Multiple_plots

Dr. Niladri Chakraborty

Multiple line plots:

Multiple line plots are a powerful tool in data visualization, allowing you to compare trends and patterns across different groups or variables. In R, you can create multiple line plots using the ggplot2 package, which provides a flexible and intuitive interface for creating high-quality graphics.

Here are the steps to create multiple line plots in R using ggplot2:

1. Load the ggplot2/tidyverse package and prepare the data:

Your data should be in a tidy format, with one column for each variable and one row for each observation. Here is an example dataset that we will use to create our line plots:

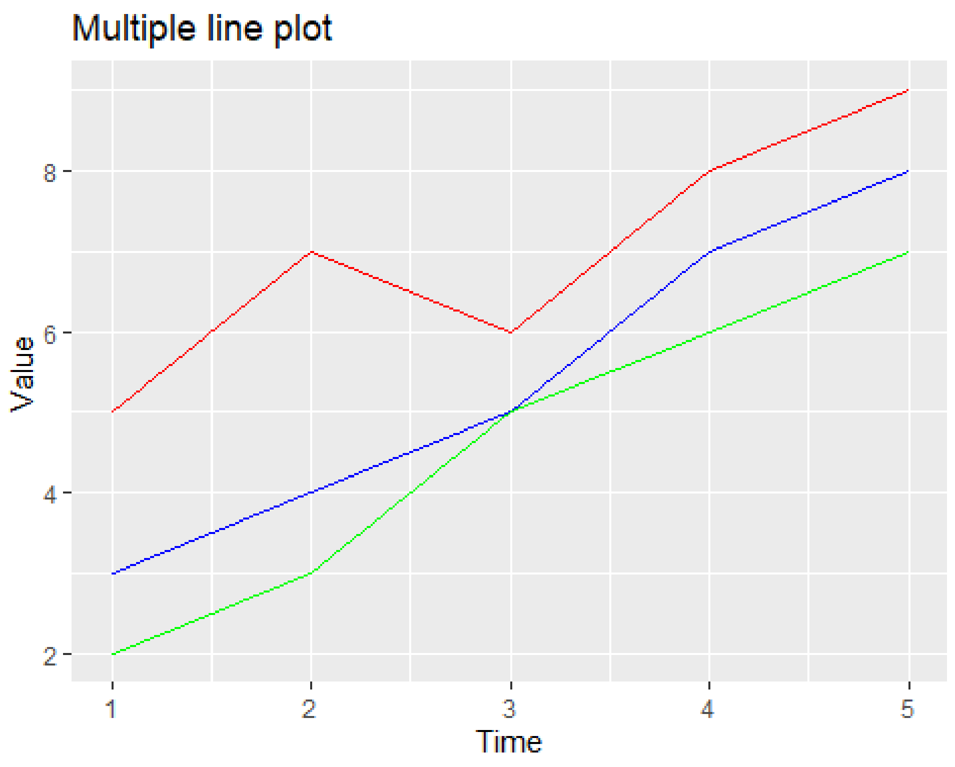
```r
library(ggplot2)
data <- data.frame(
  time = c(1, 2, 3, 4, 5),
  group1 = c(5, 7, 6, 8, 9),
  group2 = c(3, 4, 5, 7, 8),
  group3 = c(2, 3, 5, 6, 7)
)
data

##   time group1 group2 group3
## 1    1      5      3      2
## 2    2      7      4      3
## 3    3      6      5      5
## 4    4      8      7      6
## 5    5      9      8      7
```

This dataset contains five time points and three groups, with each group having a value for each time point.

2. Create the line plot:

To create a line plot, we use the ggplot() function to specify the data and aesthetics (mapping of variables to visual properties), and then add layers to customize the plot. Here is an example code to create a line plot of the three groups over time:

```r
ggplot(data, aes(x = time)) +
  geom_line(aes(y = group1), color = "red") +
  geom_line(aes(y = group2), color = "blue") +
  geom_line(aes(y = group3), color = "green") +
  labs(x = "Time", y = "Value", title = "Multiple line plot")
```

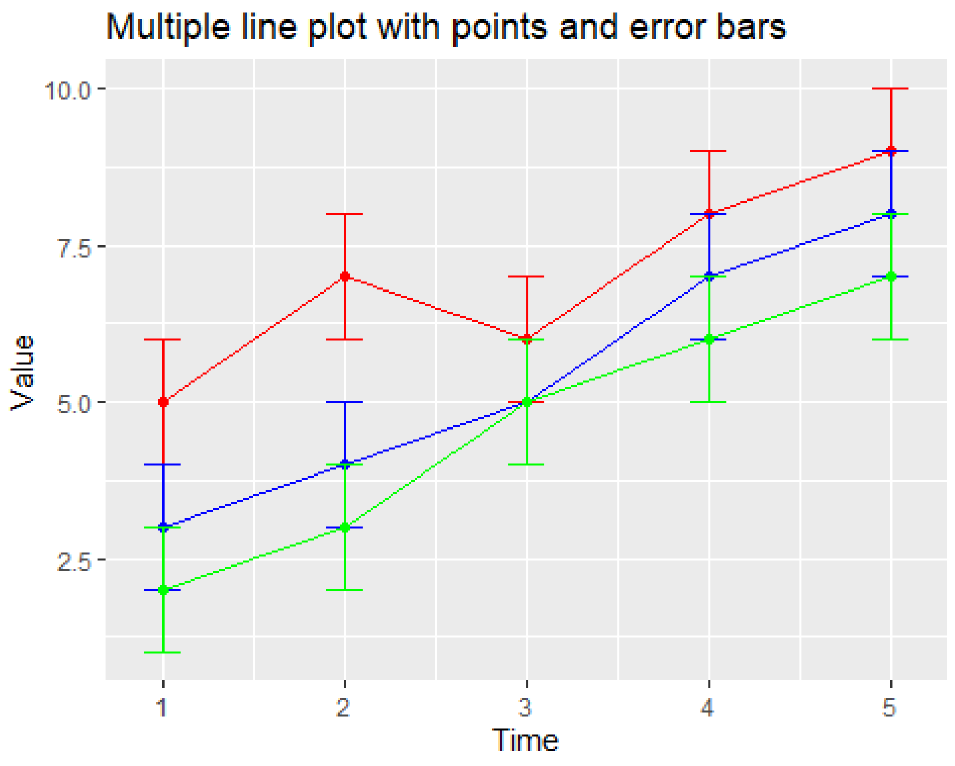


This code creates a line plot with the x-axis representing time, the y-axis representing the value of each group, and each group represented by a different colored line. We use the geom_line() function to add each line, specifying the color and the variable mapped to the y-axis using the aes() function. We also use the labs() function to add labels to the axes and the title.

3. Customize the plot:

You can customize the plot by adding additional layers, such as points, error bars, or text labels, and by adjusting the visual properties, such as the colors, sizes, and fonts. Here are some examples:

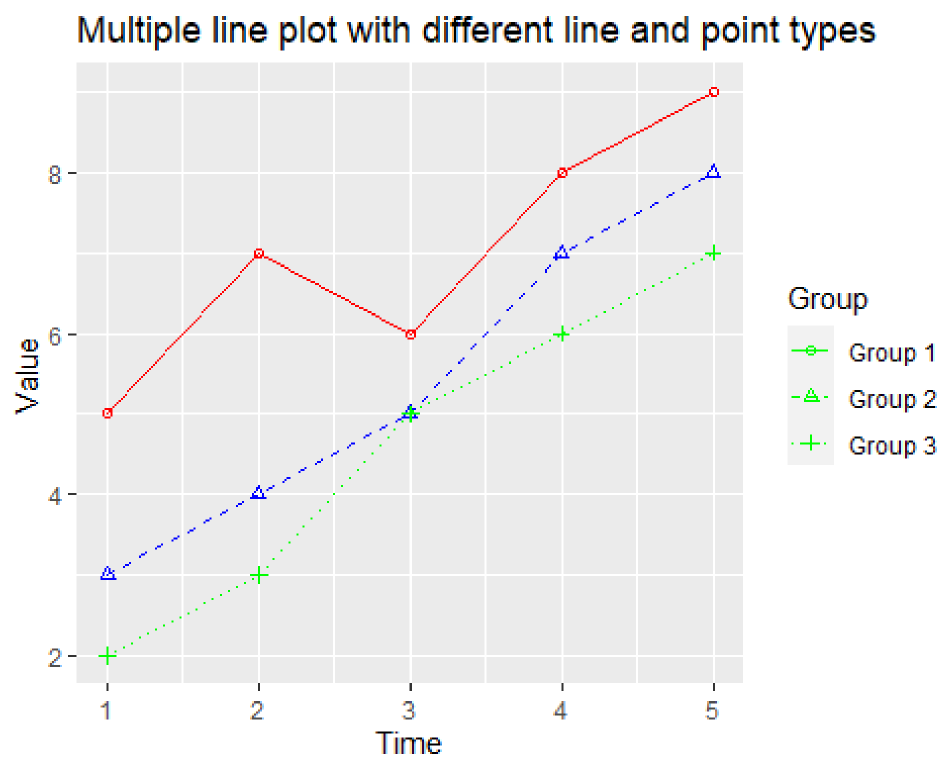
```
# Add points and error bars to the plot
ggplot(data, aes(x = time, y = group1)) +
  geom_line(color = "red") +
  geom_point(aes(y = group1), color = "red") +
  geom_errorbar(aes(ymin = group1 - 1, ymax = group1 + 1), width = 0.2, color
= "red") +
  geom_line(aes(y = group2), color = "blue") +
  geom_point(aes(y = group2), color = "blue") +
  geom_errorbar(aes(ymin = group2 - 1, ymax = group2 + 1), width = 0.2, color
= "blue") +
  geom_line(aes(y = group3), color = "green") +
  geom_point(aes(y = group3), color = "green") +
  geom_errorbar(aes(ymin = group3 - 1, ymax = group3 + 1), width = 0.2, color
= "green") +
  labs(x = "Time", y = "Value", title = "Multiple line plot with points and
error bars")
```



4. Change the line and the point types:

To change the line types and point types in the above plot, you can use the linetype and shape arguments within the geom_line() and geom_point() functions, respectively. Here is an example code to change the line types and point types:

```
ggplot(data, aes(x = time, y = group1)) +
  geom_line(aes(linetype = "Group 1"), color = "red") +
  geom_point(aes(y = group1, shape = "Group 1"), color = "red") +
  geom_line(aes(y = group2, linetype = "Group 2"), color = "blue") +
  geom_point(aes(y = group2, shape = "Group 2"), color = "blue") +
  geom_line(aes(y = group3, linetype = "Group 3"), color = "green") +
  geom_point(aes(y = group3, shape = "Group 3"), color = "green") +
  labs(x = "Time", y = "Value", title = "Multiple line plot with different
line and point types") +
  scale_linetype_manual(name = "Group", values = c("solid", "dashed",
"dotted")) +
  scale_shape_manual(name = "Group", values = c(1, 2, 3))
```

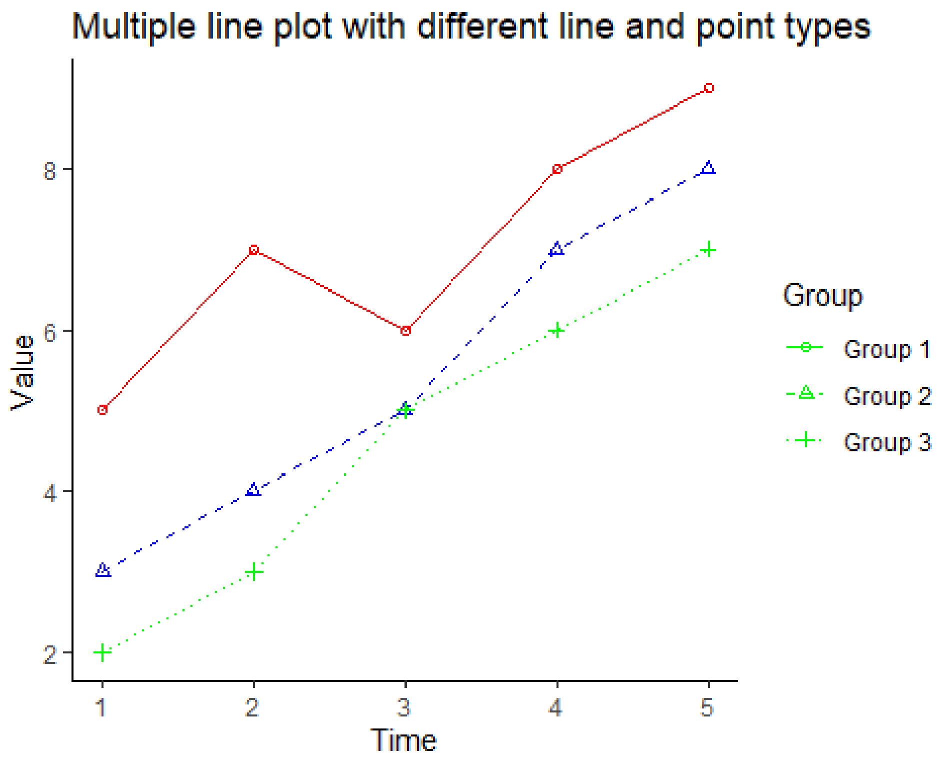


In this code, we use the linetype and shape arguments within the aes() function to map the line types and point types to the different groups. We also use the scale_linetype_manual() and scale_shape_manual() functions to customize the line and point types, respectively. The values argument within these functions allows us to specify the different types that we want to use, and the name argument allows us to add a legend to the plot that labels the different groups.

5. Change the background:

We can change the background using the 'hrbrthemes' package.

```
library(hrbrthemes)

## NOTE: Either Arial Narrow or Roboto Condensed fonts are required to use
these themes.

##       Please use hrbrthemes::import_roboto_condensed() to install Roboto
Condensed and

##       if Arial Narrow is not on your system, please see
https://bit.ly/arialnarrow

ggplot(data, aes(x = time, y = group1)) +
  geom_line(aes(linetype = "Group 1"), color = "red") +
  geom_point(aes(y = group1, shape = "Group 1"), color = "red") +
  geom_line(aes(y = group2, linetype = "Group 2"), color = "blue") +
  geom_point(aes(y = group2, shape = "Group 2"), color = "blue") +
  geom_line(aes(y = group3, linetype = "Group 3"), color = "green") +
  geom_point(aes(y = group3, shape = "Group 3"), color = "green") +
  labs(x = "Time", y = "Value", title = "Multiple line plot with different
line and point types") +
  scale_linetype_manual(name = "Group", values = c("solid", "dashed",
"dotted")) +
  scale_shape_manual(name = "Group", values = c(1, 2, 3))+
  theme_classic()
```



Multiple Histograms:

Multiple histograms are a type of visualization that is useful for comparing the distribution of several variables or groups. You need to load the package ‘ggplot2’ or ‘tidyverse’.

Then you will need to create a data frame with the variables you want to plot. In this example, we will create a data frame with three variables (x1, x2, and x3) with 100 observations each.

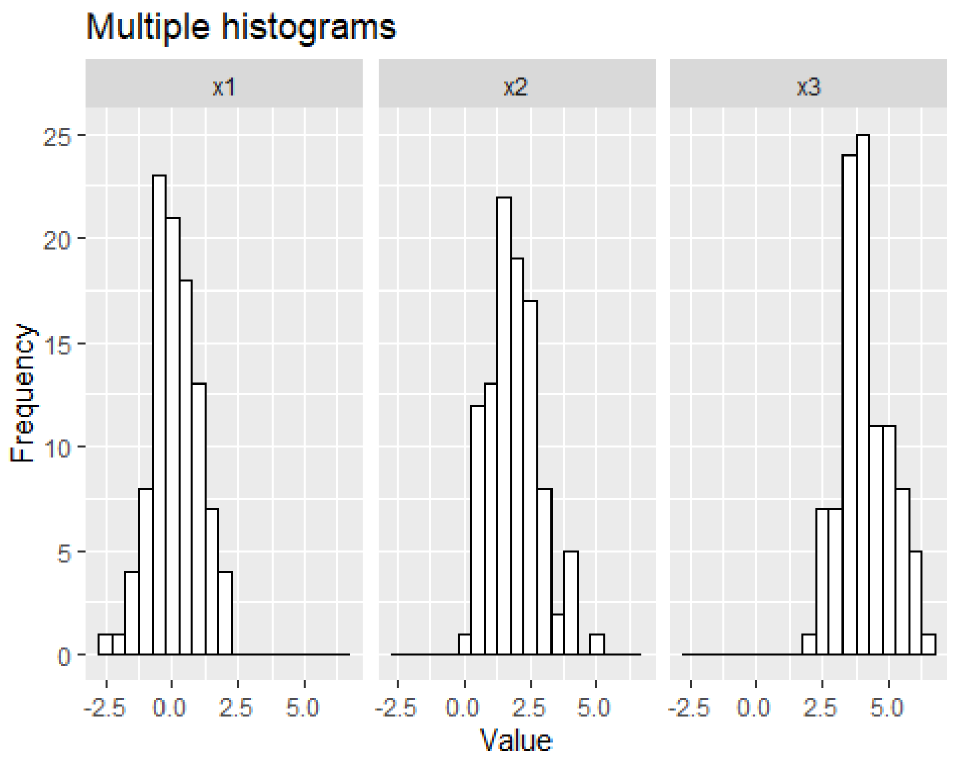
```
set.seed(123)  # for reproducibility
data <- data.frame(x1 = rnorm(100), x2 = rnorm(100, mean = 2), x3 =
rnorm(100, mean = 4))
head(data,10)

##              x1        x2       x3
## 1  -0.56047565 1.2895934 6.198810
## 2  -0.23017749 2.2568837 5.312413
## 3   1.55870831 1.7533081 3.734855
## 4   0.07050839 1.6524574 4.543194
## 5   0.12928774 1.0483814 3.585660
## 6   1.71506499 1.9549723 3.523753
## 7   0.46091621 1.2150955 3.211397
## 8  -1.26506123 0.3320581 3.405383
## 9  -0.68685285 1.6197735 5.650907
## 10 -0.44566197 2.9189966 3.945972
```

Reshape the data: The ggplot2 package expects the data to be in a "long" format, where each row represents one observation and one variable. You can use the tidyr package to reshape the data from "wide" to "long" format.

```r
library(tidyr)
data_long <- gather(data, key = "variable", value = "value")
```

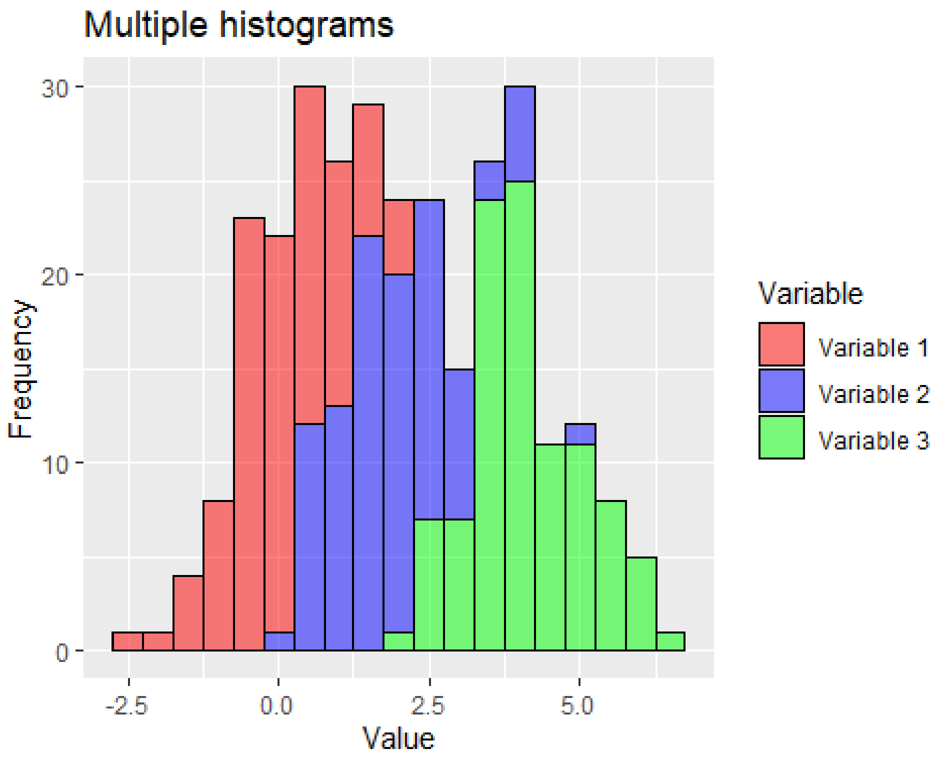
Create the histogram: You can use the ggplot() function to create the plot. Use the geom_histogram() function to create the histogram, and use the facet_wrap() function to split the plot by variable.

```r
ggplot(data_long, aes(x = value)) +
  geom_histogram(binwidth = 0.5, color = "black", fill = "white") +
  facet_wrap(~ variable, nrow = 1) +
  labs(x = "Value", y = "Frequency", title = "Multiple histograms")
```



To get different colors for different histograms and a legend in the above plot, you can use the fill aesthetic within the aes() function to specify the fill color for each group, and the scale_fill_manual() function to specify the colors for each group in the legend. Here is an example code to add different colors and a legend to the plot:

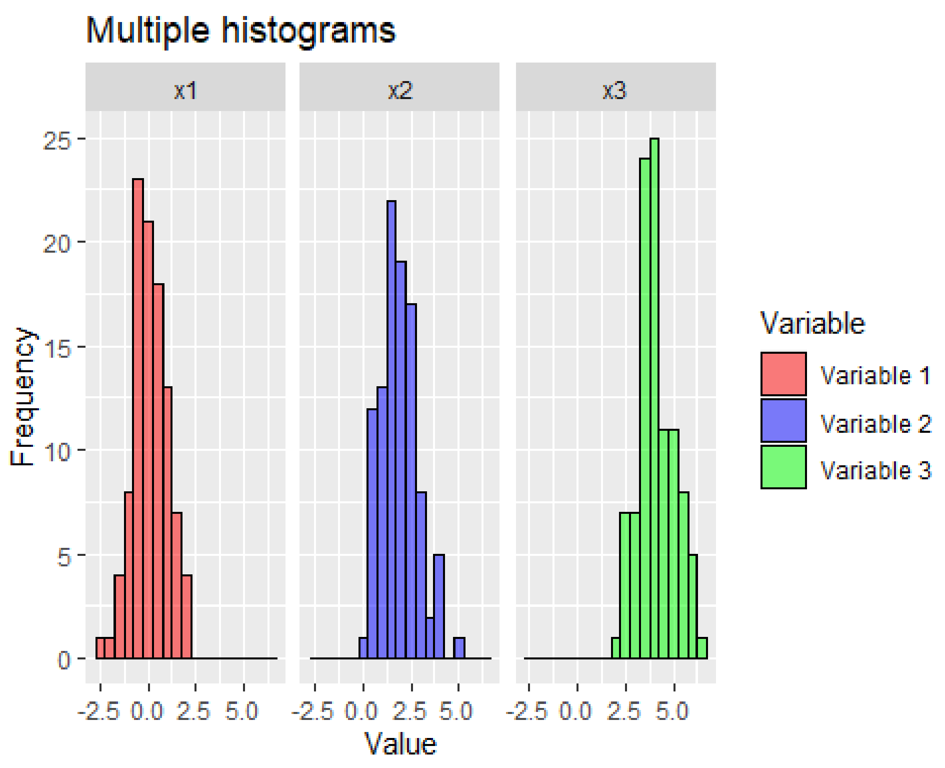
```r
ggplot(data_long, aes(x = value, fill = variable)) +
  geom_histogram(binwidth = 0.5, color = "black", alpha = 0.5) +
  labs(x = "Value", y = "Frequency", title = "Multiple histograms") +
  scale_fill_manual(name = "Variable",
                    values = c("red", "blue", "green"),
                    labels = c("Variable 1", "Variable 2", "Variable 3"))
```



In this code, we use the fill aesthetic within the aes() function to specify the fill color for each group. We also set the alpha parameter to 0.5 to make the bars slightly transparent. Finally, we use the scale_fill_manual() function to specify the colors for each group in the legend. The name argument within this function allows us to specify the name of the legend, the values argument allows us to specify the colors for each group, and the labels argument allows us to specify the labels for each group in the legend.

You split the plots with facet_wrap() function and add different colors as well.

```r
ggplot(data_long, aes(x = value, fill = variable)) +
  geom_histogram(binwidth = 0.5, color = "black", alpha = 0.5) +
  facet_wrap(~ variable, nrow = 1) +
  labs(x = "Value", y = "Frequency", title = "Multiple histograms") +
  scale_fill_manual(name = "Variable",
                    values = c("red", "blue", "green"),
                    labels = c("Variable 1", "Variable 2", "Variable 3"))
```



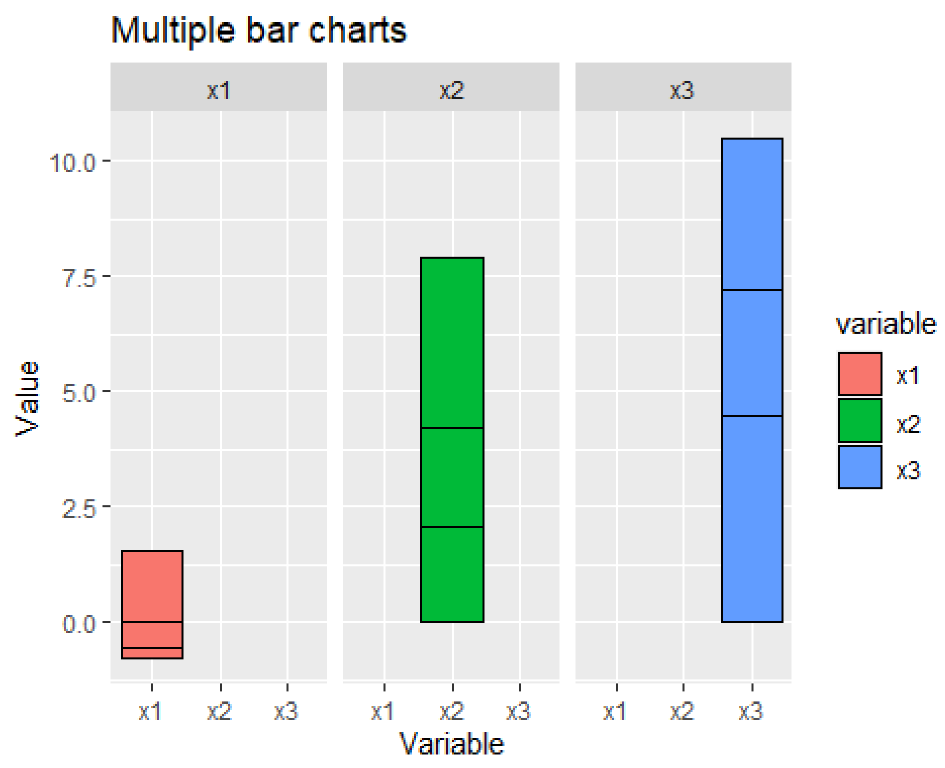
Multiple bar charts

Multiple bar charts are a type of visualization that is useful for comparing the values of several variables or groups.

```
set.seed(123)  # for reproducibility
data <- data.frame(x1 = rnorm(3), x2 = rnorm(3, mean = 2), x3 = rnorm(3, mean
= 4))
data

##           x1       x2       x3
## 1 -0.5604756 2.070508 4.460916
## 2 -0.2301775 2.129288 2.734939
## 3  1.5587083 3.715065 3.313147

library(tidyr)
data_long <- gather(data, key = "variable", value = "value")
```

You can use the ggplot() function to create the plot. Use the geom_bar() function to create the bar chart, and use the facet_wrap() function to split the plot by variable.

```
ggplot(data_long, aes(x = variable, y = value, fill = variable)) +
  geom_bar(stat = "identity", color = "black") +
  facet_wrap(~ variable, nrow = 1) +
  labs(x = "Variable", y = "Value", title = "Multiple bar charts")
```



Mixture plot:

Next we create a mixture of a histogram and a density plot. We want to have a histogram for a set of random sample from N(0,1) distribution. Then we want to fit an exponential(1) distribution to the data. Then we want to plot the histogram and the density curve together.

Remember $F(x) \sim U(0,1)$ for any random variable X. So we draw a random sample from the N(0,1) distribution of size 1000 and calculate the $F(x)$ with the pnorm() function. That is how we get the $U(0,1)$ values.

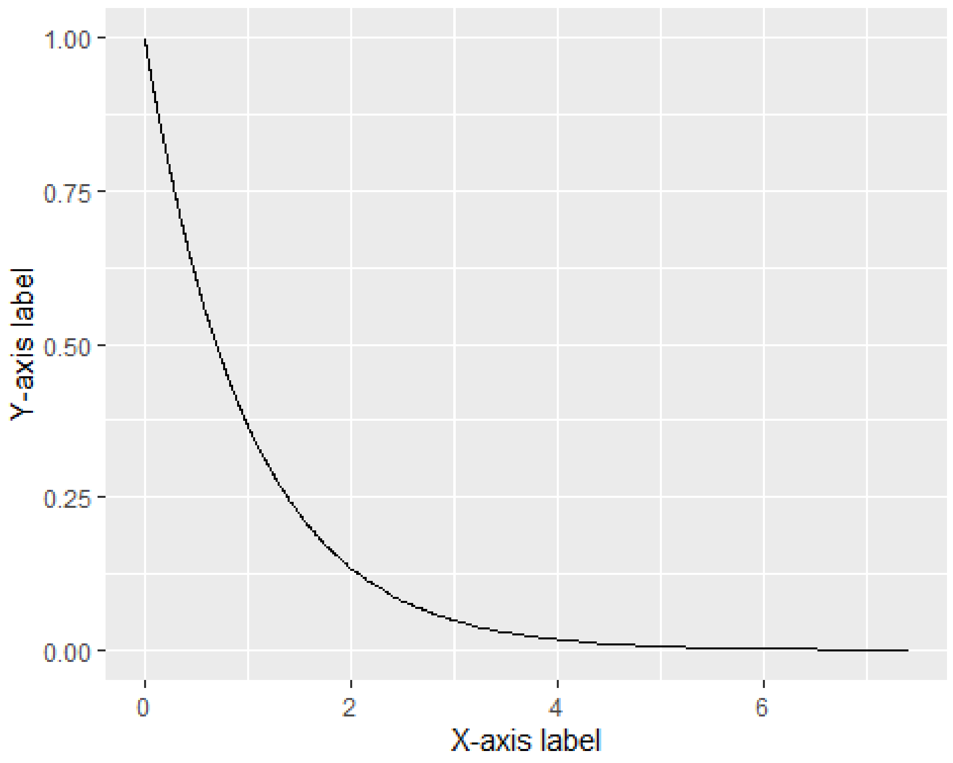
Then, based on these $U(0,1)$ values, we obtain corresponding exponential random variable values using the quantile qexp() function. This will give us a set of random sample from the exponential distribution. Then, for each of those exponential random variable values, we calculate the exponential density with the dexp() function.

```
x = rnorm(1000)
prob = pnorm(sort(x))

data1 = data.frame(x,prob)

ex = qexp(prob)

prob1 = dexp(ex)
```

```
data2 = data.frame(ex,prob1)
```

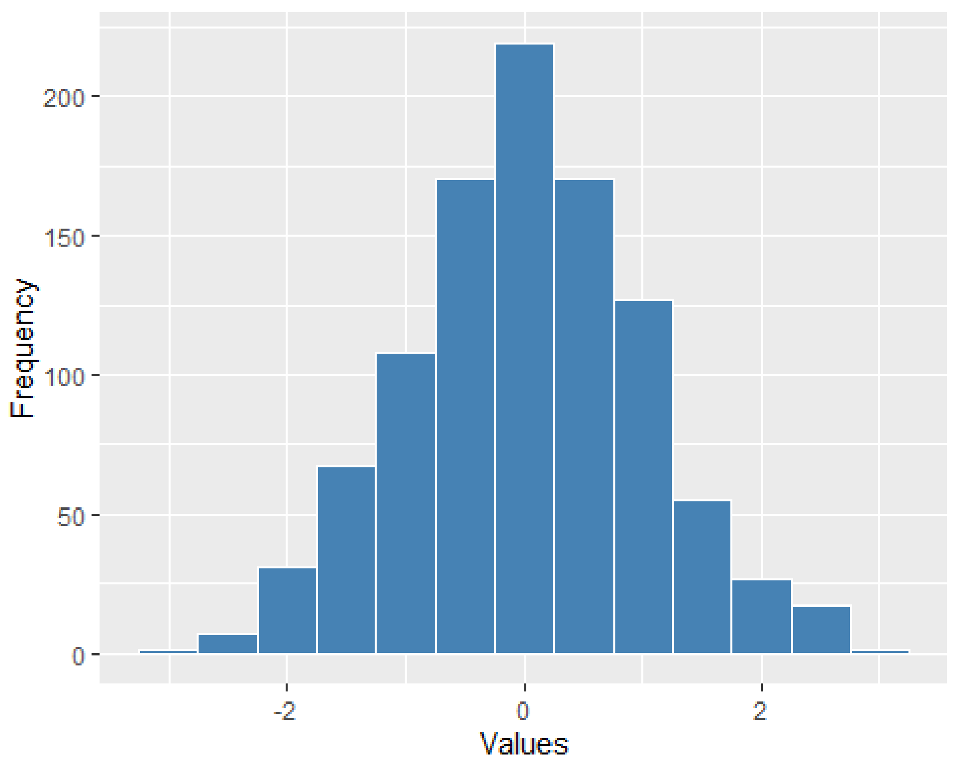
Next, we create the density curve for the exponential random sample.

```
# Create line chart
line_plot <- ggplot(data = data2, aes(x = ex, y = prob1)) +
  geom_line() +
  labs(x = "X-axis label", y = "Y-axis label")

line_plot
```



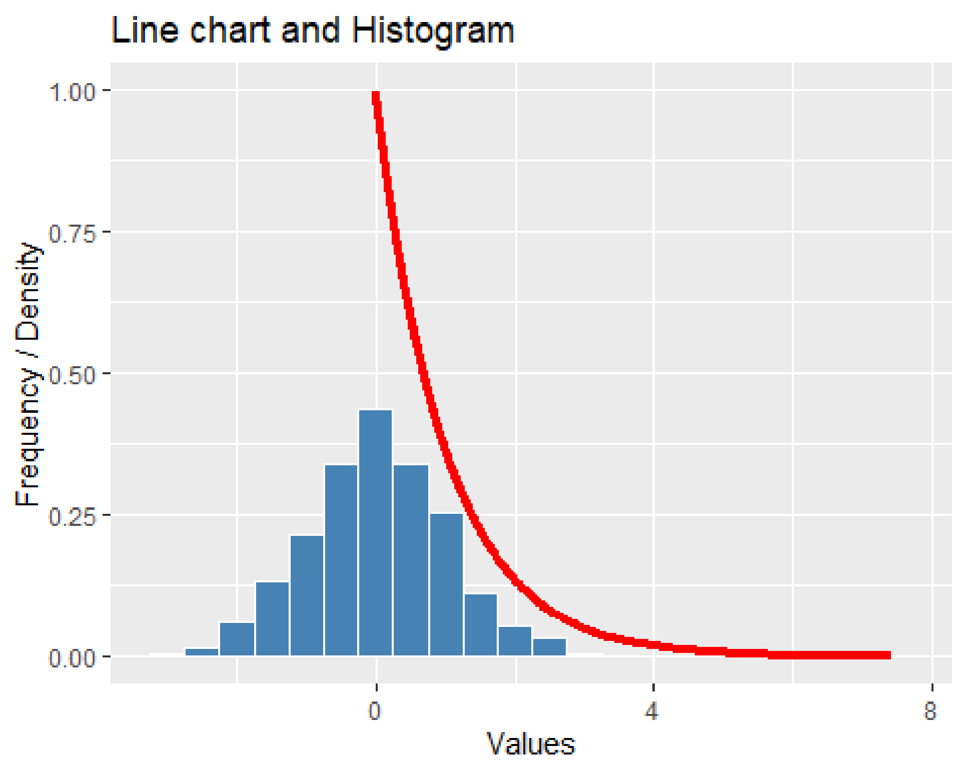
Next, we create the histogram for the N(0,1) random sample.

```
# Create histogram
histogram <- ggplot(data = data1, aes(x = x)) +
  geom_histogram(binwidth = 0.5, fill = "steelblue", color = "white") +
  labs(x = "Values", y = "Frequency")
histogram
```



Then, we combine the two plots.

```
combined_plot <- ggplot() +
  geom_histogram(data = data1, aes(x = x, y = ..density..), binwidth = 0.5,
fill = "steelblue", color = "white") +
  geom_line(data = data2, aes(x = ex, y = prob1), color = "red", size = 1.5)
+
  labs(x = "Values", y = "Frequency / Density", title = "Line chart and
Histogram")

## 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.

combined_plot

## Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2
3.4.0.
## ℹ Please use `after_stat(density)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



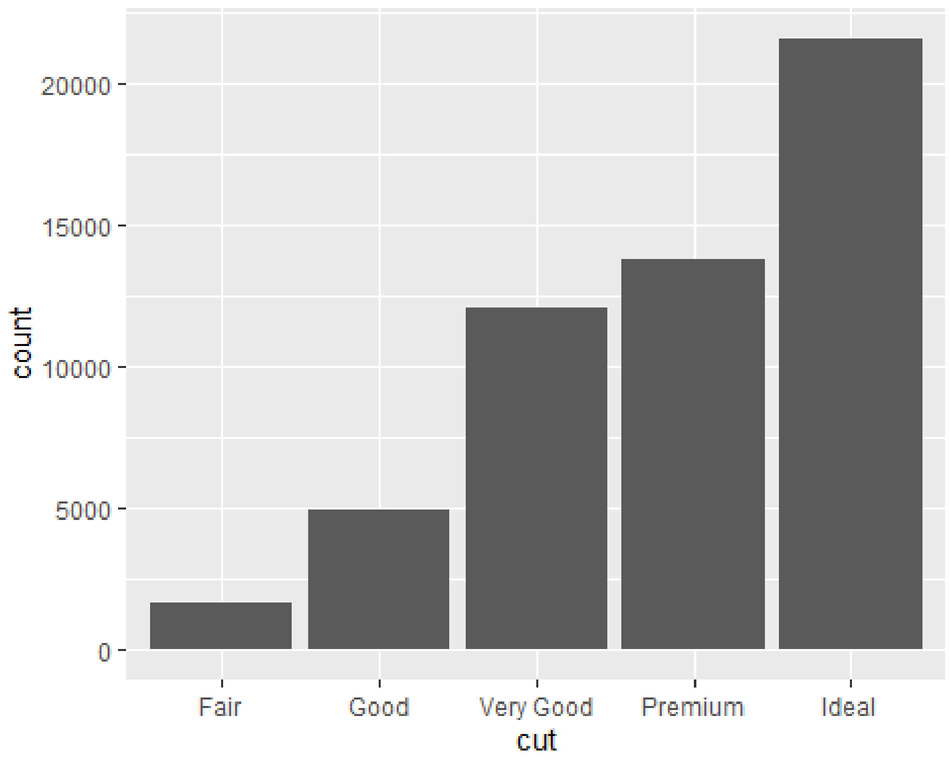
Creating barplots with the ‘diamonds’ data in tidyverse package

You need to call the tidyverse package to use the dataset diamonds.

```
library(tidyverse)

## ── Attaching core tidyverse packages ──────────────────────── tidyverse
2.0.0 ──
## ✓ dplyr     1.1.2     ✓ readr     2.1.4
## ✓ forcats   1.0.0     ✓ stringr   1.5.0
## ✓ lubridate 1.9.2     ✓ tibble    3.2.1
## ✓ purrr     1.0.1
## ── Conflicts ──────────────────────────────────────────
tidyverse_conflicts() ──
## ✕ dplyr::filter() masks stats::filter()
## ✕ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all
conflicts to become errors

data = diamonds
data

## # A tibble: 53,940 × 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
##  7  0.24 Very Good I     VVS1     62.3    57    336  3.95  3.98  2.47
##  8  0.26 Very Good H     SI1      61.9    55    337  4.07  4.11  2.53
##  9  0.22 Fair      E     VS2      65.1    61    337  3.87  3.78  2.49
## 10  0.23 Very Good H     VS1      59.4    61    338  4     4.05  2.39
## # ℹ 53,930 more rows
```

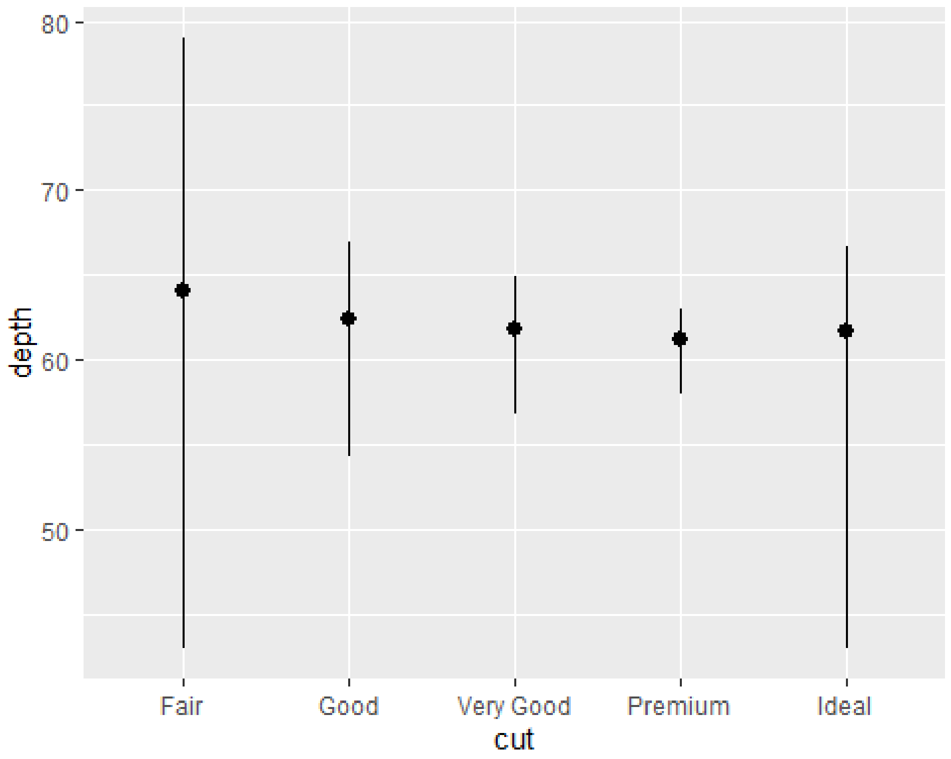
A simple bar plot can be created as follows:

```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut))
```



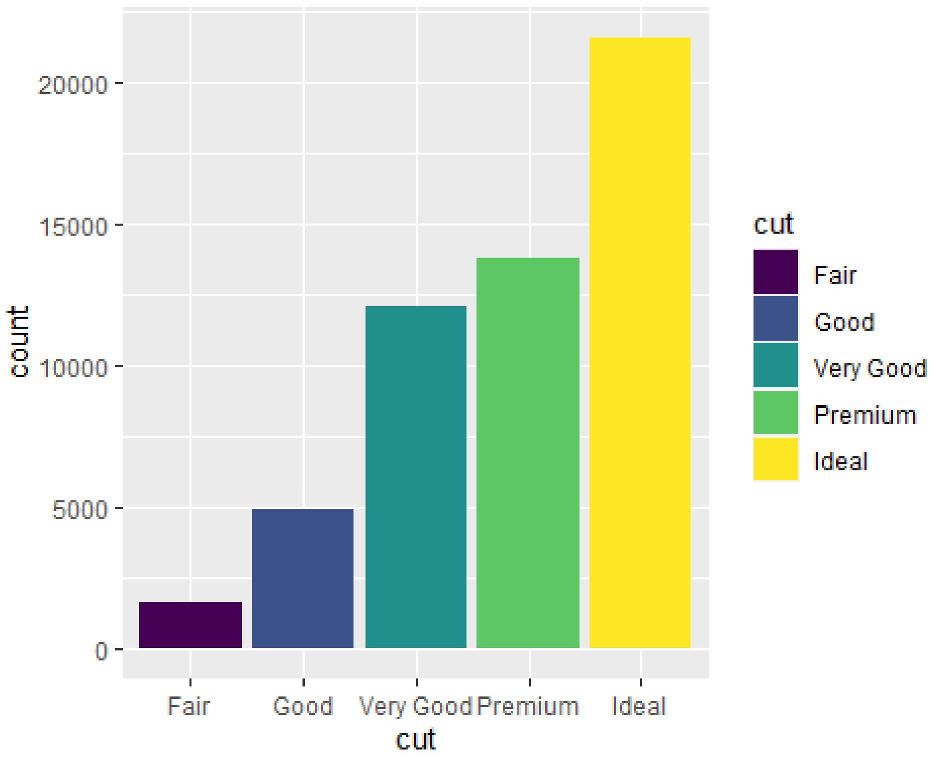
You might want to draw greater attention to the statistical transformation in your code. For example, you might use stat_summary(), which summarises the y values for each unique x value,

```
ggplot(data = diamonds) +
  stat_summary(
    mapping = aes(x = cut, y = depth),
    fun.min = min,
    fun.max = max,
    fun = mean
  )
```



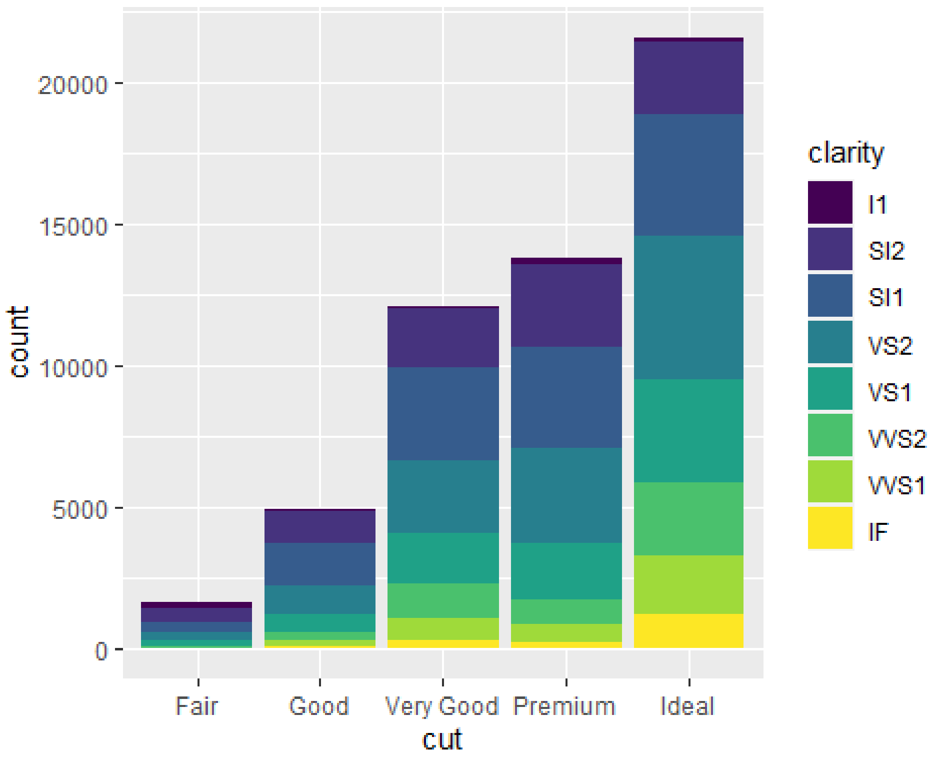
You can colour a bar chart using either the colour aesthetic, or, more usefully, fill:

```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = cut))
```



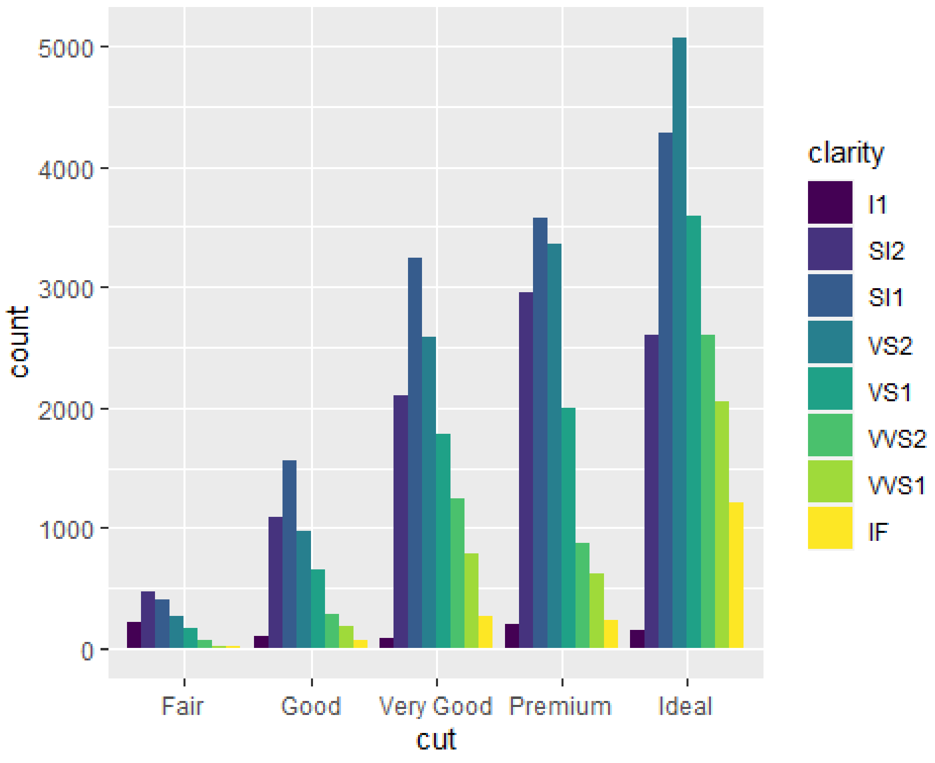
what happens if you map the fill aesthetic to another variable, like clarity: the bars are automatically stacked. Each colored rectangle represents a combination of cut and clarity.

```r
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity))
```



position = “dodge” places overlapping objects directly beside one another. This makes it easier to compare individual values.

```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity), position = "dodge")
```


5000
4000
3000
2000
1000
0
count
Fair
Good
Very Good
Premium
Ideal
cut
clarity
I1
SI2
SI1
VS2
VS1
VVS2
VVS1
IF