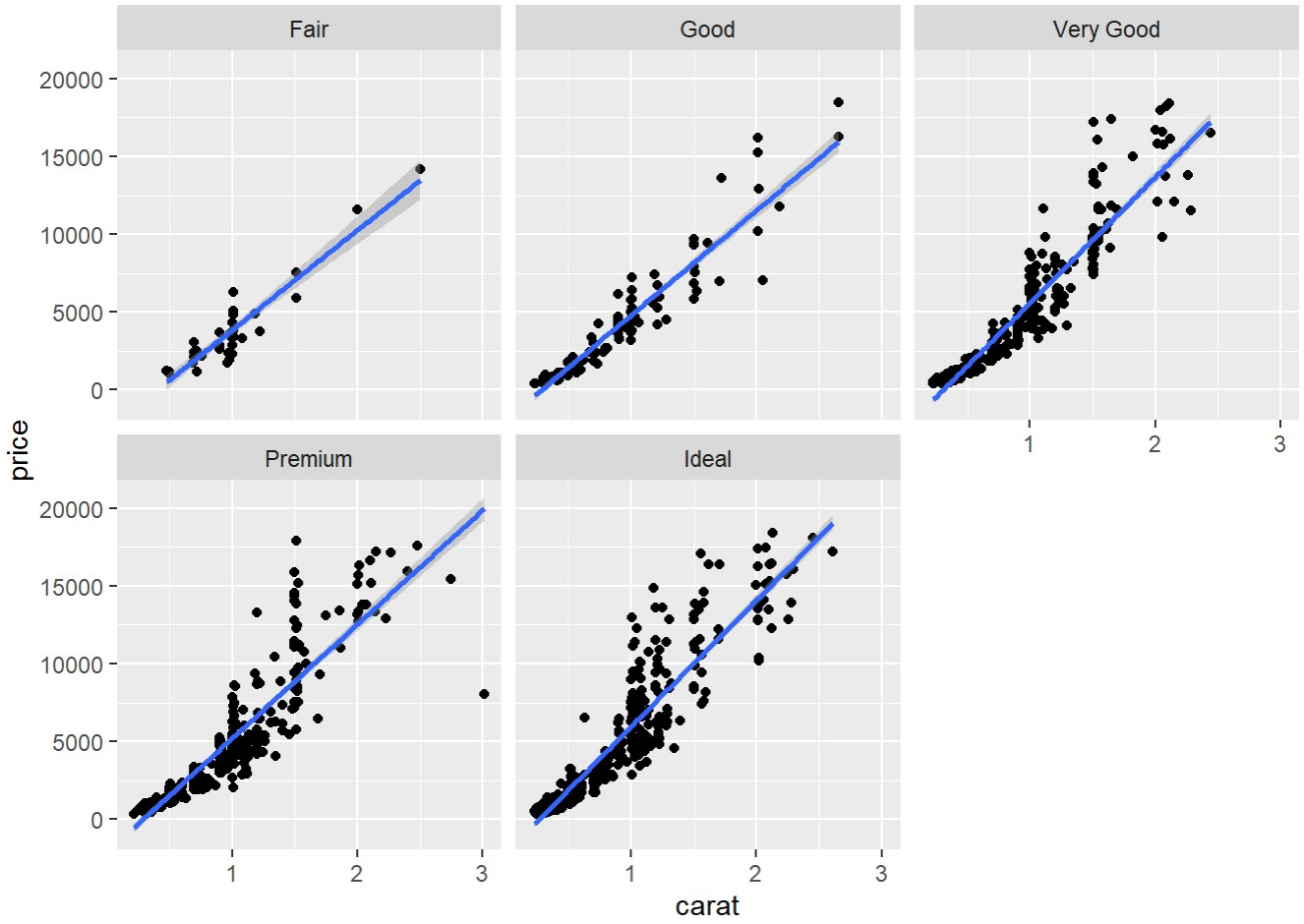
```
ggplot(data = diamonds, aes(x = carat, y = price)) +  # Sets up aesthetic mapping
  geom_point() + # adds points to the plot
  geom_smooth(method = "lm")    # Adds a linear trendline
```



Faceting Using facet_wrap()

Here, the second plot introduces `facet_wrap(~cut)`, which creates a series of smaller scatter plots, one for each level of the cut variable. Instead of overlaying everything on one chart, `facet_wrap()` splits the data into panels—each showing a subset of the data for a specific diamond cut. This is useful when you want to compare how the relationship between carat and price might differ by levels of a categorical variable.

```
ggplot(data = diamonds, aes(x = carat, y = price)) +
  geom_point() +
  geom_smooth(method = "lm") +
  facet_wrap(~cut) # makes one scatterplot for each level of a categorical variable
```

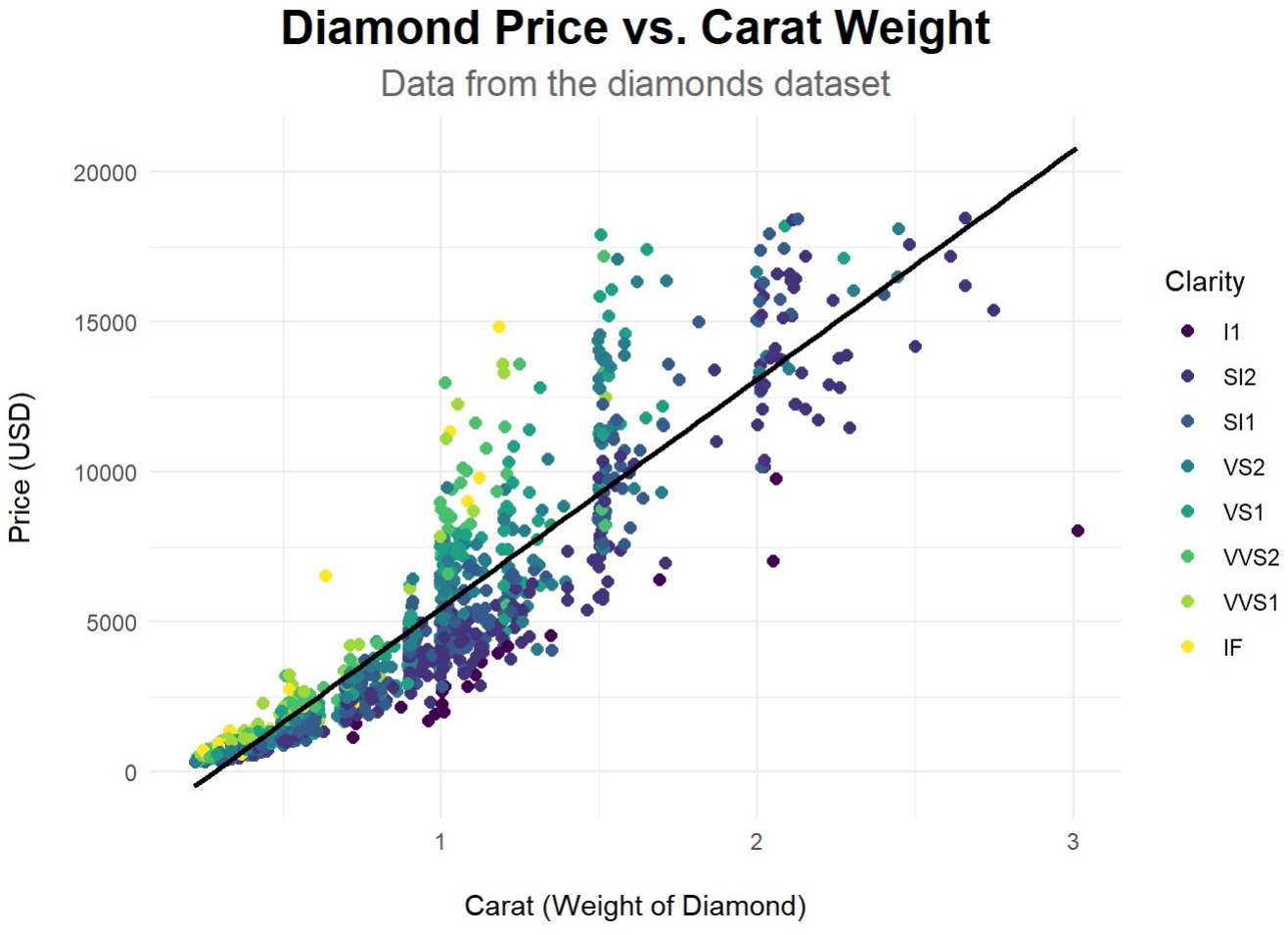


Customizing Scatter Plots

Just like bar charts, it is possible to fully customize a scatter plot in ggplot2 using `scale_*`, `labs()` and `theme()` and arguments within `geom_point()`.

The code below produces a scatter plot that visualizes the relationship between diamond carat (weight) and price, with clarity represented by distinct colors. The points are sized at 2 using `geom_point(size = 2),` making them more visible. A custom color palette is applied via `scale_color_manual()`. The `labs()` function enhances readability by adding a title, subtitle, axis labels, and a legend title. The `theme_minimal()` function is used to create a clean, modern design. Additional customizations are applied through `theme()`, where the title and subtitle are bolded, resized, and centered, and the axis labels are adjusted with extra margin space to prevent crowding.

```
ggplot(data = diamonds, aes(x = carat, y = price, color = clarity)) +
  geom_point(size = 2, position = "jitter") +
  geom_smooth(aes(group = 1), method = "lm", color = "black", se = FALSE) +
  scale_fill_viridis_d() + # Using a preset color palette
  labs(
    title = "Diamond Price vs. Carat Weight",
    subtitle = "Data from the diamonds dataset",
    x = "Carat (Weight of Diamond)",
    y = "Price (USD)",
    color = "Clarity"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(face = "bold", size = 18, hjust = 0.5),
    plot.subtitle = element_text(size = 14, hjust = 0.5, color = "gray40"),
    axis.text.x = element_text(angle = 0, hjust = 0.5),
    axis.title.x = element_text(margin = margin(t = 15)),  # Moves x-axis label downward
    axis.title.y = element_text(margin = margin(r = 15)),  # Moves y-axis label to the left
  )
```

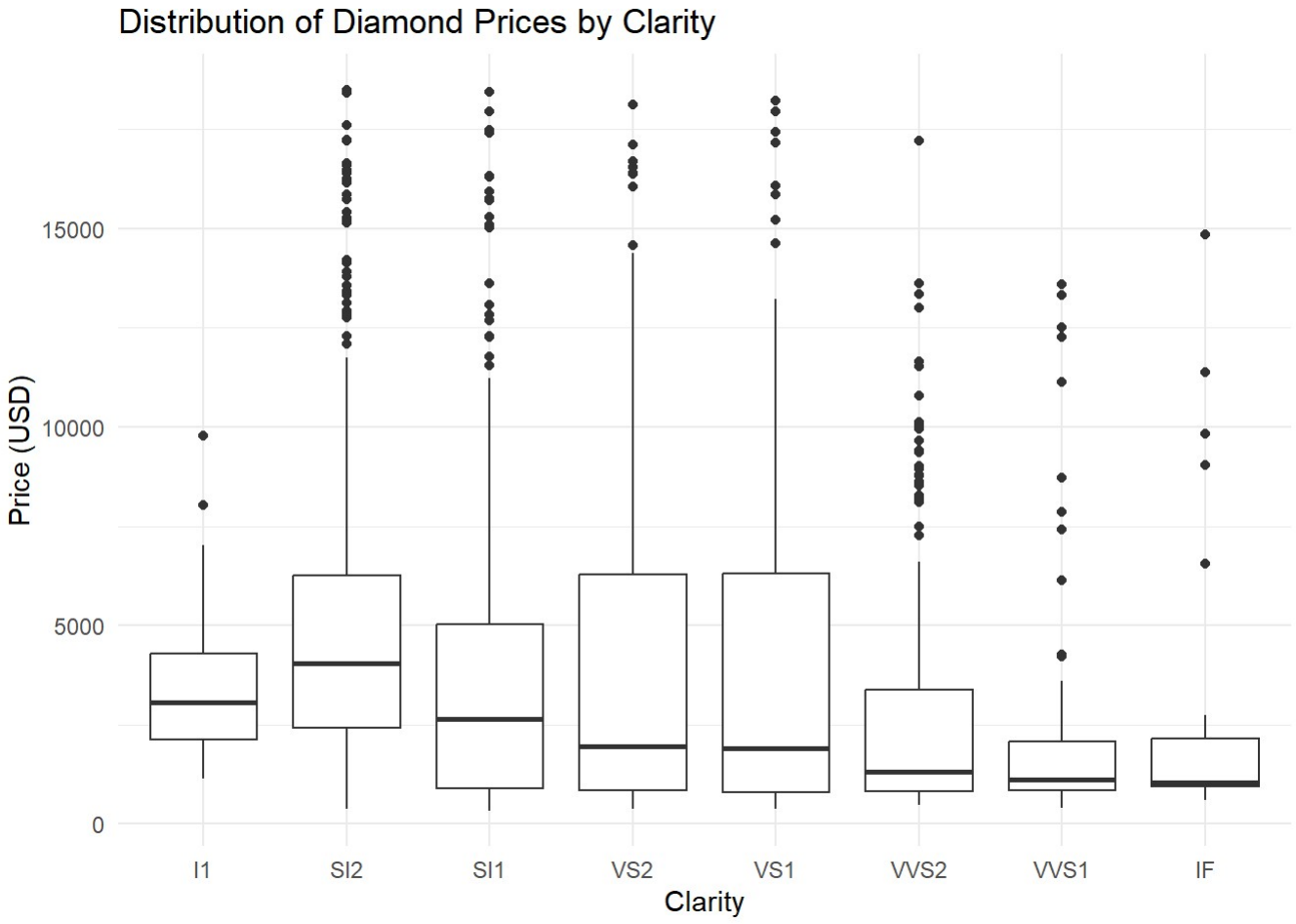


Box Plots

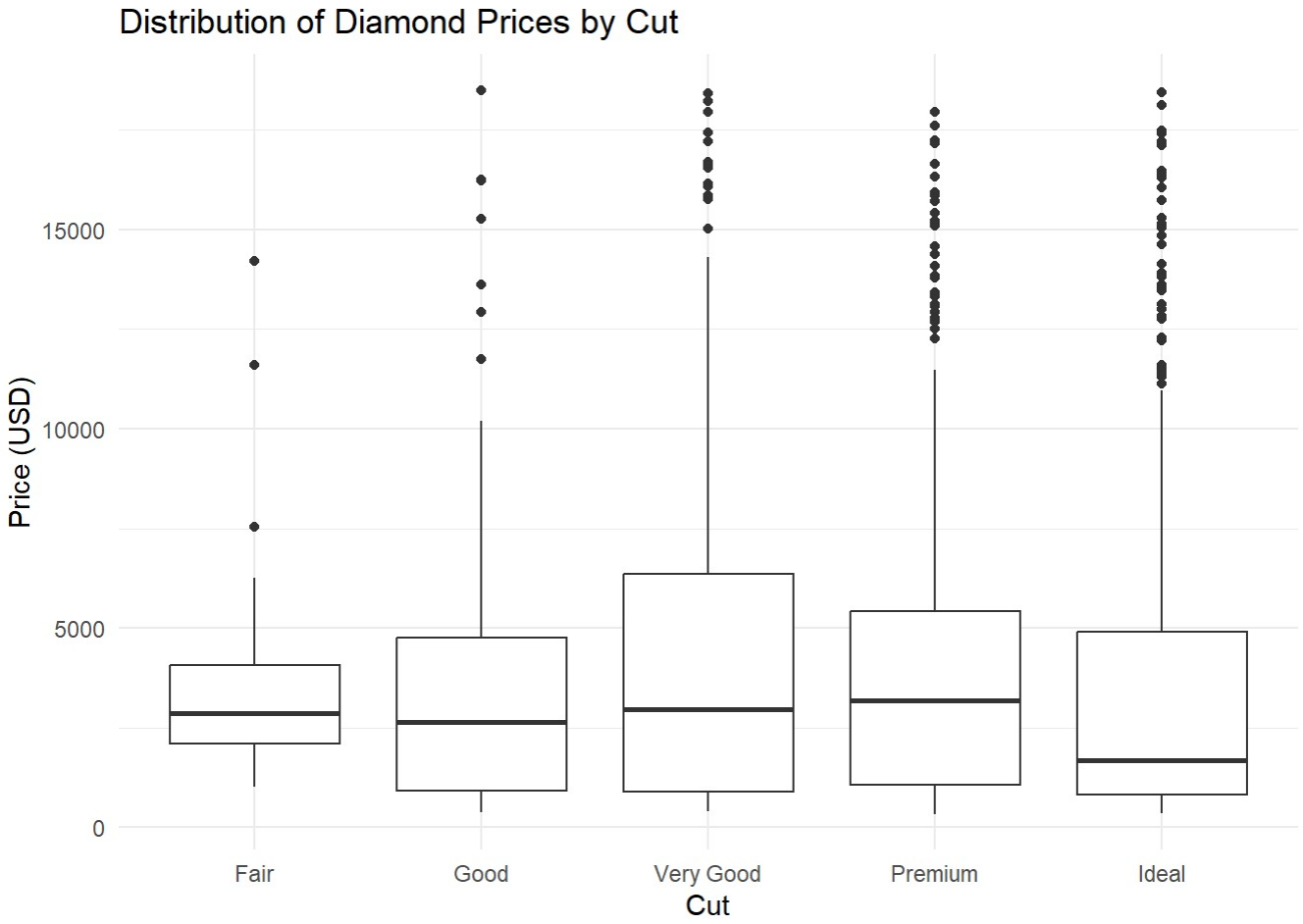
Creating Basic Box Plots

We can follow the same basic patterns to create basic box plots using `geom_boxplot()`. A box plot is useful for visualizing the ***distribution of a numerical variable*** across different categories. In the diamonds data set, you can use ggplot2 to create boxplots of price across the different categories of cut or clarity.

```
ggplot(data = diamonds, aes(x = clarity, y = price)) +
  geom_boxplot() +
  labs(
    title = "Distribution of Diamond Prices by Clarity",
    x = "Clarity",
    y = "Price (USD)"
  ) +
  theme_minimal()
```




```
ggplot(data = diamonds, aes(x = cut, y = price)) +
  geom_boxplot() +
  labs(
    title = "Distribution of Diamond Prices by Cut",
    x = "Cut",
    y = "Price (USD)"
  ) +
  theme_minimal()
```

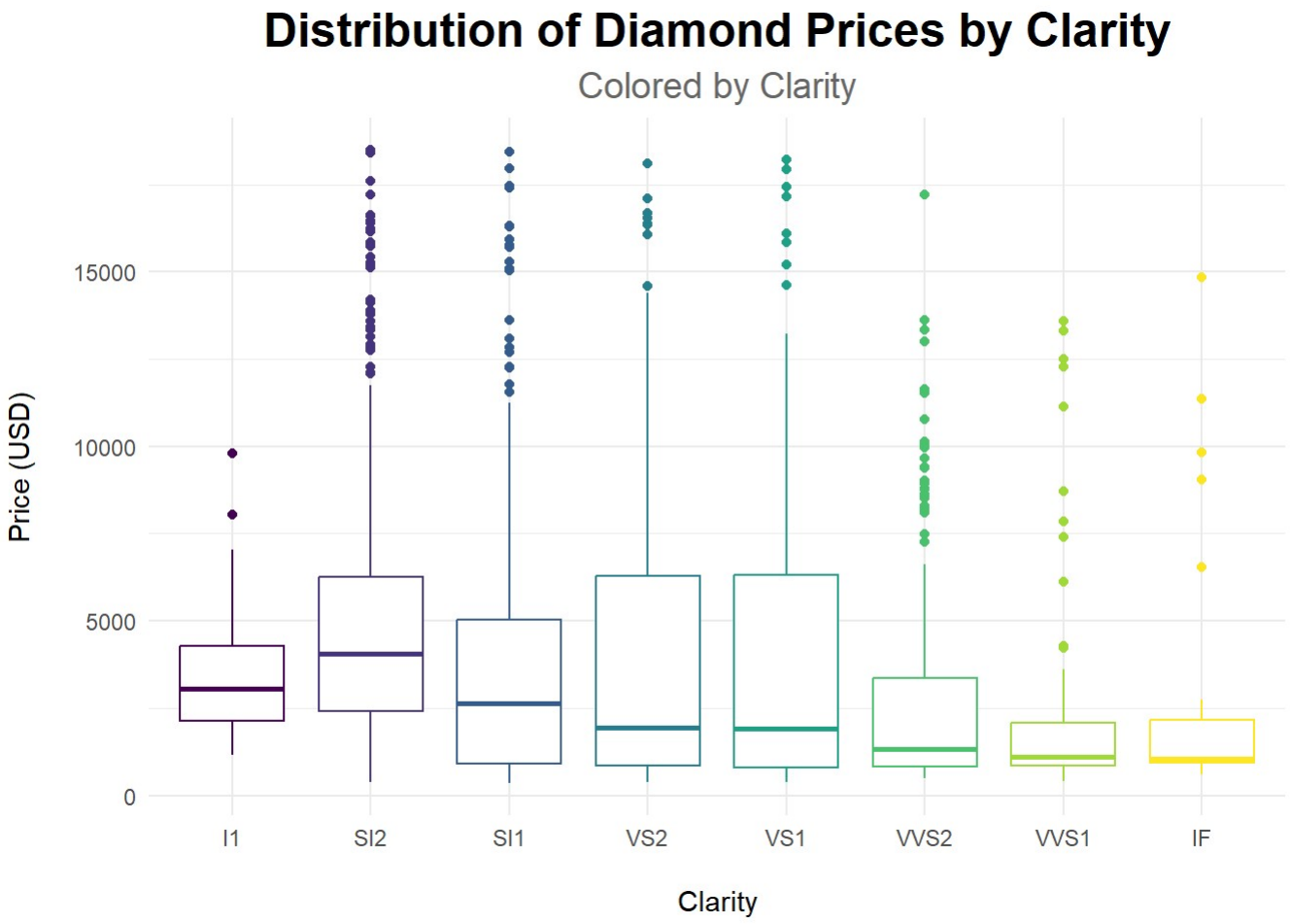


Customizing Box Plots

Once again, we can fully customize a box plot in ggplot2 using `scale_*`, `labs()` and `theme()` and arguments within `geom_boxplot()`. The code below produces a box plot that visualizes the relationship between diamond clarity and price, with clarity represented by distinct colors. A custom color palette is applied via `scale_color_manual()`, ensuring that each clarity level has a unique, high-contrast color. The `labs()` function adds a title, subtitle, axis labels, and a legend title. The `theme_minimal()` function removes background elements and additional customizations are applied through `theme()`.

```r
library(ggplot2)

ggplot(data = diamonds, aes(x = clarity, y = price, color = clarity)) +
  geom_boxplot() +
  scale_color_viridis_d() +   # Use the same custom color scale as the scatter plot
  labs(
    title = "Distribution of Diamond Prices by Clarity",
    subtitle = "Colored by Clarity",
    x = "Clarity",
    y = "Price (USD)",
    color = "Clarity"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(face = "bold", size = 18, hjust = 0.5),
    plot.subtitle = element_text(size = 14, hjust = 0.5, color = "gray40"),
    axis.text.x = element_text(angle = 0, hjust = 0.5),
    axis.title.x = element_text(margin = margin(t = 15)),  # Moves x-axis label downward
    axis.title.y = element_text(margin = margin(r = 15)),  # Moves y-axis label to the left
    legend.position = "none" # removes the legend
  )
```



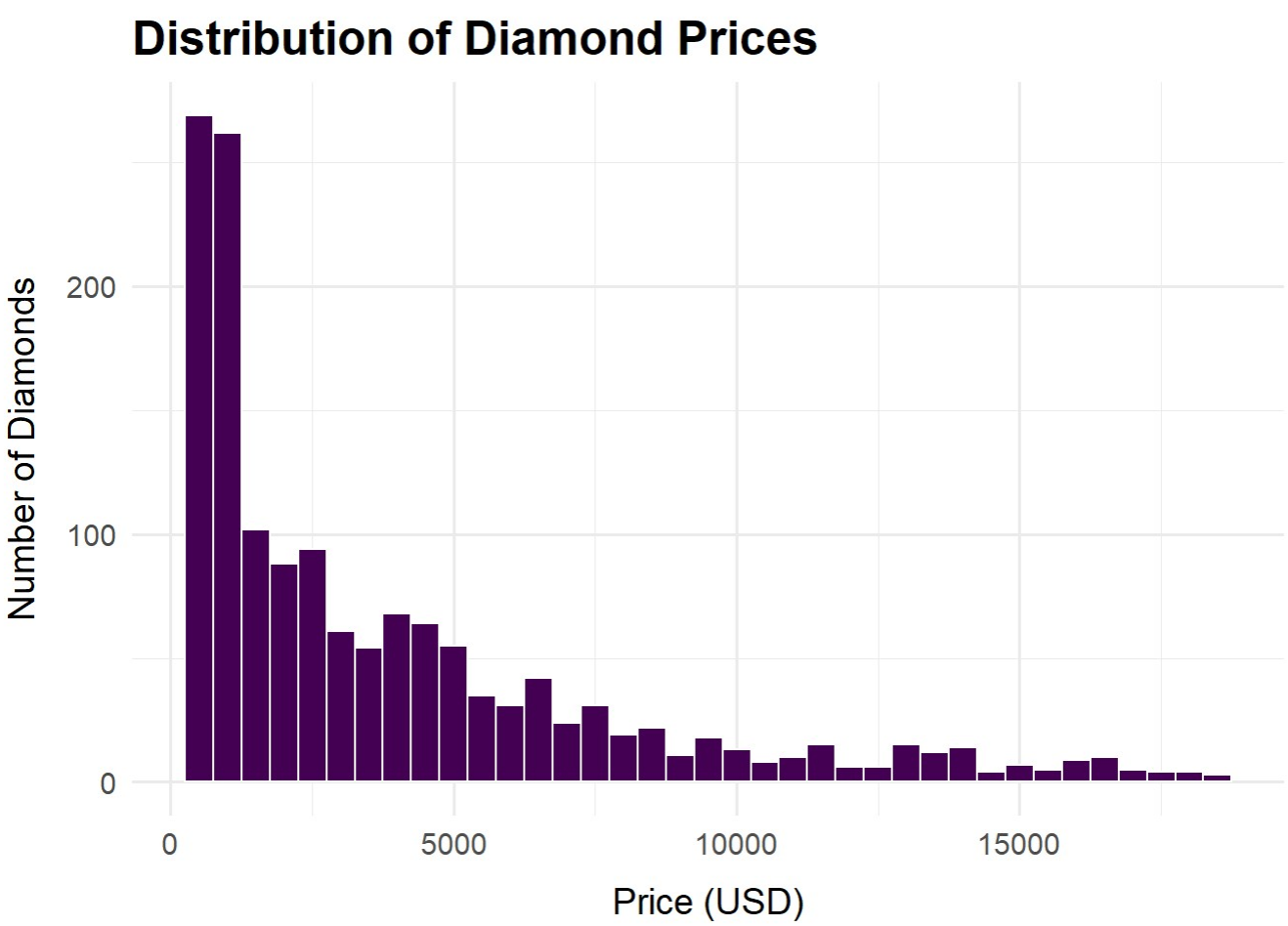
Histograms - One Numeric Variable

A histogram is a way to visualize the distribution of a single numeric variable—in this case, the price of diamonds. Here again, the line `ggplot(diamonds, aes(price))` initializes the plot, telling ggplot to use the diamonds dataset and to map the price variable to the x-axis. Then, `geom_histogram()` creates the histogram. By setting binwidth = 500, we group diamond prices into bins that are $500 wide.

The `labs()` function adds a title and axis labels to make the plot easier to interpret. Finally, `theme_minimal()` applies a clean visual style, and the `theme()` function is used to fine-tune the text formatting.

```
# ggplot(diamonds, aes(price)) + geom_histogram()

ggplot(diamonds, aes(price)) +
  geom_histogram(binwidth = 500, fill = "#440154", color = "white") +
  labs(
    title = "Distribution of Diamond Prices",
#   subtitle = "Sample of ggplot2 diamonds dataset",
    x = "Price (USD)",
    y = "Number of Diamonds",
  ) +
  theme_minimal(base_size = 14) +
  theme(
    plot.title = element_text(face = "bold", size = 18),
    plot.subtitle = element_text(margin = margin(b = 10)),
    axis.title.x = element_text(margin = margin(t = 10)),
    axis.title.y = element_text(margin = margin(r = 10))
  )
```



Recap: Key Functions and Arguments

For reference, the table below summarizes what we covered. These are the key ggplot2 functions and their arguments, which are crucial for building effective data visualizations in R.

- `ggplot()` initializes the plot and requires the data set (data =) and aesthetic mappings (mapping =), which define how variables are mapped to visual elements.
- `aes()` (aesthetic mappings) specifies which variables correspond to graphical properties like x/y position, fill, color, size, and shape of data points.
- `geom_*()` functions, such as geom_point() and geom_bar(), add specific plot types and accept arguments like color, fill, width, and position for customization.
- `scale_*()` functions allow modifications to axes and legend scales, including setting limits, breaks, or transformations (e.g., log scale).
- `labs()` improves readability by adding titles, axis labels, and legend labels.
- `theme()` customizes the plot's appearance, adjusting text elements, legend placement, margins, and overall styling.

By combining these functions and arguments effectively, we can create clear, engaging, and well-structured visualizations tailored to different data storytelling needs.

Function	Essential Arguments
`ggplot()`	`data , mapping`

Function	Essential Arguments
`aes()`	`x`, `y`, `fill`, `color`, `size`, `shape`
`geom_*()`	`mapping`, `data`, `position`, `color`, `fill`, `width`
`scale_*()`	`limits`, `breaks`, `values`, `trans`
`labs()`	`title`, `x`, `y`, `fill`, `color`
`theme()`	`axis.text`, `legend.position`, `plot.title`, `plot.margin`