R Bootcamp - Course 3: Basic Statistics in R

James Baglin

Last updated: 13 July, 2020

Overview

Summary

This course will familiarise yourself with using basic statistical functions in R. The course will cover functions for descriptive statistics and data visualisation of quantitative and qualitative type variables.

Learning Objectives

By the end of this course, you will have completed the following:

- Basic descriptive statistics and visualisations for qualitative variables:
  - frequencies
  - cross-tabulations
  - bar charts
  - clustered bar charts
- Basic descriptive statistics and visualisations for quantitative variables:
  - Dot plots
  - Histograms
  - Mean and standard deviation
  - Quartiles, median, and IQR
  - Box plots
  - Side-by-side box plots
  - Scatter plots

Data

This module will utilise the following data sources:

Diamonds

The `Diamonds` (data/Diamonds.csv) dataset includes the prices and other attributes of almost 54,000 diamonds. The variables are as follows:

- **price**: price in US dollars ($326 - $18,823)
- **carat**: weight of the diamond (0.2 - 5.01)
- **cut**: quality of the cut (Fair, Good, Very Good, Premium, Ideal)
- **colour**: diamond colour, from J (worst) to D (best)
- **clarity**. a measurement of how clear the diamond is (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best))
- **x**: length in mm (0–10.74)
- **y**: width in mm (0–58.9)
- **z**: depth in mm (0–31.8)
- **depth**: total depth percentage = z / mean(x, y) = 2 * z / (x + y) (43–79)
- **table**: width of top of diamond relative to widest point (43–95)

Import data:

```
Diamonds <- read.csv("data/Diamonds.csv")
```

Define factors:

```
Diamonds$cut<- factor(Diamonds$cut, levels=c('Fair','Good','Very Good','P
  remium','Ideal'),
                      ordered=TRUE)

Diamonds$color<- factor(Diamonds$color, levels=c('J','I','H','G','F','E',
  'D'),
                        ordered=TRUE)

Diamonds$clarity<- factor(Diamonds$clarity,
                          levels=c('I1','SI2','SI1','VS2','VS1','VVS2','V
  VS1','IF'),
                          ordered=TRUE)
```

This code ensures that cut, colour and clarity variables are treated correctly as ordered qualitative factors.

Here is a random sample of the full dataset:

Show 10 ▼ **entries** **Search:**

	carat	cut	color	clarity	depth	table	price	x
1	0.23	Premium	E	VVS2	61.8	58	680	3.94
2	0.51	Premium	F	VS2	61.4	58	1619	5.13
3	0.76	Ideal	E	SI2	61.5	57	2680	5.83
4	0.54	Very Good	G	SI2	59.5	60	1083	5.27
5	1.24	Ideal	G	VS2	62	56	8797	6.87
6	0.31	Ideal	G	VVS1	61.8	54	789	4.38
7	0.53	Ideal	G	SI1	62.4	55	1329	5.21
8	1.5	Very Good	H	VS1	63.3	59	9173	7.15
9	1.33	Ideal	J	VS1	62.4	55	6190	7.02
10	0.5	Premium	G	VVS2	60.9	61	1935	5.13

Showing 1 to 10 of 100 entries

Previous 1 2 3 4 5 … 10 Next

Qualitative Variables

Frequencies

Let's summarise the colour variable by counting the frequencies of different colour diamonds in the dataset. We use the base `table()` function:

```
Diamonds$color %>% table()
```

```
## .
##     J     I     H     G     F     E     D
##  2808  5422  8304 11292  9542  9797  6775
```

Now, what if we want proportions reported instead? We can use the `prop.table()` function:

```
Diamonds$color %>% table() %>% prop.table()
```

```
## .
##          J          I          H          G          F          E
D
## 0.05205784 0.10051910 0.15394883 0.20934372 0.17690026 0.18162773 0.12
560252
```

Percentages?

```
Diamonds$color %>% table() %>% prop.table()*100
```

```
## .
##         J         I         H         G         F         E         D
##  5.205784 10.051910 15.394883 20.934372 17.690026 18.162773 12.560252
```

Easy!

Bar charts

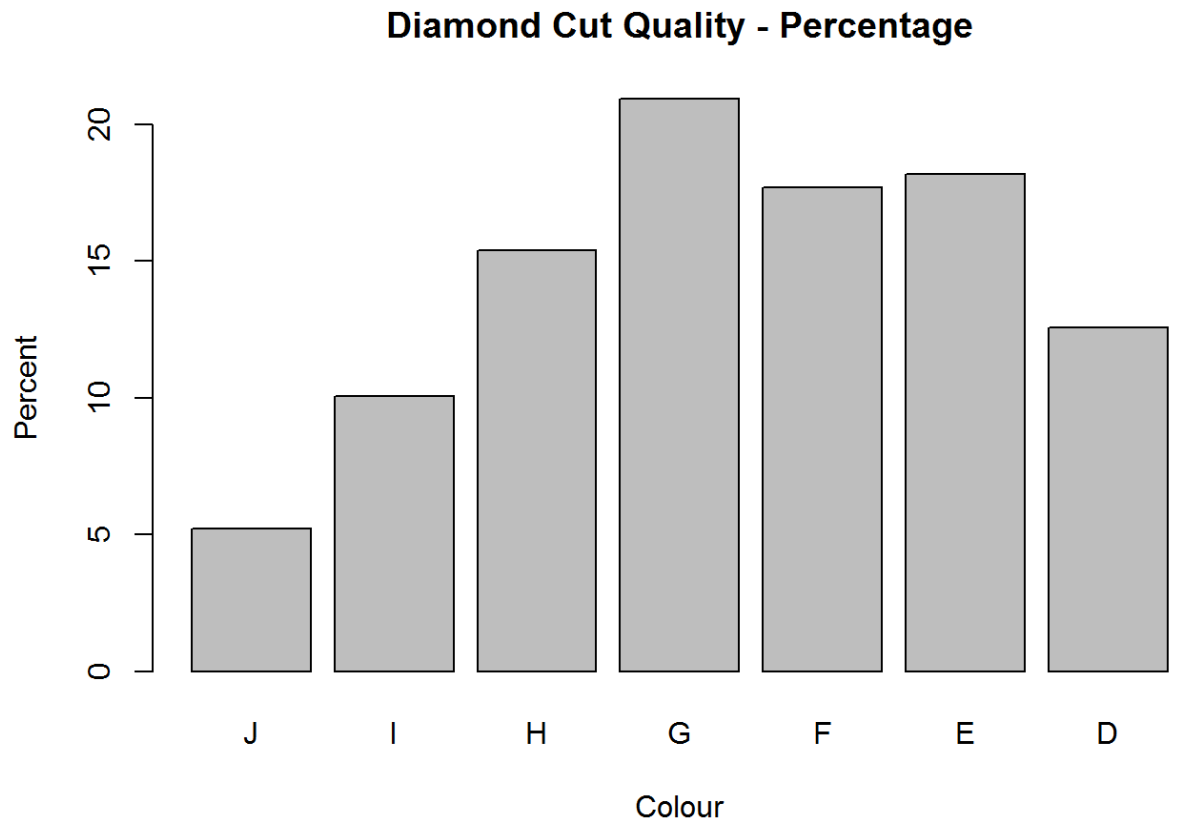
To create a bar chart summarising the percentage of colours in the sample, we first save the percentages into an table object, called `freq` .

```
freq <- Diamonds$color %>% table() %>% prop.table()*100
```

After running this code, you should see "freq" appear in the environment window under Values.

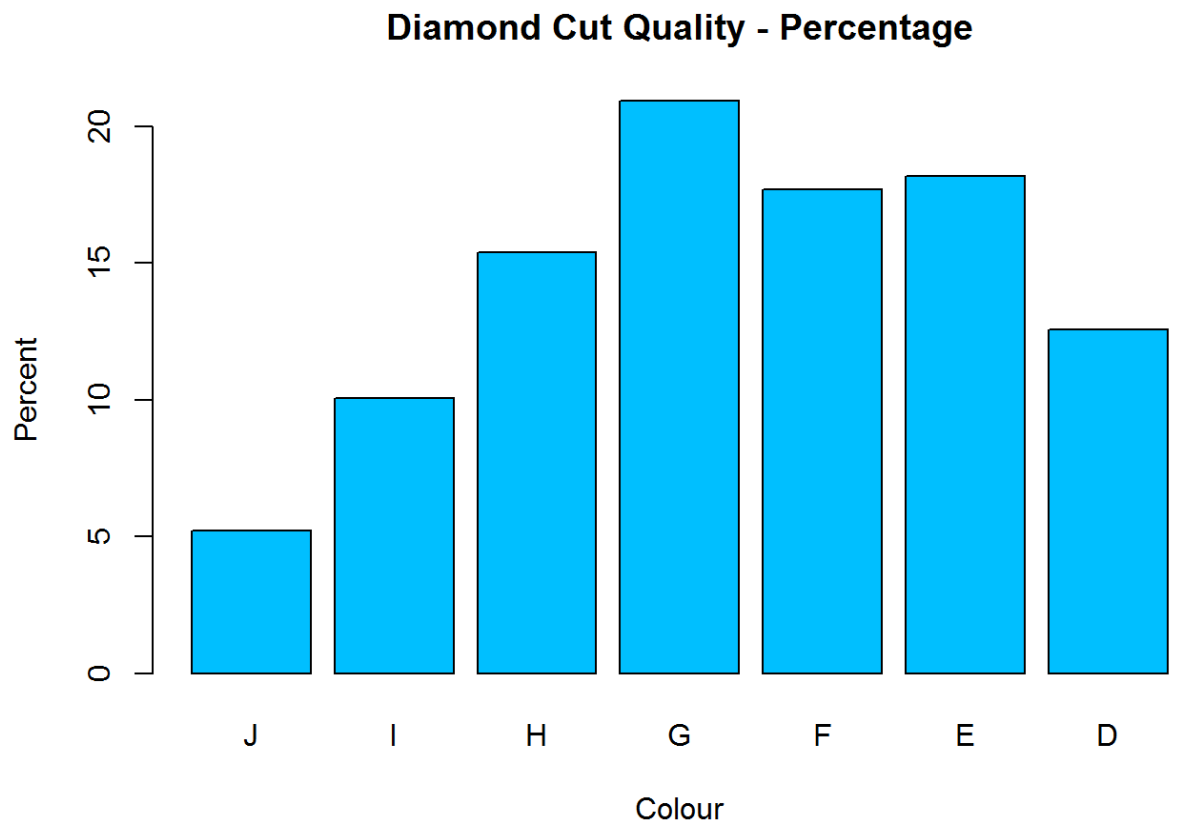
Next, we create a bar chart using this object.

```
barplot(freq, main = "Diamond Cut Quality - Percentage",ylab="Percent", x
  lab="Colour")
```



Notice how the title, y and x axis labels were defined using `main`, `ylab` and `xlab`, respectively. If you want to add some colour use the `col =` option...

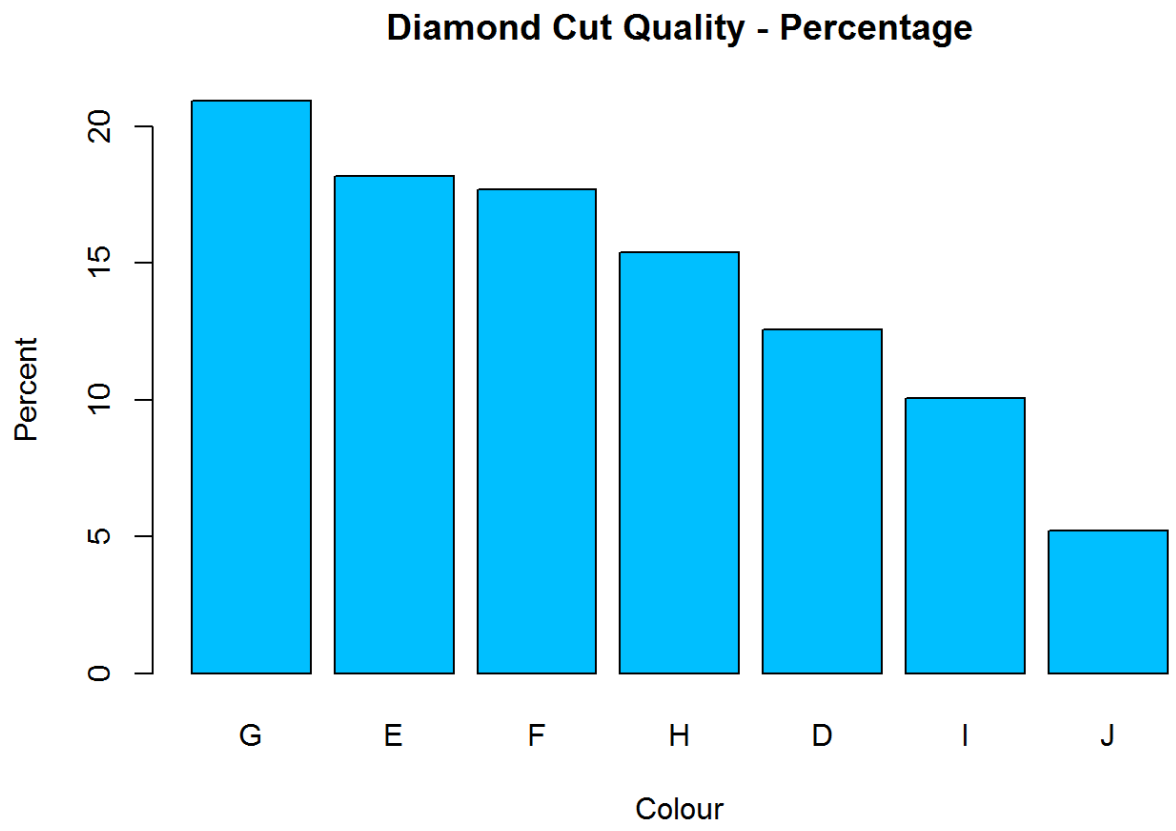
```
barplot(freq, main = "Diamond Cut Quality - Percentage",
        ylab="Percent", xlab="Colour", col="deepskyblue")
```



You can see all the colour options here (http://research.stowers-institute.org/efg/R/Color/Chart/ColorChart.pdf).

Order by frequency?

```
barplot(freq[order(freq, decreasing = TRUE)], main = "Diamond Cut Quality
  - Percentage",
        ylab="Percent", xlab="Colour", col="deepskyblue")
```



Cross-tabulations

Contingency tables or cross tabulations organise two qualitative variables into rows and columns. For example, what if we wanted to look at the relationship between colour and cut? We can modify the `table()` function to report this:

```
table(Diamonds$color,Diamonds$cut) %>% prop.table(margin = 2)
```

```
##
##           Fair       Good  Very Good    Premium      Ideal
##   J 0.07391304 0.06257644 0.05611654 0.05858893 0.04157580
##   I 0.10869565 0.10640033 0.09965238 0.10354579 0.09711846
##   H 0.18819876 0.14309009 0.15096838 0.17112610 0.14454086
##   G 0.19503106 0.17753771 0.19028307 0.21202233 0.22662521
##   F 0.19378882 0.18528333 0.17910942 0.16902328 0.17753237
##   E 0.13913043 0.19017530 0.19864261 0.16945834 0.18110529
##   D 0.10124224 0.13493681 0.12522761 0.11623523 0.13150202
```

The `margin = 2` option reports column proportions.

That's a little overzealous with the decimal places. We use the `round()` function to limit the table to three decimal places:

```
table(Diamonds$color,Diamonds$cut) %>% prop.table(margin = 2) %>% round(3
  )
```

```
##
##      Fair  Good Very Good Premium Ideal
##   J 0.074 0.063     0.056   0.059 0.042
##   I 0.109 0.106     0.100   0.104 0.097
##   H 0.188 0.143     0.151   0.171 0.145
##   G 0.195 0.178     0.190   0.212 0.227
##   F 0.194 0.185     0.179   0.169 0.178
##   E 0.139 0.190     0.199   0.169 0.181
##   D 0.101 0.135     0.125   0.116 0.132
```

Better!

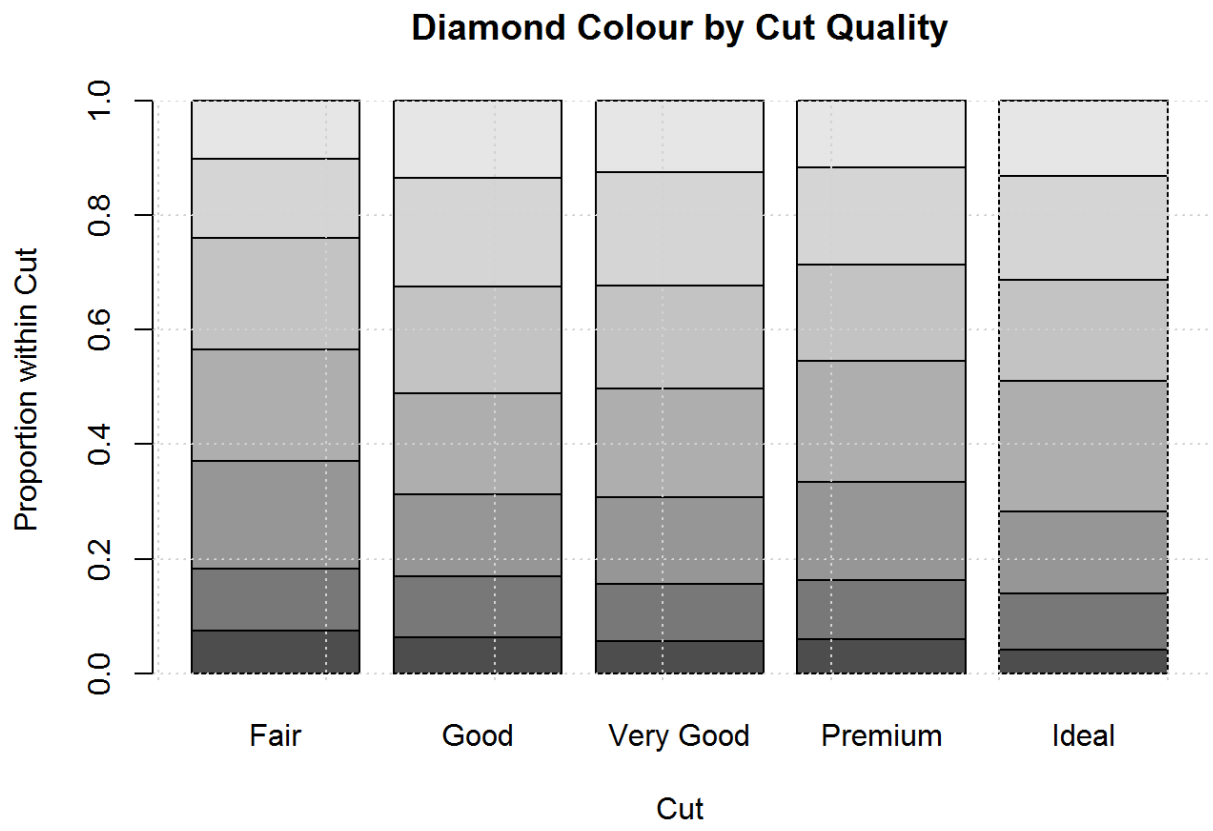
Clustered Bar Charts

Clustered bar plots are a great way to help understand the relationships between two categorical variables. First, we save the contingency table as an object called `color_cut` :

```
color_cut <- table(Diamonds$color,Diamonds$cut) %>% prop.table(margin = 2
  )
```

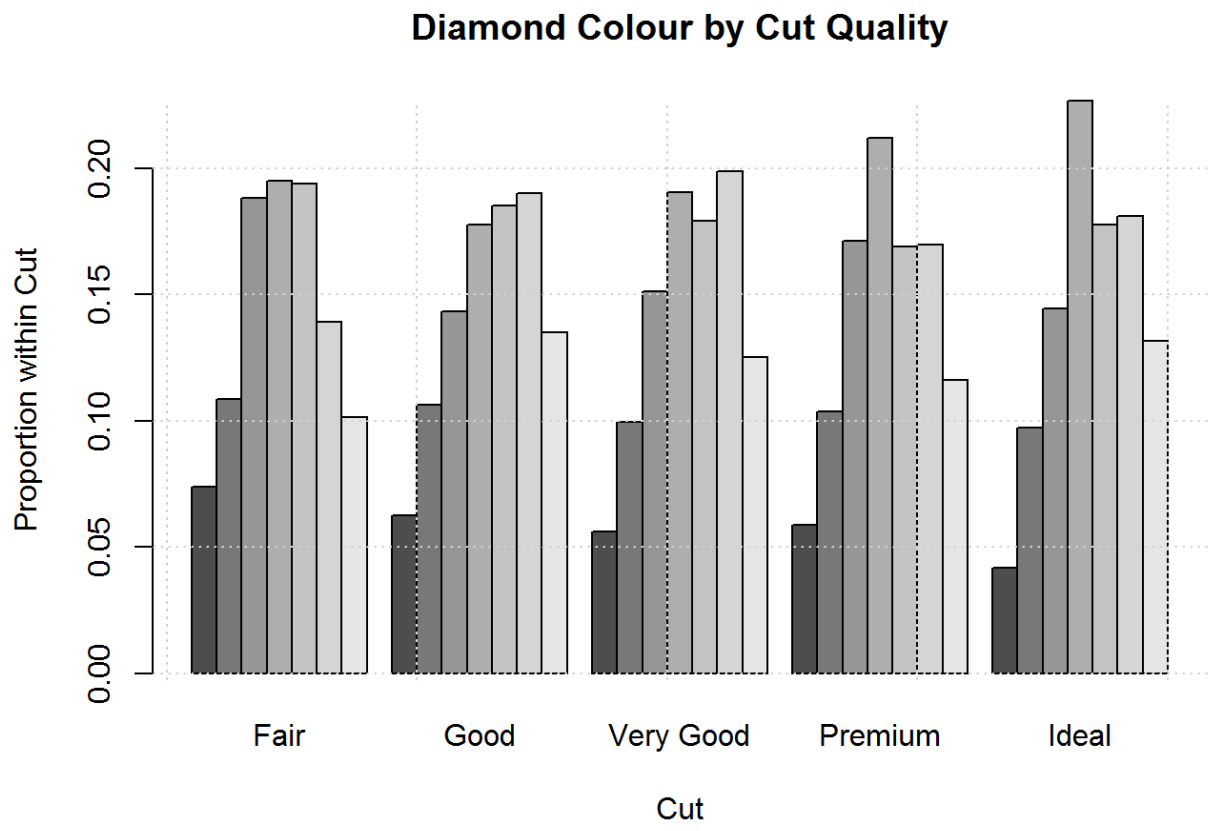
Now for the plot:

```
barplot(color_cut, main = "Diamond Colour by Cut Quality",
          ylab="Proportion within Cut", xlab="Cut")
grid()
```



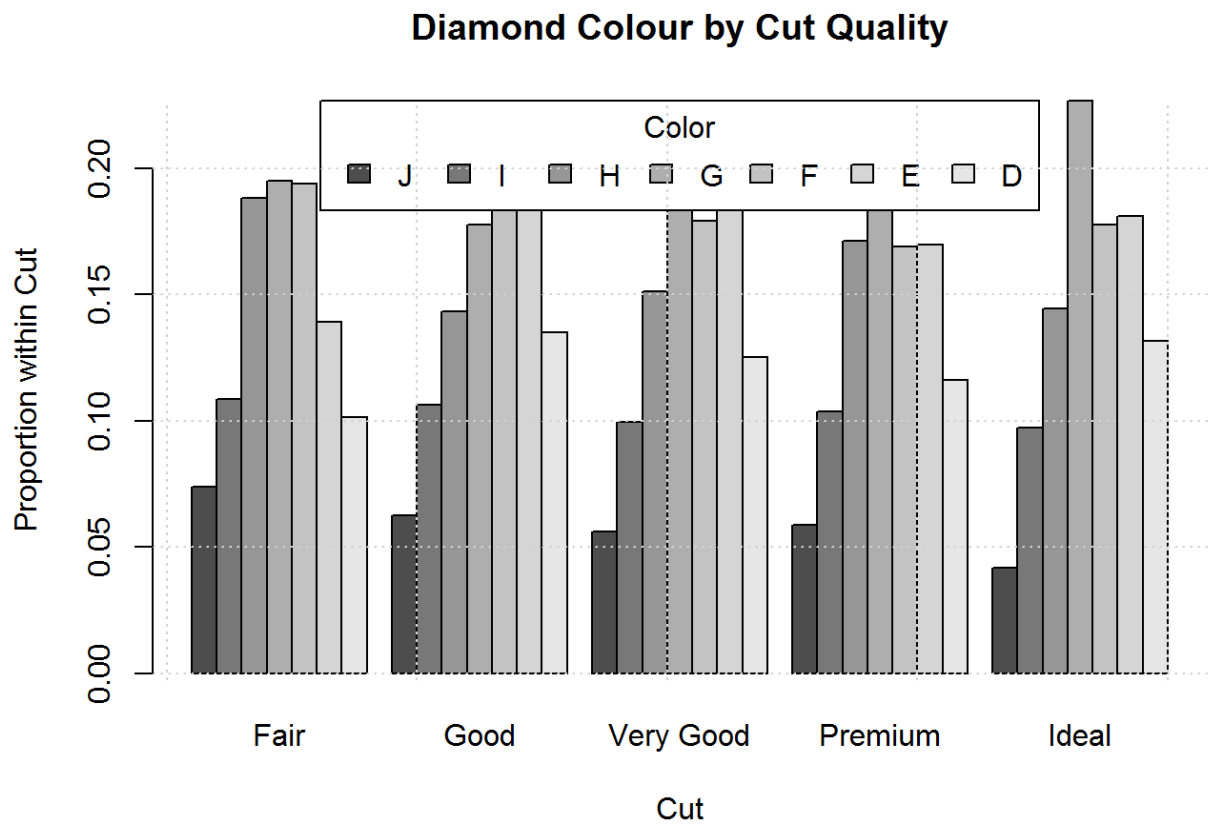
We add the `grid()` option to make the plot easier to read values off the axis. Not quite right. Let's add `beside = TRUE` to align the bars next to each other:

```
barplot(color_cut, main = "Diamond Colour by Cut Quality",
        ylab="Proportion within Cut",
        xlab="Cut", beside=TRUE)
grid()
```



Better, but we need to add a legend:

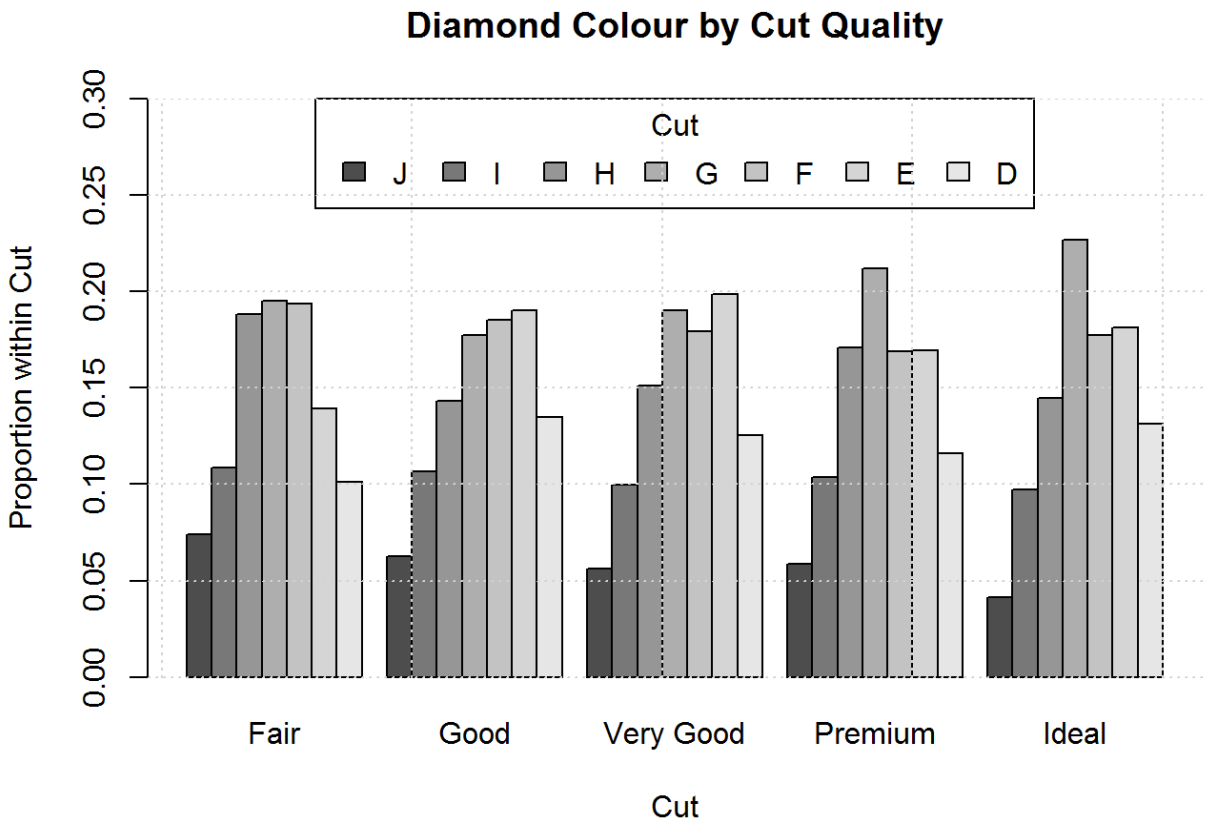
```
barplot(color_cut, main = "Diamond Colour by Cut Quality",ylab="Proportio
  n within Cut",
        xlab="Cut", beside=TRUE, legend=rownames(color_cut),
        args.legend=c(x = "top",horiz=TRUE,title="Color"))
grid()
```



We added `legend = rownames(color_cut)`. This labels the legend using the row names (diamond colour names). The `args.legend()` option instructs R where to place the legend relative to the plot and its title.

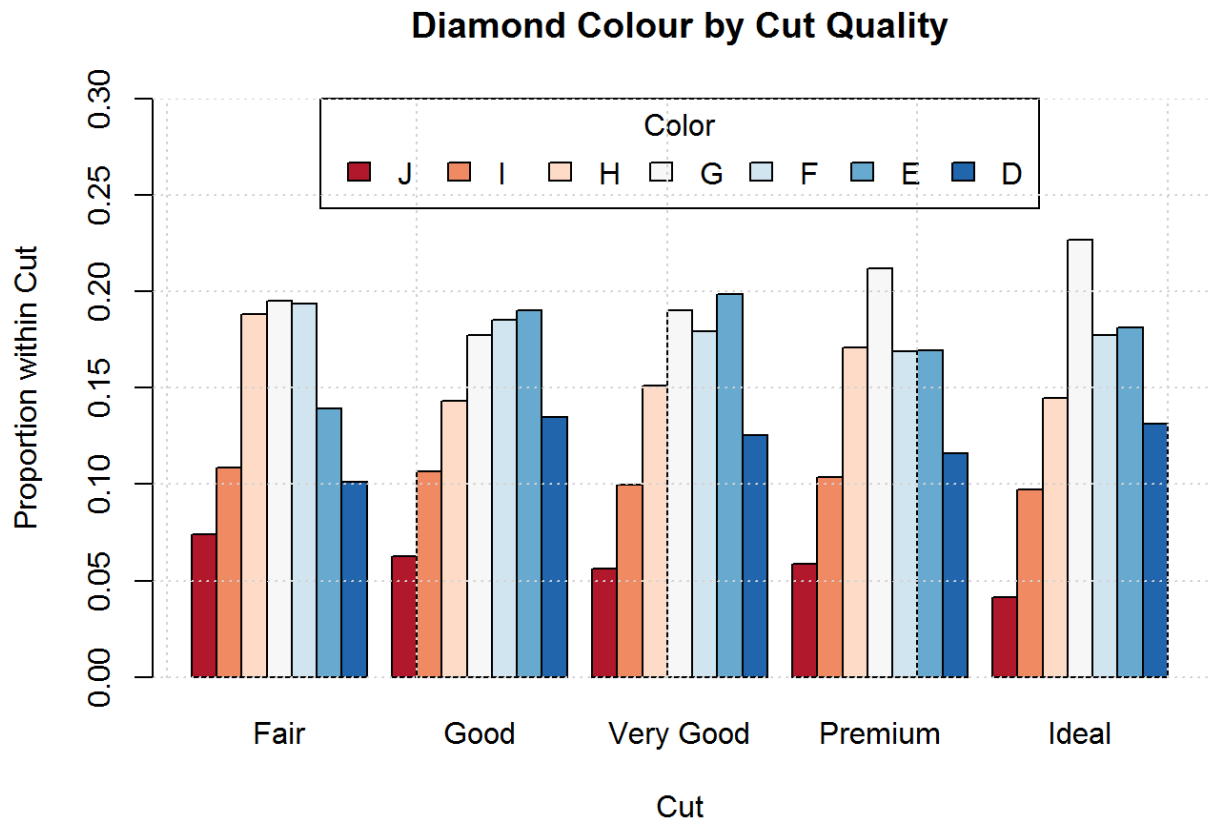
Getting closer... However, the legend overlaps the bars! Let's adjust the y-axis to increase the white space in the top of the plot, using the `ylim` option.

```
barplot(color_cut, main = "Diamond Colour by Cut Quality",ylab="Proportio
  n within Cut",
        xlab="Cut", beside=TRUE, legend=rownames(color_cut),
        args.legend=c(x = "top",horiz=TRUE,title="Cut"), ylim = c(0,.30))
grid()
```



Better, but it's hard to differentiate between the grey colours! Let's use some more discernible colours. We can use a colour Brewer palette from the `RColorBrewer` package (ensure you install this package first). For example, we can set a diverging blue-red palette using `col=brewer.pal(n = 7, name = "RdBu")`.

```
library(RColorBrewer)
barplot(color_cut, main = "Diamond Colour by Cut Quality",ylab="Proportio
  n within Cut",
        xlab="Cut", beside=TRUE,
        legend=rownames(color_cut),
        args.legend=c(x = "top",horiz=TRUE,title="Color"),
        ylim = c(0,.30),
        col=brewer.pal(7, name = "RdBu"))
grid()
```



Quantiative Variables

Dot plots

Dot plots are a great visualisation for small samples. For example, let's create a dot plot of a random sample of 30 diamonds' mass measured in carats. We make use of the `sample_n()` function from the `dplyr` package, which we will re-visit in Module 5. Before we do this, we will set the seed to the random number generator built into R:

```
set.seed(4532)
```

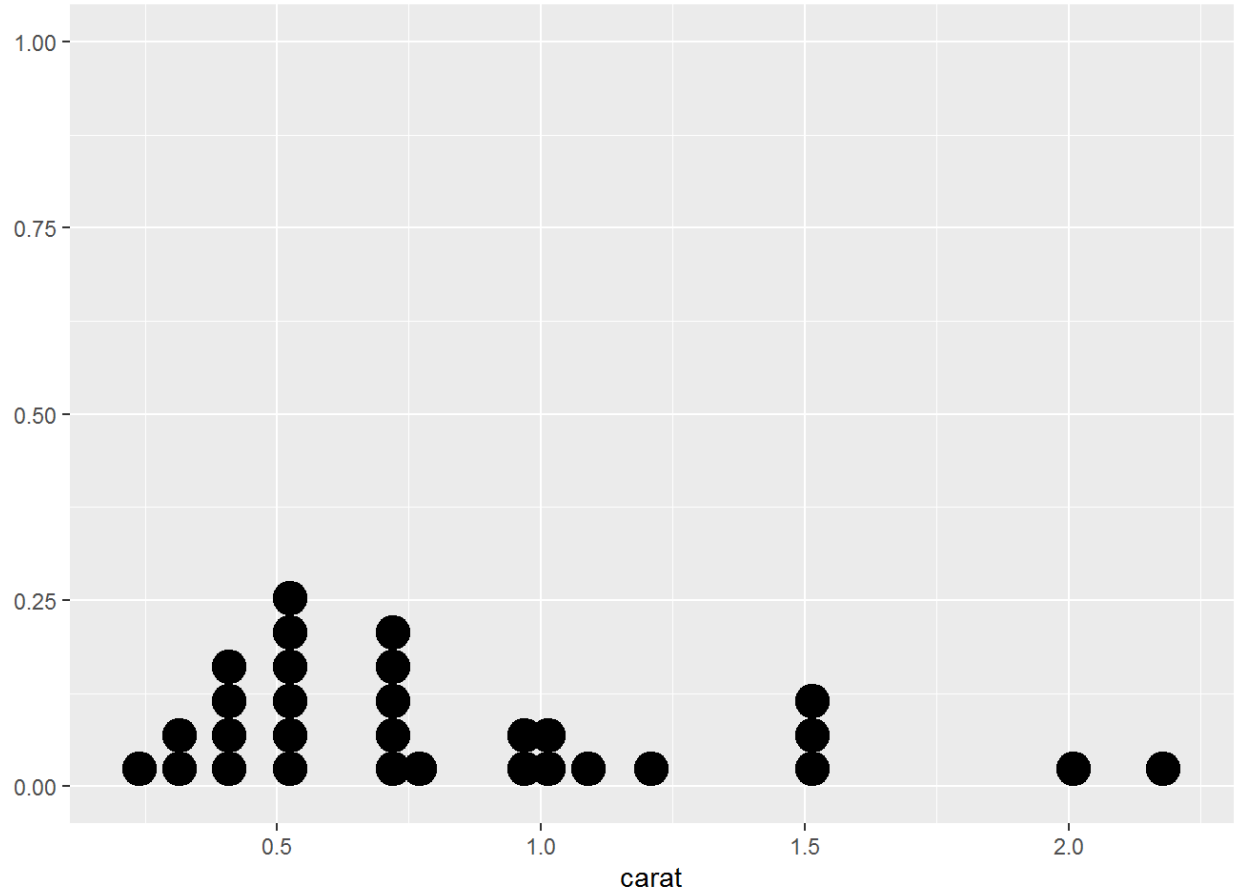
This ensures you can replicate the exact same random sample used in this example. Now to the dot plot. First install the `ggplot2` package.

```
library(ggplot2)
library(dplyr)

samp <- Diamonds %>% sample_n(30)

qplot(data = samp, x = carat, geom = "dotplot")
```

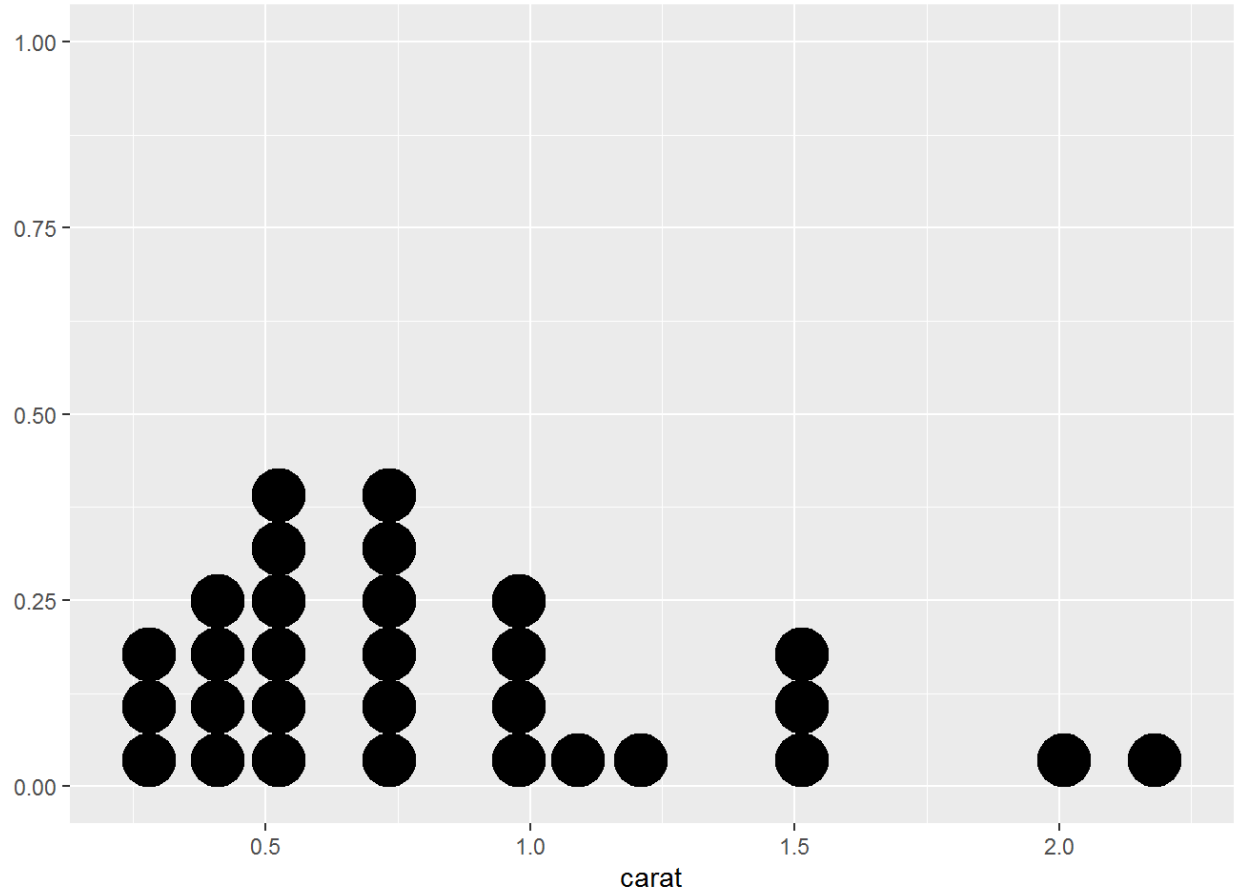
```
## `stat_bindot()` using `bins = 30`. Pick better value with `binwidth`.
```



Notice how we have used the `sample_n()` function to randomly sample 30 diamonds.

We can use the `binwidth` option to change the number stacks or breaks in the dot plot.

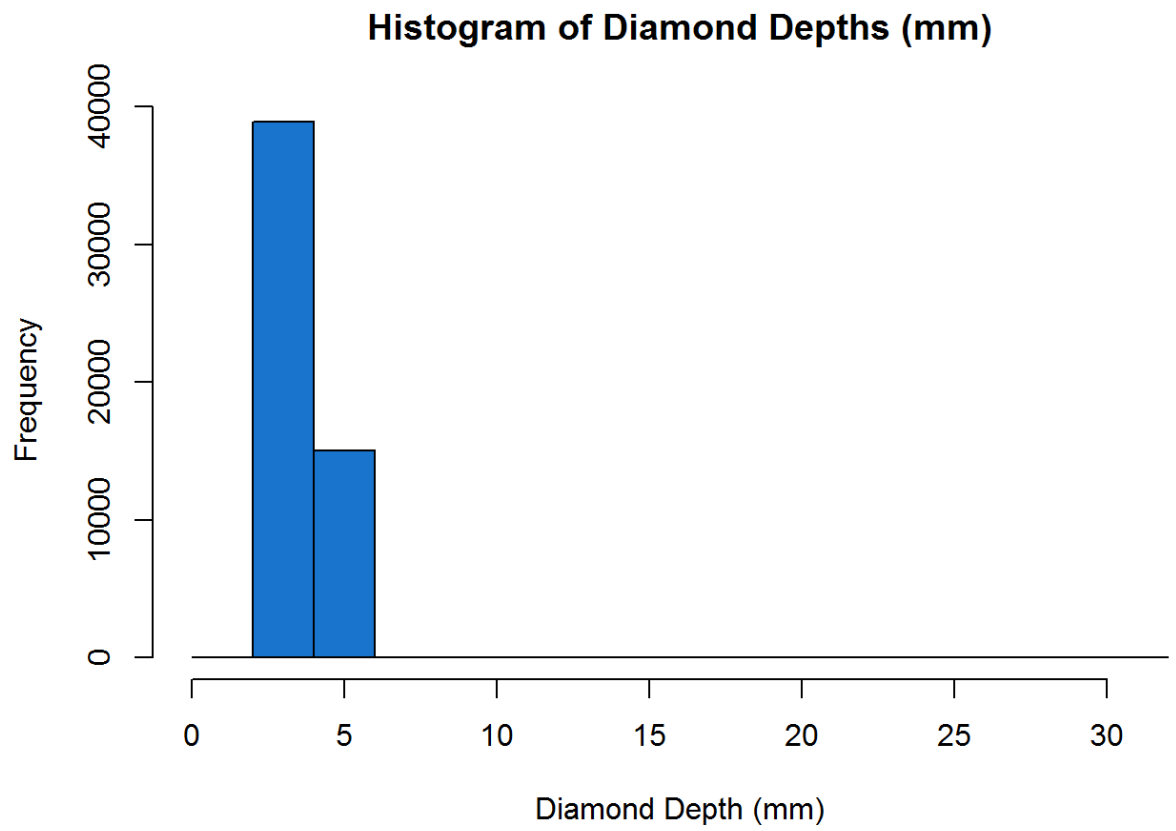
```
samp %>% qplot(data = ., x = carat, geom = "dotplot", binwidth =.1)
```



Histogram

Let's look at a histogram of diamond depths, `z`.

```
Diamonds$z %>% hist(xlab="Diamond Depth (mm)", main="Histogram of Diamond
  Depths (mm)",
                    col = "dodgerblue3")
```



Hmm, something is not right here. I think there are some extremely small and large values...

```
Diamonds$z %>% summary()
```

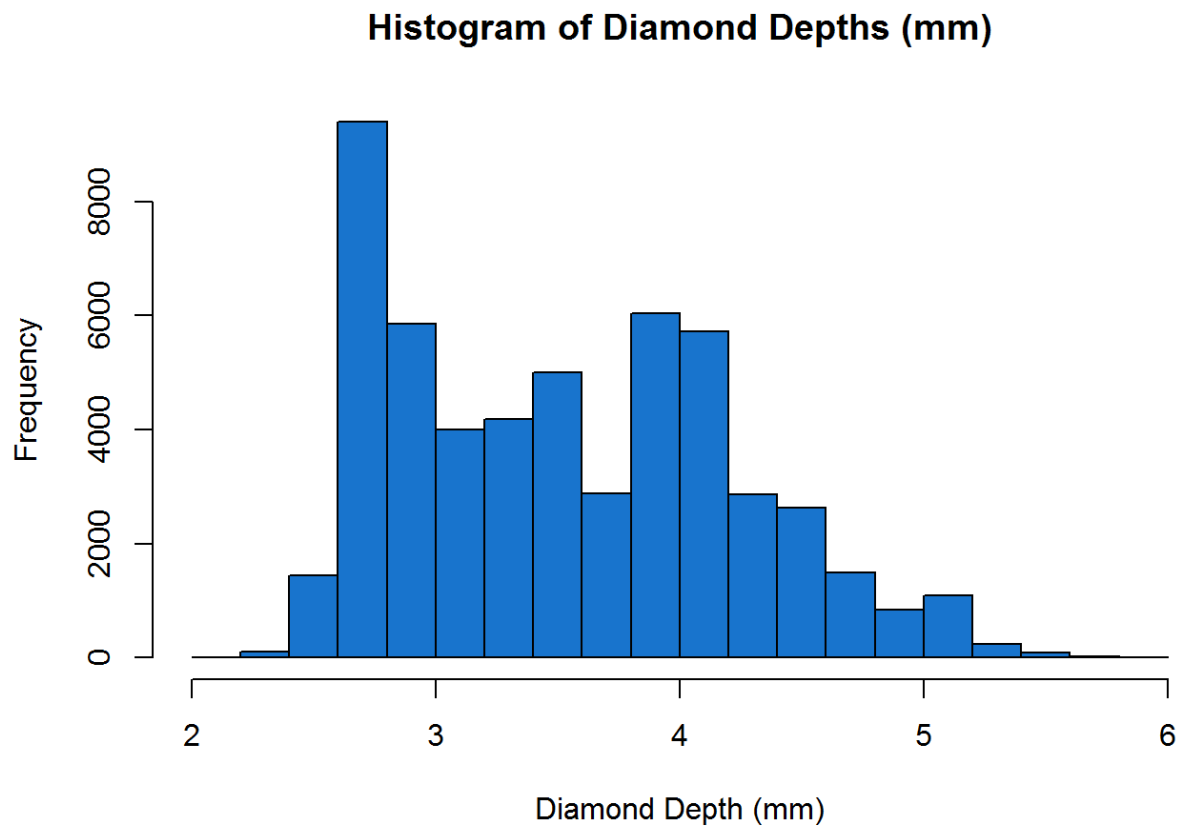
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   2.910   3.530   3.539   4.040  31.800
```

We have some diamonds with 0 depth (impossible and probably a data entry error) and a diamond with a depth of 31mm! Probably another error, as the carat measurement for this case is only .5. Let's focus our attention on diamonds between 2 and 6 mm using the `filter()` function. We will call the filtered data file object, `Diamonds_clean` .

```
Diamonds_clean <- Diamonds %>% filter(z < 6 & z > 2)
```

Now re-run the histogram on the filtered data:

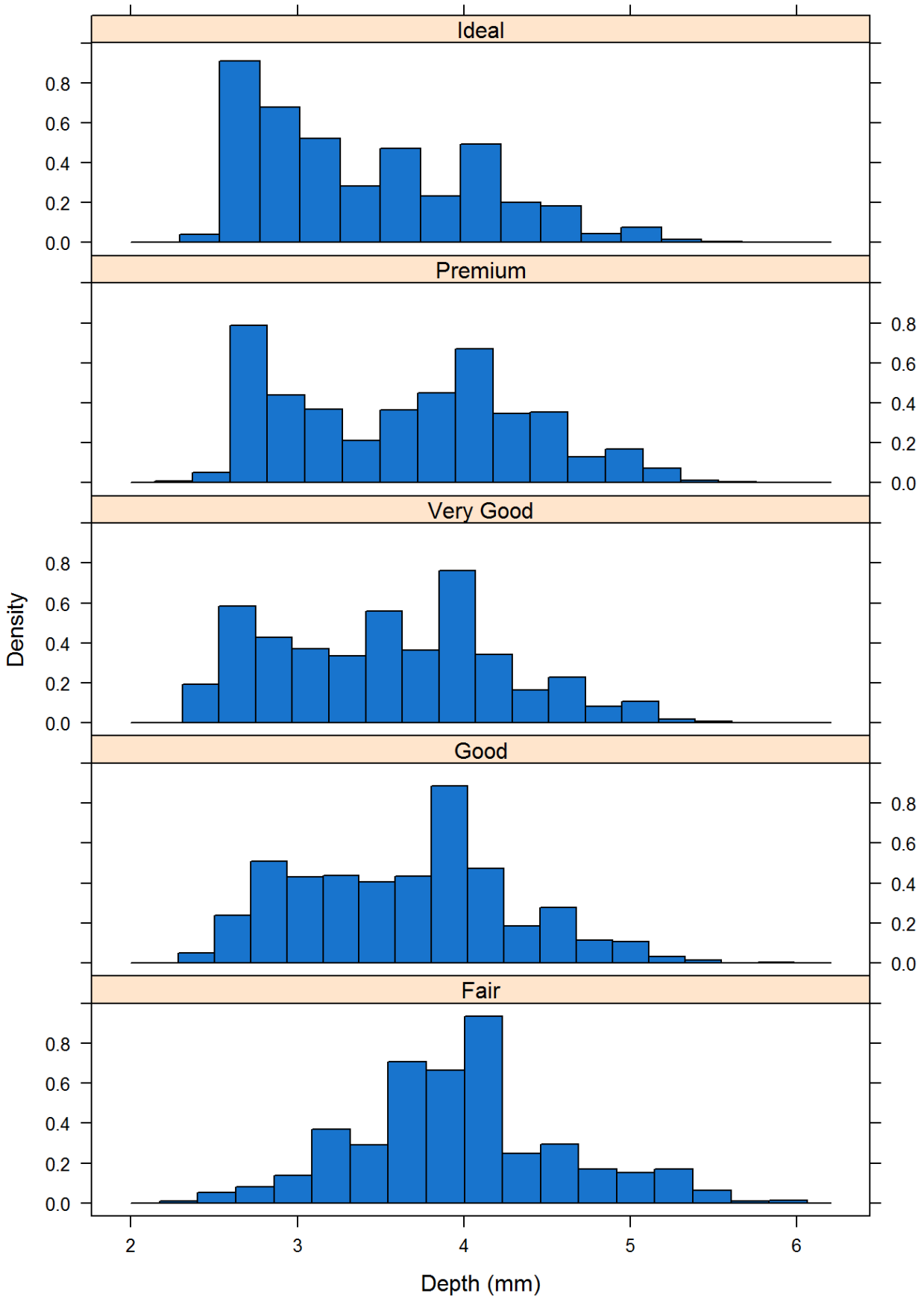
```
hist(Diamonds_clean$z,xlab="Diamond Depth (mm)",
     main="Histogram of Diamond Depths (mm)", col = "dodgerblue3")
```



Nice.

We can also panel histograms to help us compare a quantitative variable across categories, e.g. cut. We have to use a slightly different `histogram()` function from the `lattice` package:

```
library(lattice)
Diamonds_clean %>% histogram(~ z|cut,col="dodgerblue3",
                             layout=c(1,5), data=.,xlab="Depth (mm)")
```



Summary Statistics

R has many built-in functions and other functions available in various packages to quickly summarise your data. The following code will show you the basic functions. We will summarise the carat variable:

```
Diamonds$carat %>% mean() #Mean
```

```
## [1] 0.7979397
```

```
Diamonds$carat %>% range() #Min and Max
```

```
## [1] 0.20 5.01
```

```
5.01 - 0.20 #Calculate range
```

```
## [1] 4.81
```

```
Diamonds$carat %>% var()
```

```
## [1] 0.2246867
```

```
Diamonds$carat %>% sd()
```

```
## [1] 0.4740112
```

```
Diamonds$carat %>% quantile() #Quartiles
```

```
##   0%  25%  50%  75% 100%
## 0.20 0.40 0.70 1.04 5.01
```

```
Diamonds$carat %>% IQR() #Interquartile range
```

```
## [1] 0.64
```

```
Diamonds$carat %>% median() # Median
```

```
## [1] 0.7
```

The power of the `dplyr` package can be used to generate helpful descriptive statistics tables by groups. The logic is simple to remember. Select your dataset, group by a factor and summarise. You can select the descriptive statistics you need.

There are many other packages that make this easier, however, I believe learning to use `dplyr` to build these tables gives you far more flexibility and prevents the need to install another package.

Let's step through the process. We will summarise the diamonds' carat by their cut. The table will include min, Q_1, median, Q_3, max, mean, standard deviation, n and missing value count. I've included quite a few descriptive statistics so you can get a sense of how to customise the table. Essentially, the table can incorporate any descriptive functions in R provided the function results in a single value. Take note how we have to include the option `na.rm = TRUE` to most of these functions.

```
Diamonds %>% group_by(cut) %>% summarise(Min = min(carat,na.rm = TRUE),
                                         Q1 = quantile(carat,probs = .25,
  na.rm = TRUE),
                                         Median = median(carat, na.rm = T
  RUE),
                                         Q3 = quantile(carat,probs = .75,
  na.rm = TRUE),
                                         Max = max(carat,na.rm = TRUE),
                                         Mean = mean(carat, na.rm = TRUE
  ),
                                         SD = sd(carat, na.rm = TRUE),
                                         n = n(),
                                         Missing = sum(is.na(carat)))
```

```
## # A tibble: 5 x 10
##   cut         Min    Q1 Median    Q3   Max  Mean    SD     n Missing
##   <ord>     <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <int>   <int>
## 1 Fair       0.22  0.7    1     1.2   5.01 1.05  0.516  1610       0
## 2 Good       0.23  0.5    0.82  1.01  3.01 0.849 0.454  4906       0
## 3 Very Good  0.2   0.41   0.71  1.02  4    0.806 0.459 12082       0
## 4 Premium    0.2   0.41   0.86  1.2   4.01 0.892 0.515 13791       0
## 5 Ideal      0.2   0.35   0.54  1.01  3.5  0.703 0.433 21551       0
```

Now it is easy to factor by another grouping variable.

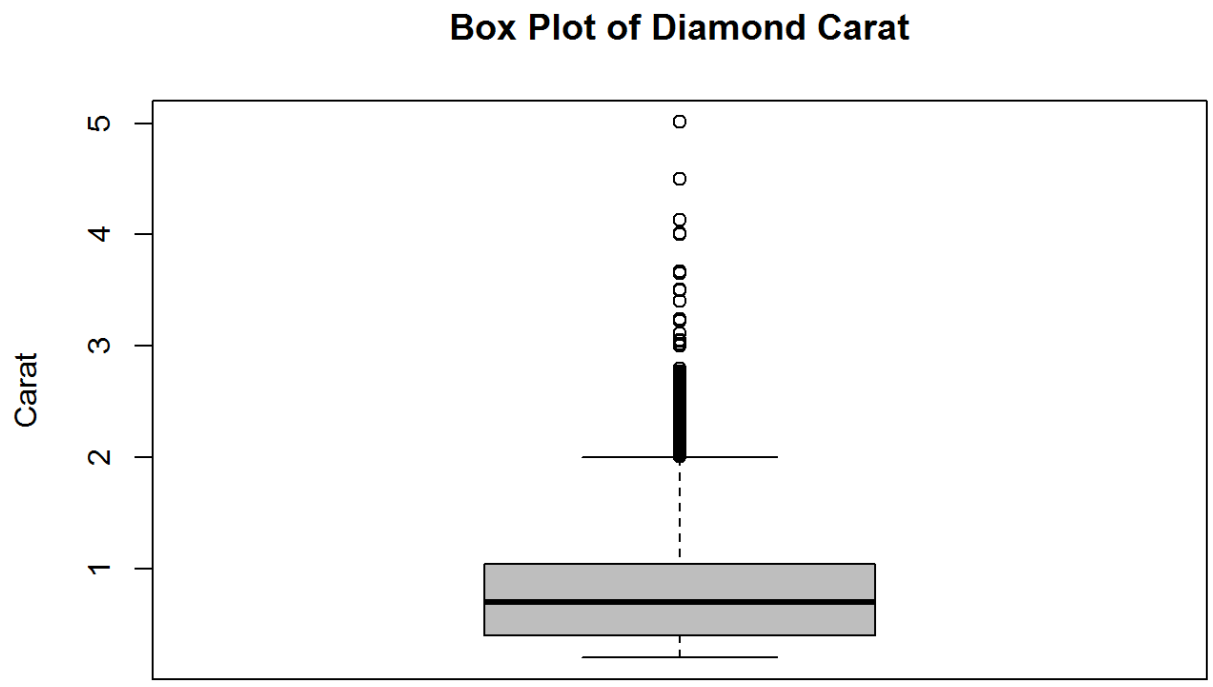
```
Diamonds %>% group_by(cut, color) %>% summarise(Min = min(carat,na.rm = T
  RUE),
                                                 Q1 = quantile(carat,probs = .25,
  na.rm = TRUE),
                                                 Median = median(carat, na.rm = T
  RUE),
                                                 Q3 = quantile(carat,probs = .75,
  na.rm = TRUE),
                                                 Max = max(carat,na.rm = TRUE),
                                                 Mean = mean(carat, na.rm = TRUE
  ),
                                                 SD = sd(carat, na.rm = TRUE),
                                                 n = n(),
                                                 Missing = sum(is.na(carat)))
```

```
## # A tibble: 35 x 11
## # Groups:   cut [5]
##    cut   color   Min    Q1 Median    Q3   Max  Mean    SD     n Missin
g
##    <ord> <ord> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <int>   <int
>
##  1 Fair  J      0.3  0.905   1.03  1.68  5.01 1.34  0.734   119
0
##  2 Fair  I      0.41 0.885   1.01  1.50  3.02 1.20  0.522   175
0
##  3 Fair  H      0.33 0.9     1.01  1.51  4.13 1.22  0.548   303
0
##  4 Fair  G      0.23 0.7     0.98  1.07  2.6  1.02  0.493   314
0
##  5 Fair  F      0.25 0.6     0.9   1.01  2.58 0.905 0.419   312
0
##  6 Fair  E      0.22 0.552   0.9   1.01  2.04 0.857 0.365   224
0
##  7 Fair  D      0.25 0.7     0.9   1.01  3.4  0.920 0.405   163
0
##  8 Good  J      0.28 0.71    1.02  1.5   3    1.10  0.537   307
0
##  9 Good  I      0.3  0.7     1     1.5   3.01 1.06  0.576   522
0
## 10 Good  H      0.25 0.51    0.9   1.09  3.01 0.915 0.498   702
0
## # ... with 25 more rows
```

Box plots

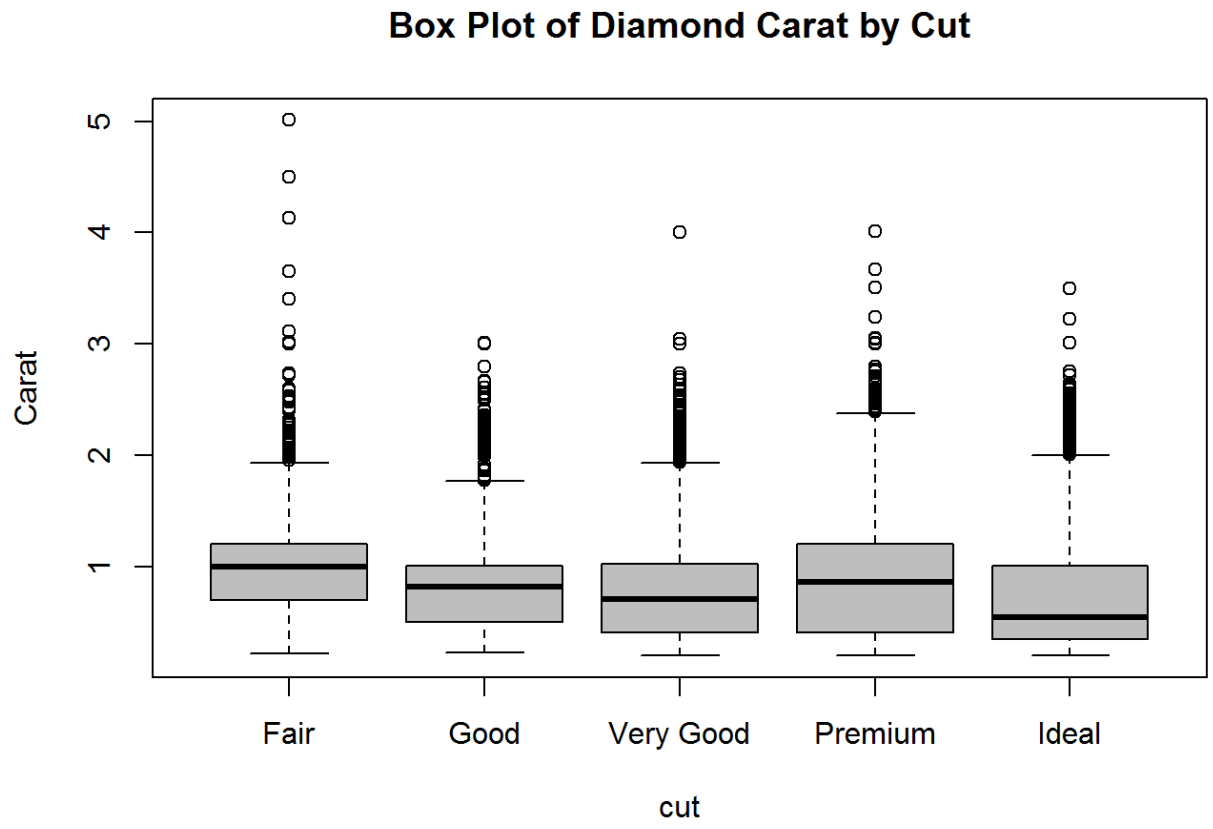
Let's get a box plot of the carat variable using R:

```
Diamonds$carat %>%  boxplot(main="Box Plot of Diamond Carat", ylab="Cara
  t", col = "grey")
```



We use a formula, `carat ~ cut`, and add `data = .` to pipe in the dataset, in order to produce a side-by-side box plot:

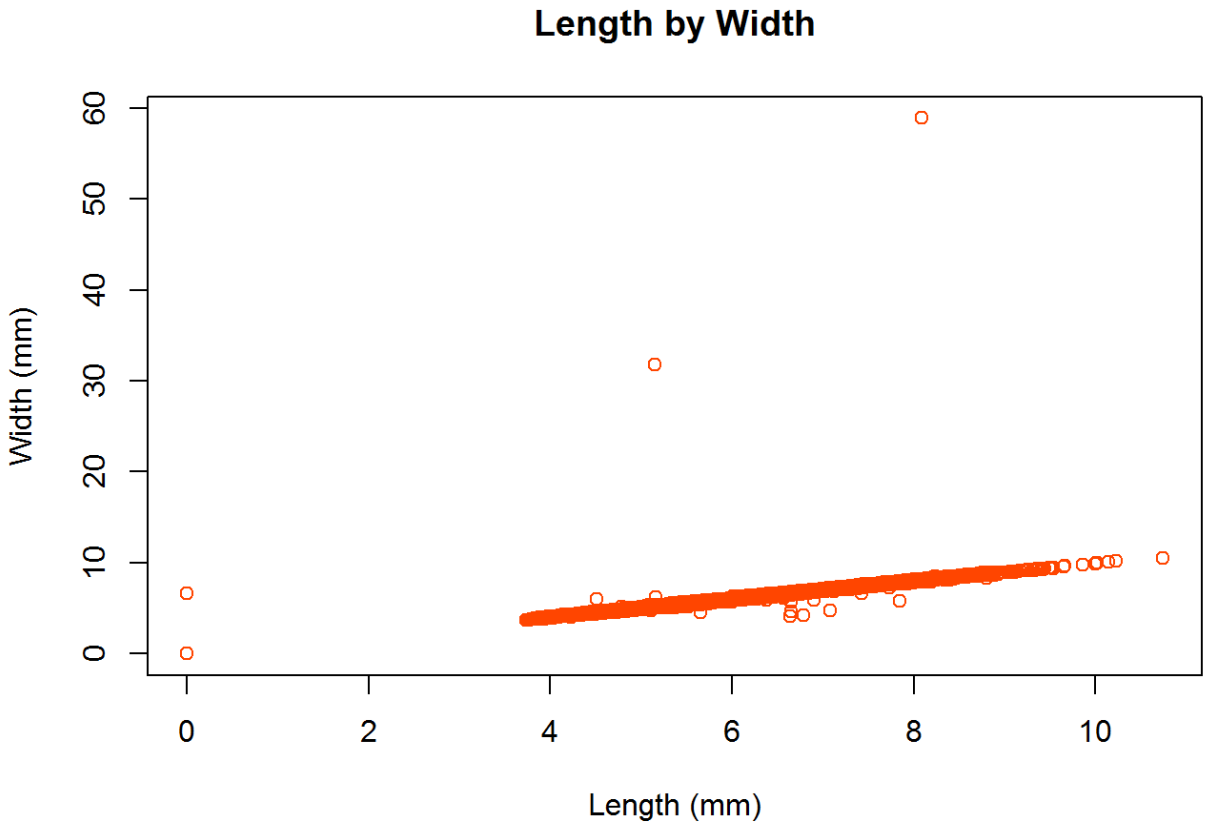
```
Diamonds %>% boxplot(carat ~ cut, data = ., main="Box Plot of Diamond Car
  at by Cut",
        ylab = "Carat", xlab = "cut", col="grey")
```



Scatter plots

Scatter plots are produced using the base `plot()` function. For example, let's get a scatter plot of diamond width (`y`) length (`x`) measured in mm:

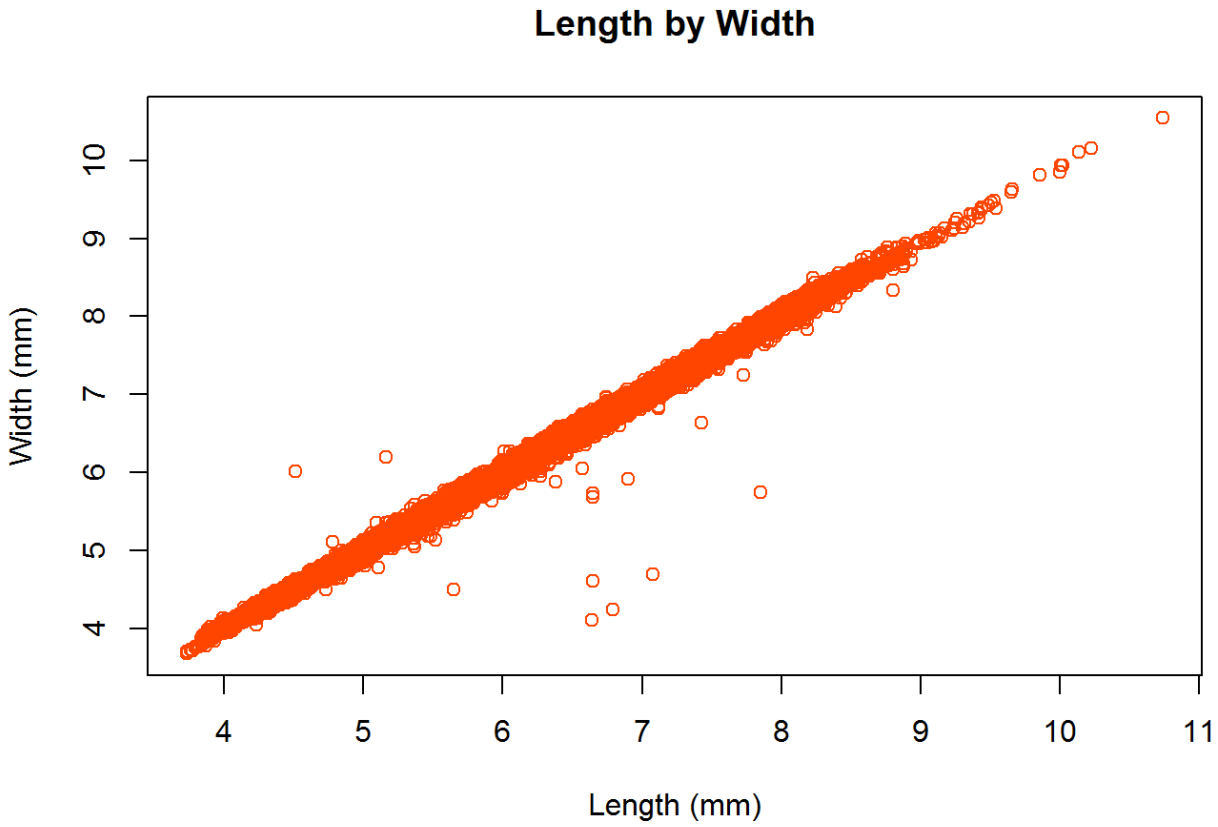
```
plot(y ~ x, data = Diamonds, ylab="Width (mm)", xlab="Length (mm)", col=
  "orangered",
     main="Length by Width")
```



Hmm, we have outliers at Length = 0 mm and widths > 20 mm that are distorting the main trend. Let's filter them out and re-do the scatter:

```
Diamonds_clean <- Diamonds %>% filter(y < 20 & x > 0)

plot(y ~ x, data = Diamonds_clean, ylab="Width (mm)", xlab="Length (mm)",
     col="orangered",main="Length by Width")
```



Now the trend is more apparent. We shouldn't forget to investigate the outliers.

References