PRACTICUM, day 1: R graphing: basic plotting and ggplot2

CRG Bioinformatics Unit, sarah.bonnin@crg.eu

May 6th, 2016

Contents

Introduction

In this tutorial, you will learn how to use some R functions and parameters to represent your data. We will mainly focus on producing and customizing graphs using the ggplot2 R package.

Packages

First, let's install and load the packages we will need:

```
install.packages(c("ggplot2","RColorBrewer", "gridExtra", "gplots", "VennDiagram"))
```

If R does not find automatically the packages, you can specify a package repository:

```
install.packages(c("ggplot2", "RColorBrewer", "gridExtra", "gplots", "VennDiagram"),
  repos="http://cran.r-project.org/web/packages/")
```

Once the packages are installed, you can load them in your current R session:

```
library("ggplot2")
library("RColorBrewer")
library("gridExtra")
library("gplots")
library("VennDiagram")
```

Dataset

We will use the **diamonds** dataset from the ggplot2 package (added automatically into the global environment when ggplot2 is loaded). This dataset contains the prices and other attributes of almost 54,000 diamonds.

It is interesting for us because it contains plenty of discrete and continuous data that we will be able to use:

```
head(diamonds)
```

```
##   carat       cut color clarity depth table price    x    y    z
## 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 4.20 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
```

```
# for more details:
?diamonds
```

Basic plotting

Basic plotting functions in R include (but are not limited to):

- plot (for scatter plots)
- hist (for histograms)
- boxplot (for boxplots)
- lines (for line charts)
- pie (for pie charts)

Parameters can be added to each of these functions to control sizes, colors, labels, lines, and more.

We will see a few examples of scatter plots and box plots using the basic R functions.

Scatter plots: plot()

> A scatter plot is a type of plot or mathematical diagram using Cartesian coordinates to display values for typically two variables for a set of data. (Wikipedia)

The **plot** function takes x and y coordinates as base parameters:

```
plot(x, y)
```

Here, we will plot **price** versus **carat** attributes of diamonds.

We will subset the dataset to manipulate more easily the data (and to practice subscripting!):

```
# Create a 2 column data frame
diamonds1 <- diamonds[,c("carat","price")]
head(diamonds1)
```

```
##   carat price
## 1  0.23   326
## 2  0.21   326
## 3  0.23   327
## 4  0.29   334
## 5  0.31   335
## 6  0.24   336
```

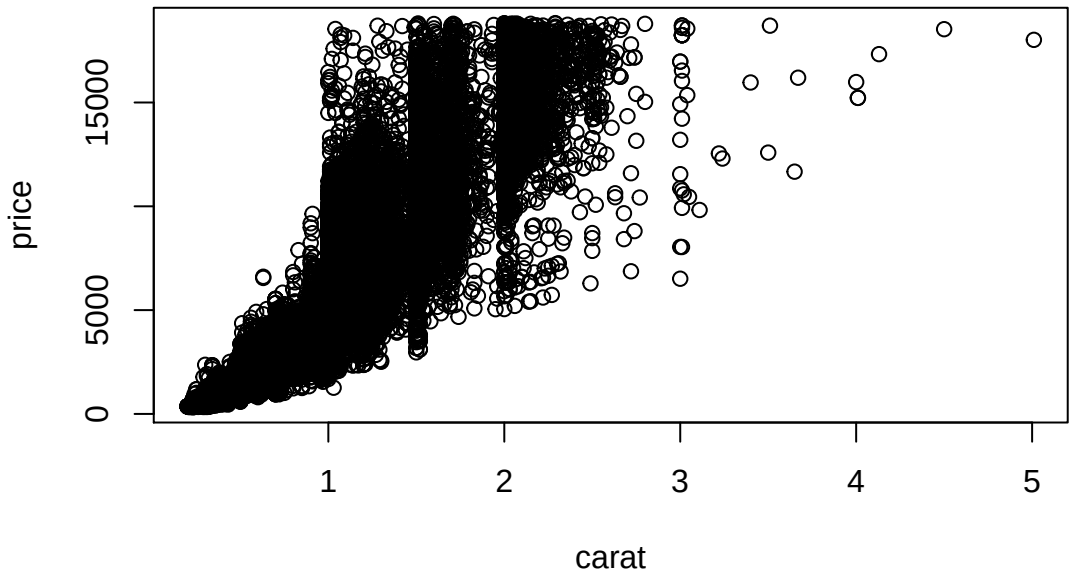
If a data frame is given as an input, plot will use all columns and create as many pairwise scatter plots as possible.

So command:

```
plot(diamonds1$price, diamonds1$carat)
```

will produce the same plot as:

```
plot(diamonds1)
```

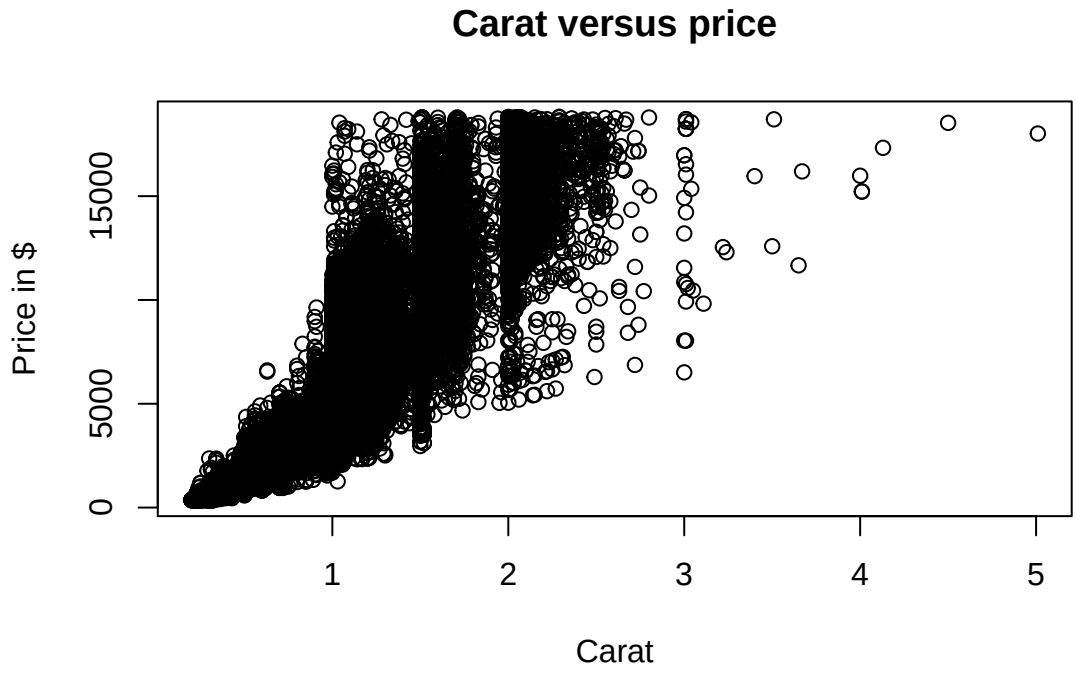


Each point in these scatter plots represents one diamond.

We can:

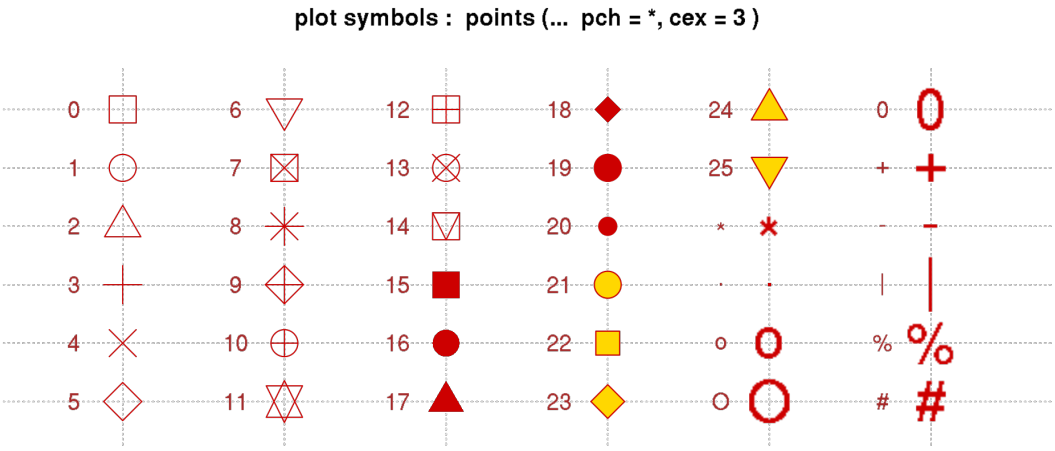
- Add a title
- Change x and y axis labels

```
# main: plot title
# xlab and ylab: x and y axis labels, respectively
# cex: point size
plot(diamonds1,
     main="Carat versus price",
     xlab="Carat",
     ylab="Price in $")
```



Parameters **col**, **cex** and **pch** can be used to modify point color, point size and point shape, respectively.

POINT TYPES



BASIC COLORS

A vector of 657 colors is available by default:

```
colors()
```

You can for example take the first 10 values from that vector:

```
colors()[1:10]
```

```
##  [1] "white"         "aliceblue"     "antiquewhite"  "antiquewhite1"
##  [5] "antiquewhite2" "antiquewhite3" "antiquewhite4" "aquamarine"
##  [9] "aquamarine1"   "aquamarine2"
```

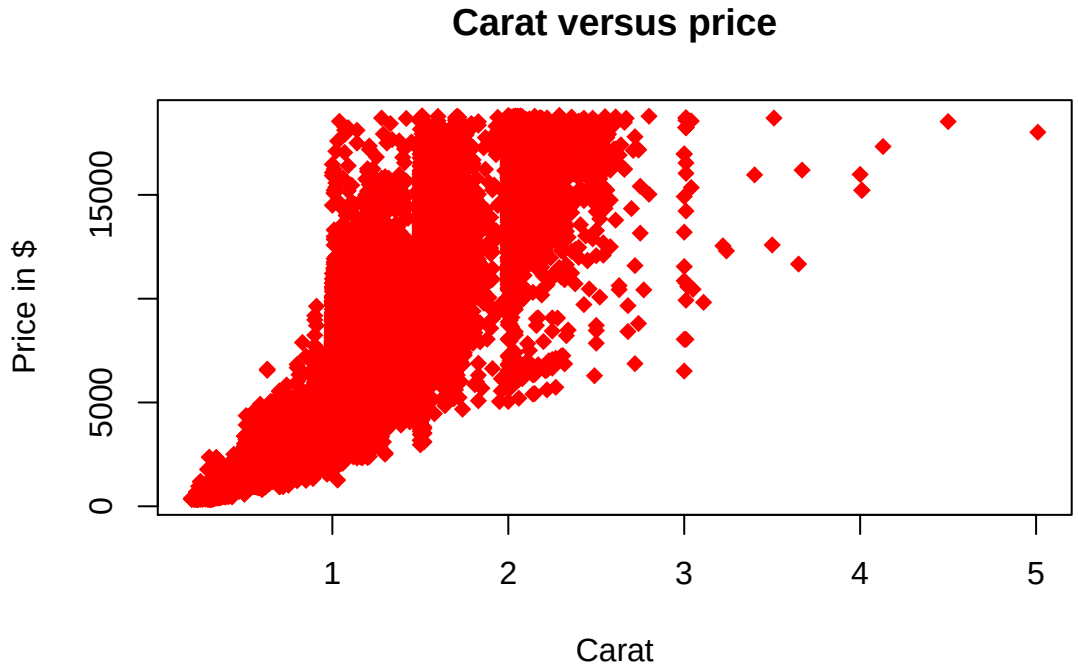
Or select 10 colors randomly using the **sample** function:

```
?sample
# sample(x, size=n): randomly select n elements from x
sample(colors(), 10)
```

```
##  [1] "gray86"        "grey74"        "coral2"        "darkorange2"
##  [5] "lightskyblue2" "coral3"        "mistyrose3"    "brown4"
##  [9] "linen"         "orange"
```

Exercise 1

Review the previous scatter plot to obtain the following one:



Using the following parameters:

- pch
- col
- cex

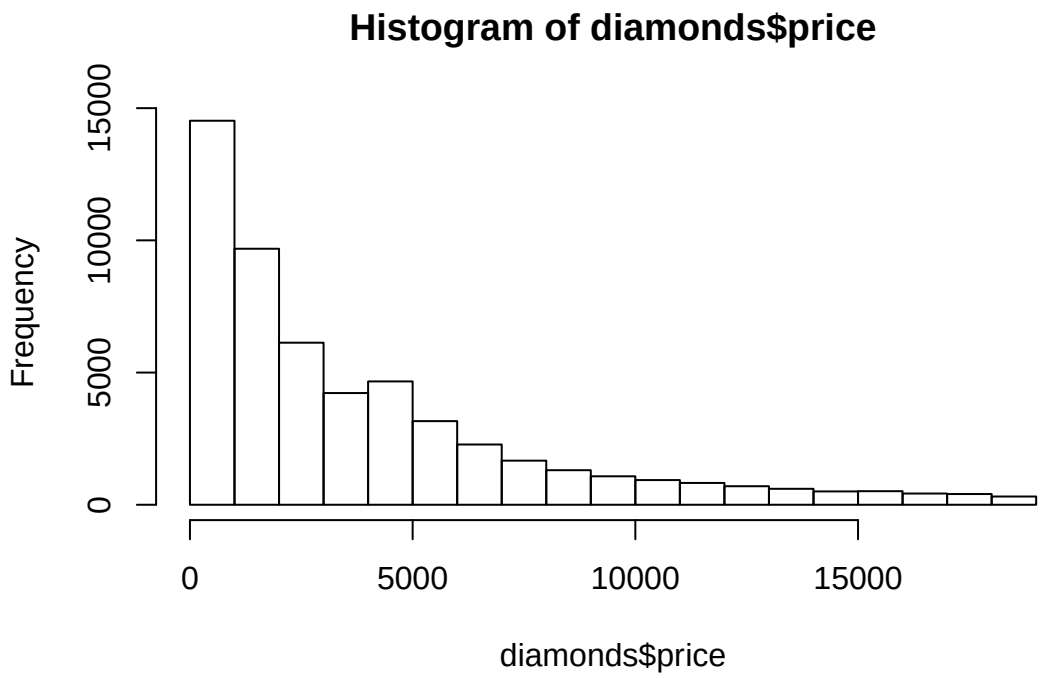
Histogram: hist()

> A histogram is a plot that lets you discover, and show, the underlying frequency distribution of a set of continuous data. (Wikipedia)

Function **hist** takes a numeric vector as a minimum input.

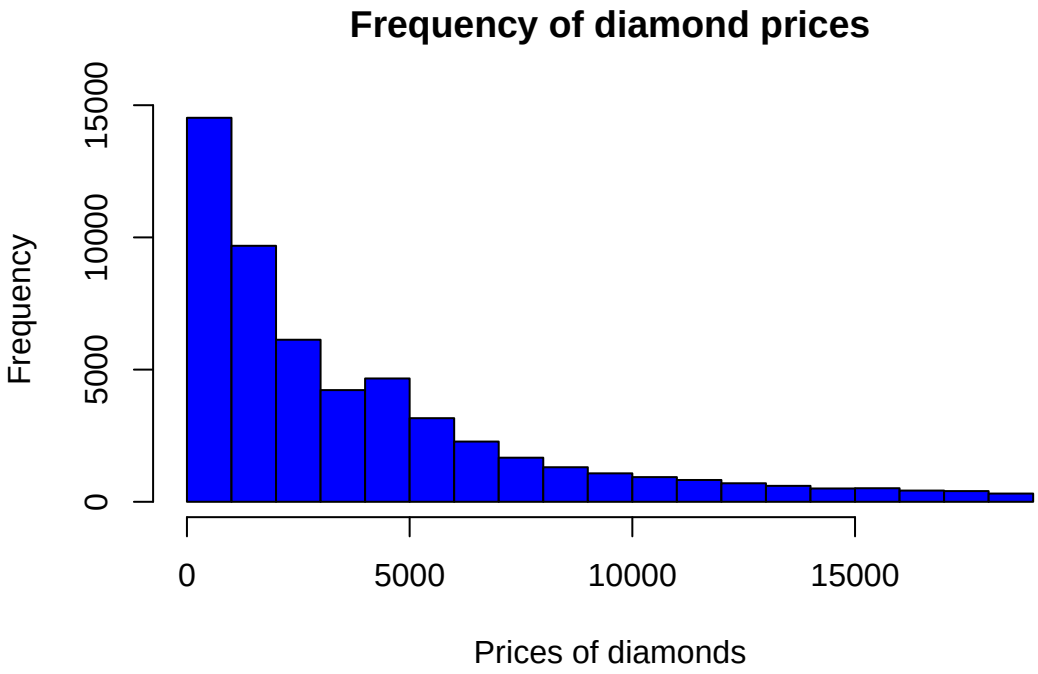
We will use this function to plot the distribution of diamonds prices in this dataset.

```
hist(diamonds$price)
```



We can change bar color, x and y labels, and add a title to the graph the same way as with the plot function.

```
hist(diamonds$price, col="blue",
     xlab="Prices of diamonds",
     ylab="Frequency",
     main="Frequency of diamond prices")
```



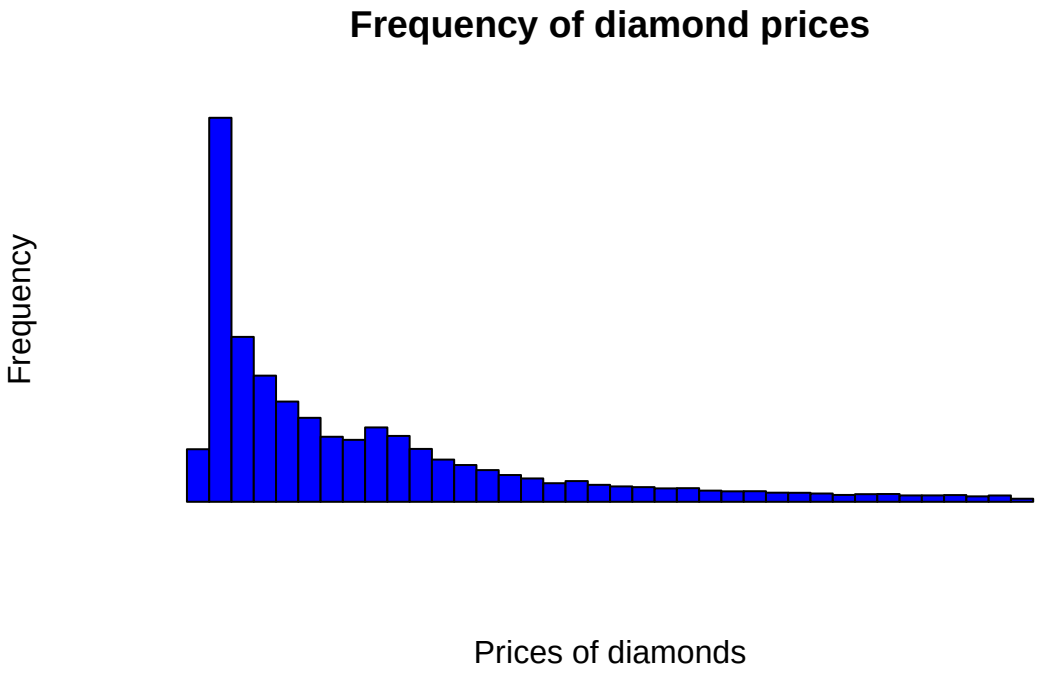
Bins represent the number of bars into which the histogram is divided.

The higher the number of bins, the higher the histogram's resolution.

We can change more parameters, to:

- Control the number of bins
- Remove the axes

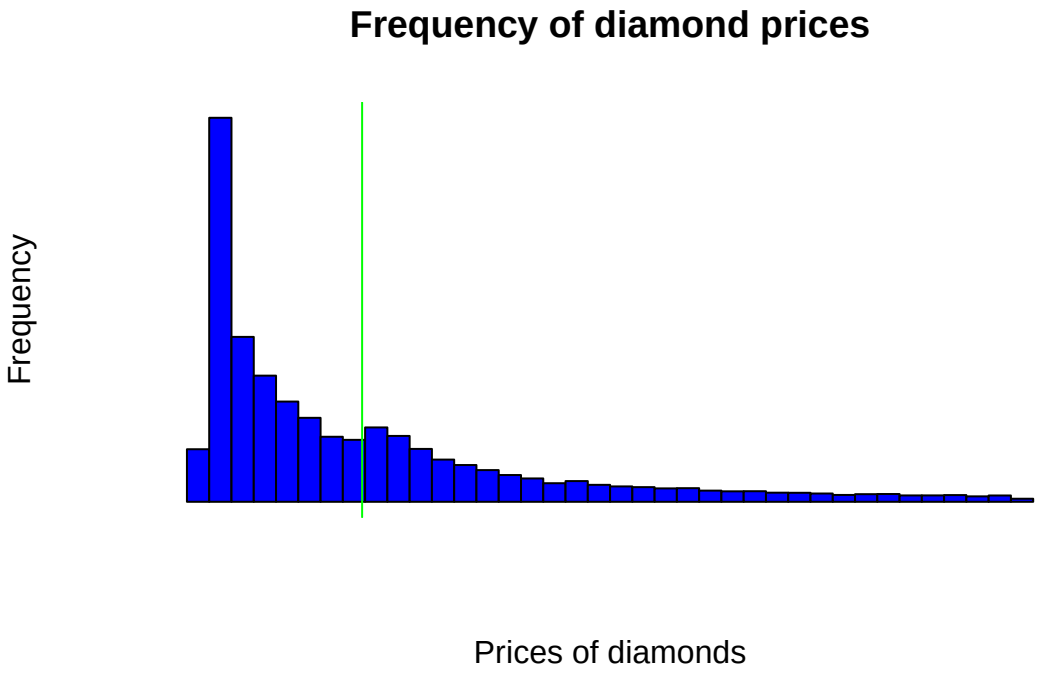
```
# breaks: control the number of bins used to draw the histogram
# axes: logical, whether axes should be drawn or not
hist(diamonds$price, col="blue",
     xlab="Prices of diamonds",
     ylab="Frequency",
     main="Frequency of diamond prices",
     breaks=30,
     axes=FALSE)
```



We can add vertical or horizontal lines: Here we want to display a **green vertical line** to show the average price.

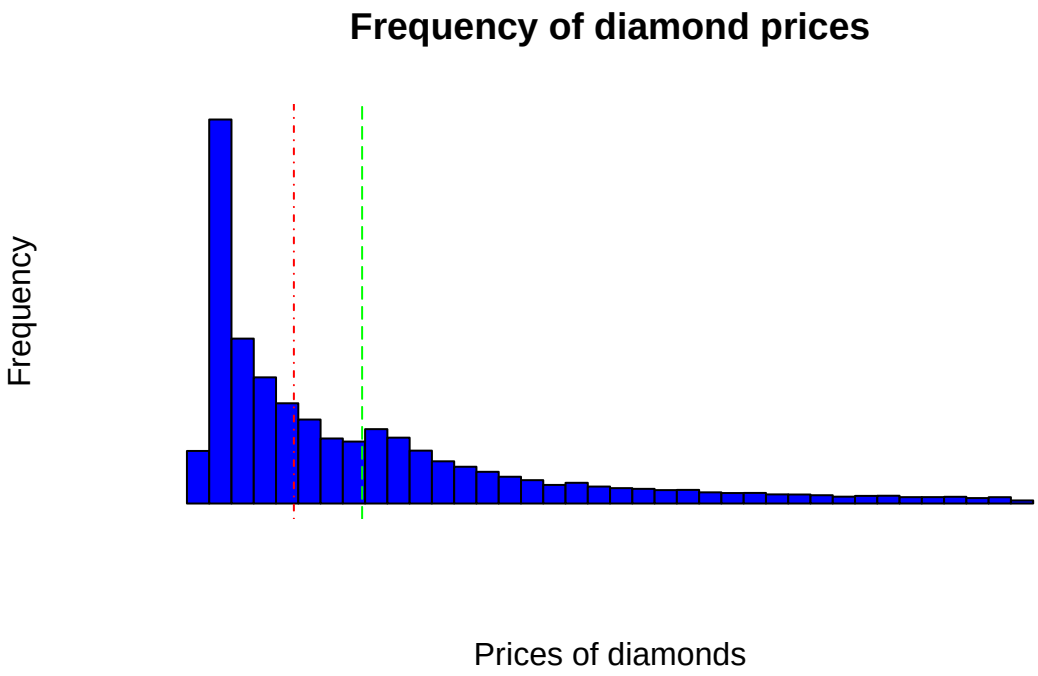
abline function is called to add the line, **additionally to the hist function and not as a parameter**

```
hist(diamonds$price, col="blue",
     xlab="Prices of diamonds",
     ylab="Frequency",
     main="Frequency of diamond prices",
     breaks=30,
     axes=FALSE)
# abline: "v" specifies the x-value(s) for vertical line(s).
abline(v=mean(diamonds$price), col="green")
```


Frequency of diamond prices
Frequency
Prices of diamonds

Exercise 2

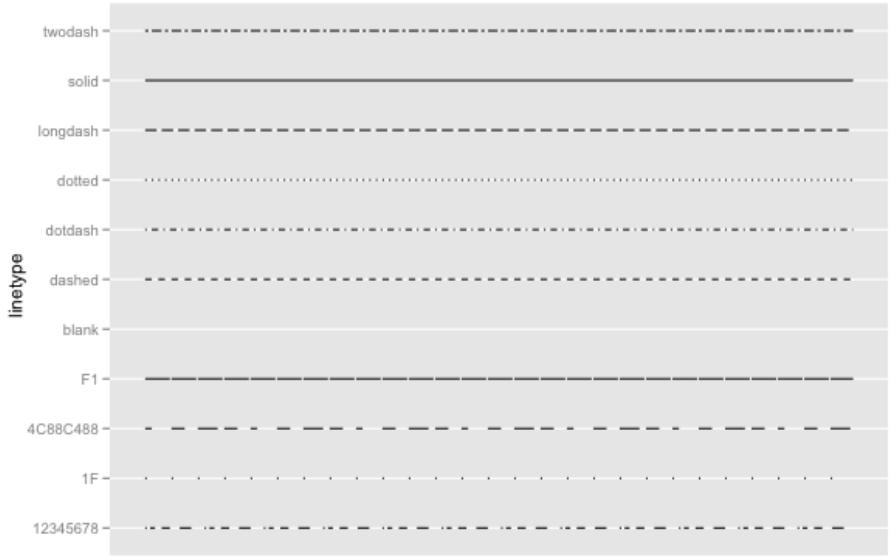
Produce the following histogram:



You will need:

- abline function: to add a vertical line that represents the median.
- lty parameter: to control the type of the lines.

LINE TYPES in R



Plotting with ggplot2

> Graphing package inspired by the Grammar of Graphics seminal work of Leland Wilkinson. A tool that enables us to concisely describe the components of a graphic.

Why ggplot2?

- More flexible
- More customizable
- Prettier
- Easy to modify
- Well documented.

Getting started

What you need to start any ggplot are:

- a data set.
- a coordinate system.
- a set of **geoms**: visual marks that represent the data point.

Once the plot is started, you can add **layers** and additional elements to customize the plot.

Layers elements are added with +.

Starting a plot: "base" layer:

```
# One variable
a <- ggplot(dataframe, aes(x))
# Two variables
a <- ggplot(dataframe, aes(x, y))
```

aes function generate aesthetic mappings that describe how variables in the data are mapped to visual properties (aesthetics) of geoms.

A few simple examples of the possible **geoms** that can be mapped:

```
# One continuous variable:
a + geom_density()
a + geom_dotplot()
a + geom_histogram()
# One discrete variable:
a + geom_bar()
# Two variables: Continuous X, Continuous Y:
a + geom_point()
a + geom_smooth()
# Two variables: Discrete X, Continuous Y:
a + geom_bar()
a + geom_boxplot()
```

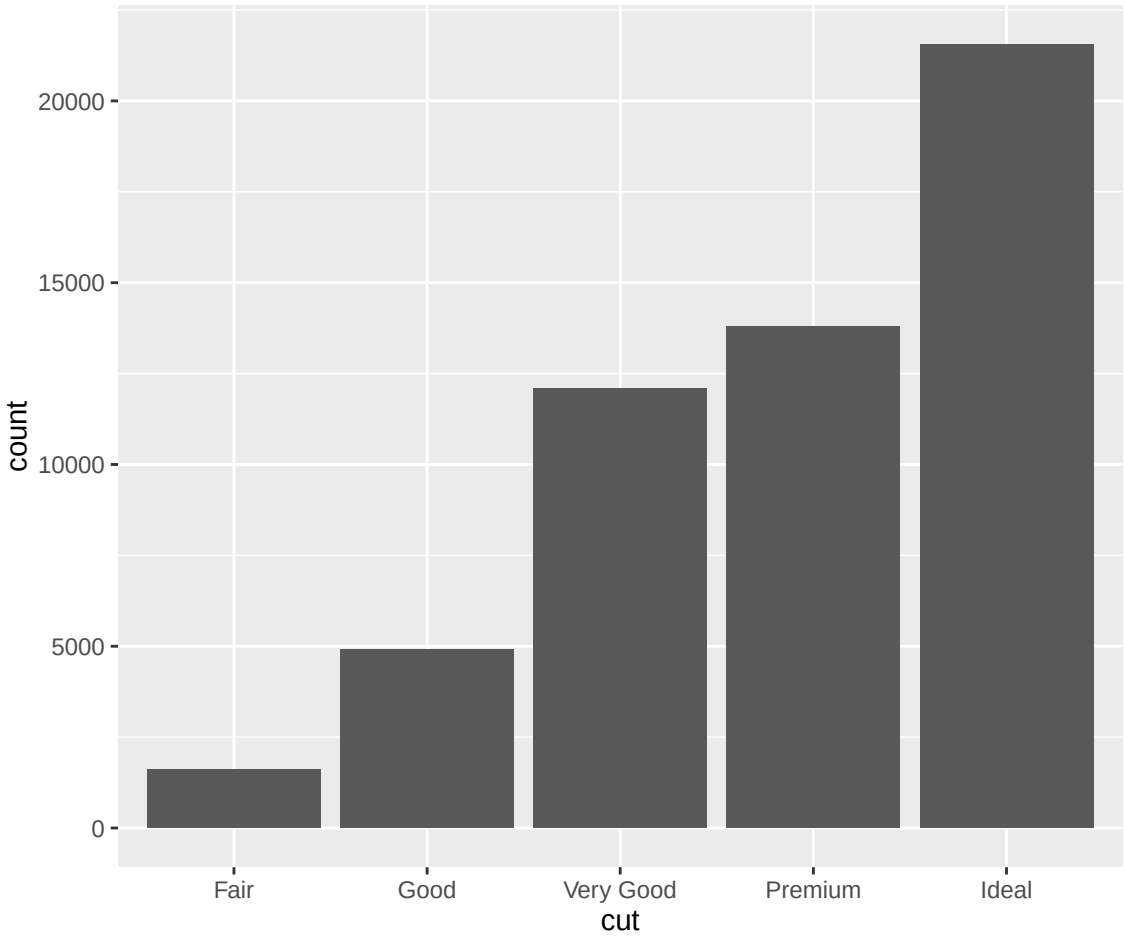
A good cheatsheet to get started.

Barplots: geom_bar()

> A bar chart or bar graph is a chart that presents grouped data with rectangular bars with lengths proportional to the values that they represent. (Wikipedia)

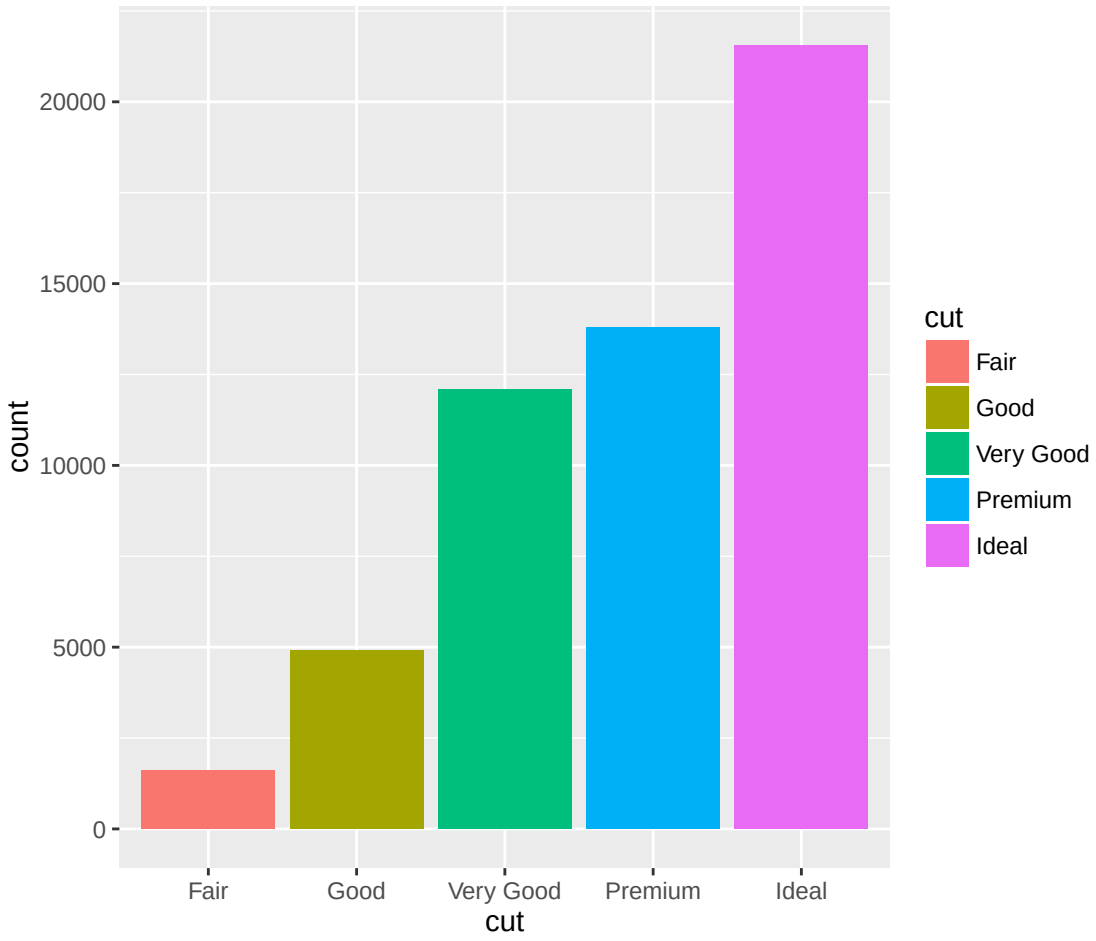
We will start with the most basic barplot using the **geom_bar()** geom, representing the different diamond cuts:

```
barp0 <- ggplot(data=diamonds, aes(x=cut)) +
    geom_bar()
barp0
```



We can color each bar per category (type of cuts):

```
barp1 <- ggplot(data=diamonds, aes(x=cut, fill=cut)) +
    geom_bar()
barp1
```



We might want to change the default color scheme. There are plenty of color schemes in R.

Palettes available by default

The following functions are pre-made palettes that create vectors of **n** different colors:

```
rainbow(n)
heat.colors(n)
terrain.colors(n)
topo.colors(n)
cm.colors(n)
```

For example:

```
rainbow(n=10)
```

```
##  [1] "#FF0000FF" "#FF9900FF" "#CCFF00FF" "#33FF00FF" "#00FF66FF"
##  [6] "#00FFFFFF" "#0066FFFF" "#3300FFFF" "#CC00FFFF" "#FF0099FF"
```

Additional palettes

One example is the package **RColorBrewer** that provides many color schemes for graphics:

```r
# Check what the main function offers
?brewer.pal
```

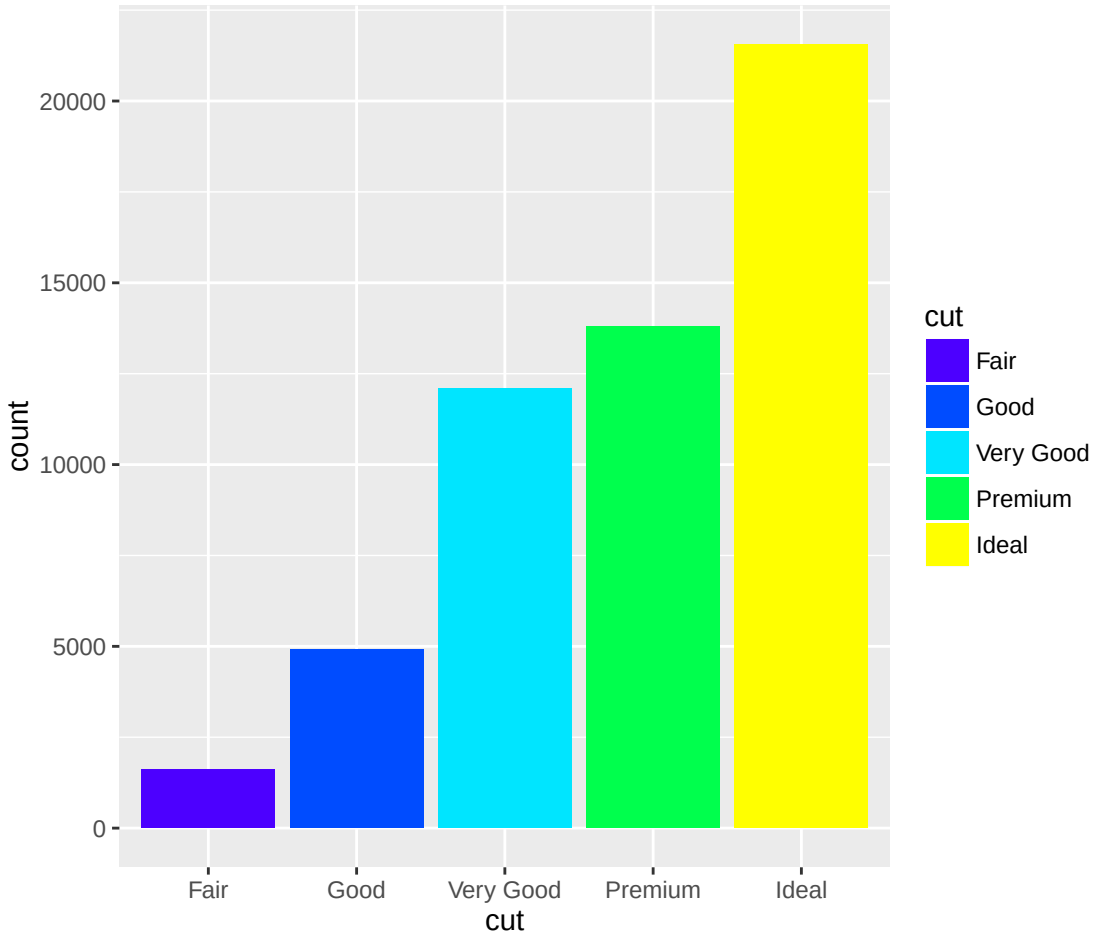
One example on how to use the **brewer.pal** function:

```r
# Getting 8 colors from the "Oranges" scheme
brewer.pal(n=8, name="Oranges")
```

```
## [1] "#FFF5EB" "#FEE6CE" "#FDD0A2" "#FDAE6B" "#FD8D3C" "#F16913" "#D94801"
## [8] "#8C2D04"
```

Exercise 3

Modify barp1 to obtain the following plot (barp2)

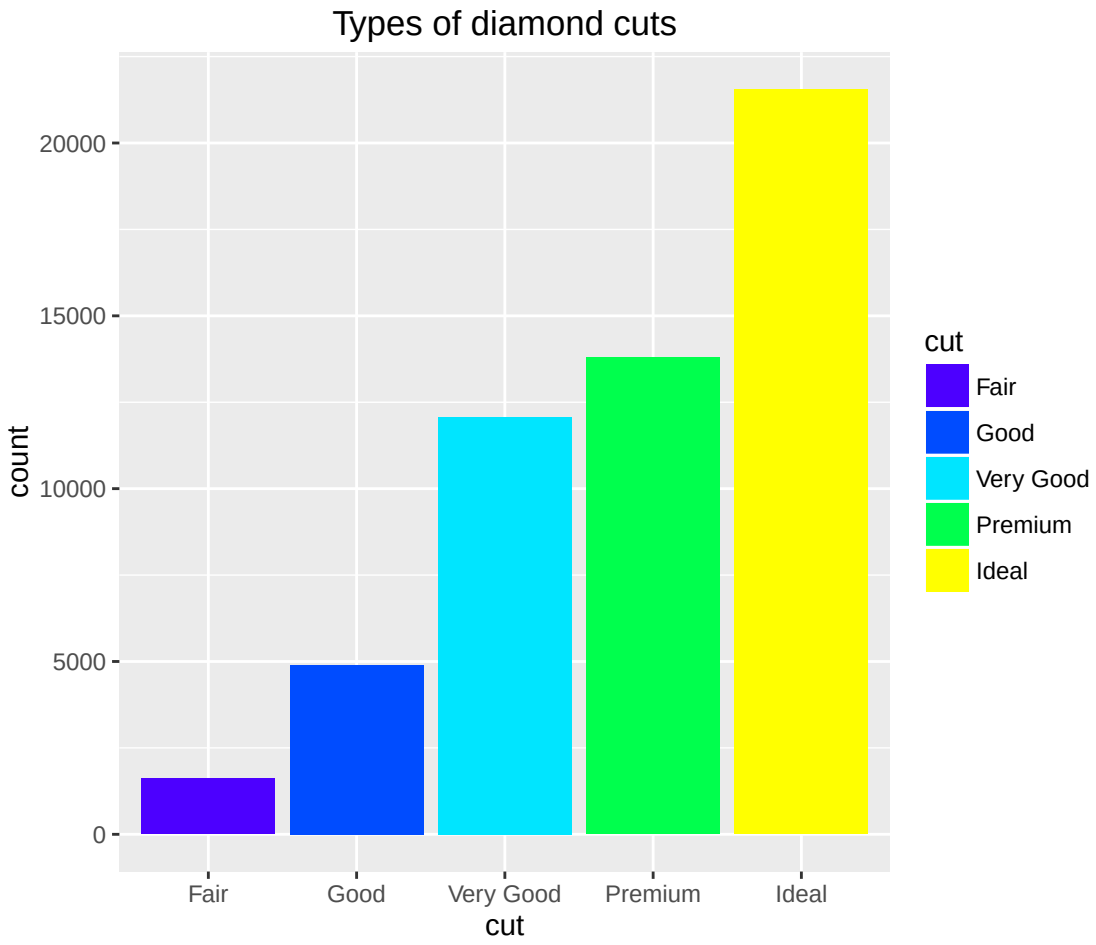


Using:

- scale_fill_manual layer
- top.colors palette

We can next add a title to the plot:

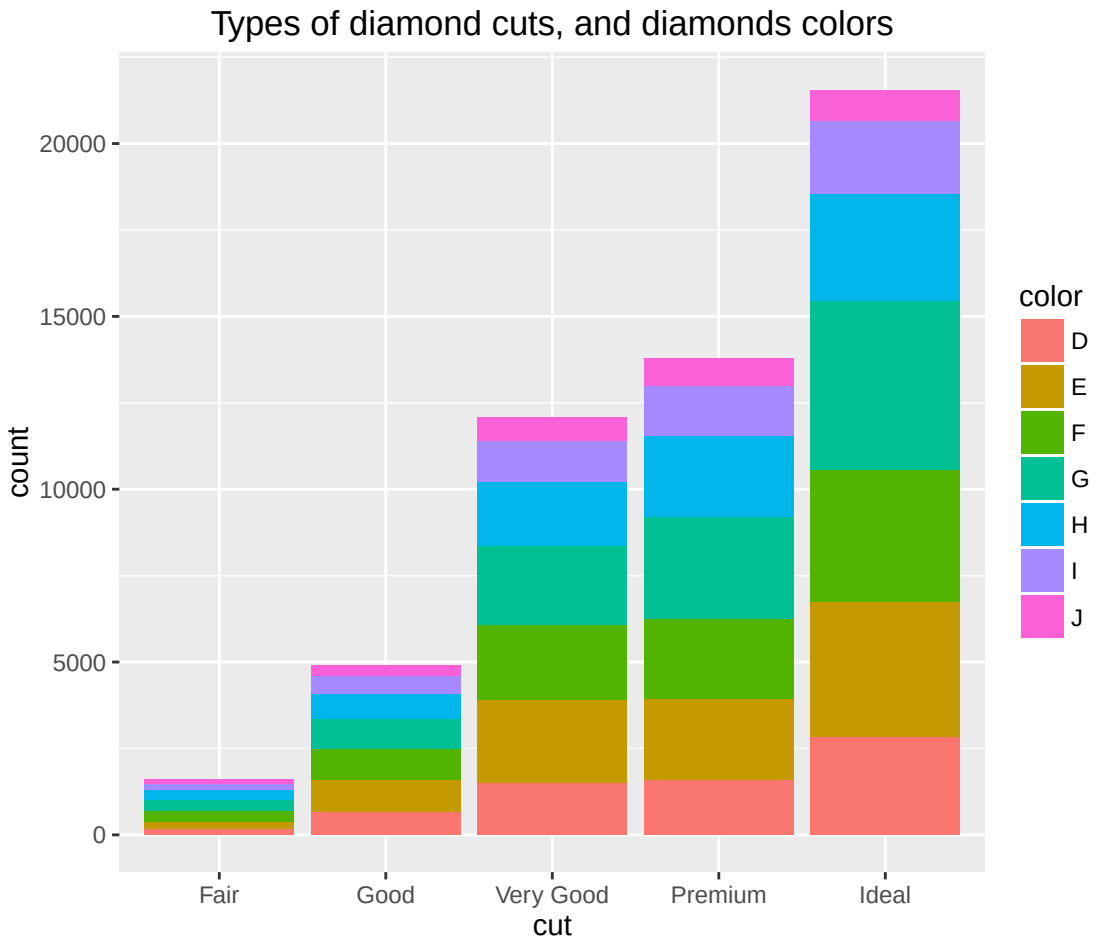
```
barp3 <- barp2 +
  ggtitle("Types of diamond cuts")
barp3
```



The diamond dataset gives additional information about the diamonds color (diamonds$color).

We can color the bars given the proportion of each of the colors present for each type of cuts:

```
barp4 <- ggplot(data=diamonds, aes(x=cut, fill=color)) + geom_bar() +
  ggtitle("Types of diamond cuts, and diamonds colors")
barp4
```



Pie charts: geom_bar() + coord_polar()

> A pie chart is a type of graph in which a circle is divided into sectors that each represent a proportion of the whole. (Wikipedia)

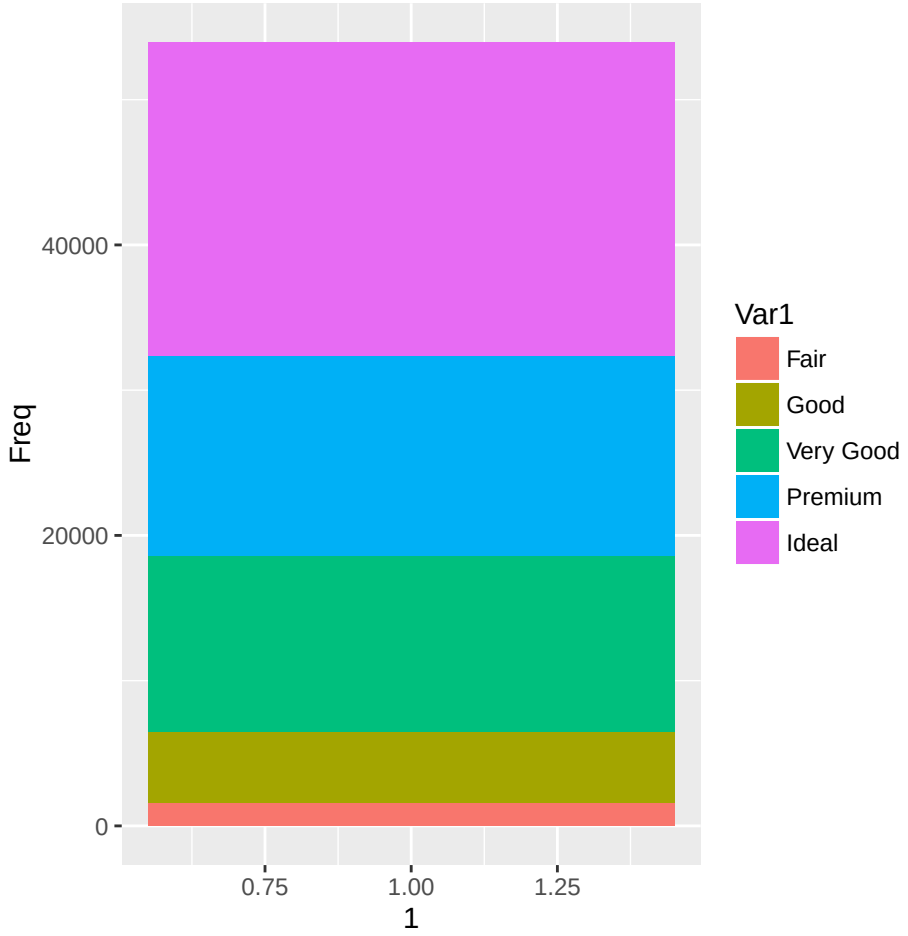
The base of a pie plot in ggplot2 is a stacked barplot.

We will first count how many occurences of each cut are present in the dataset using the **table** function.

```
# table: builds a contingency table of the counts at each combination of factor levels.
cuts_counts <- as.data.frame(table(diamonds$cut))
cuts_counts
```

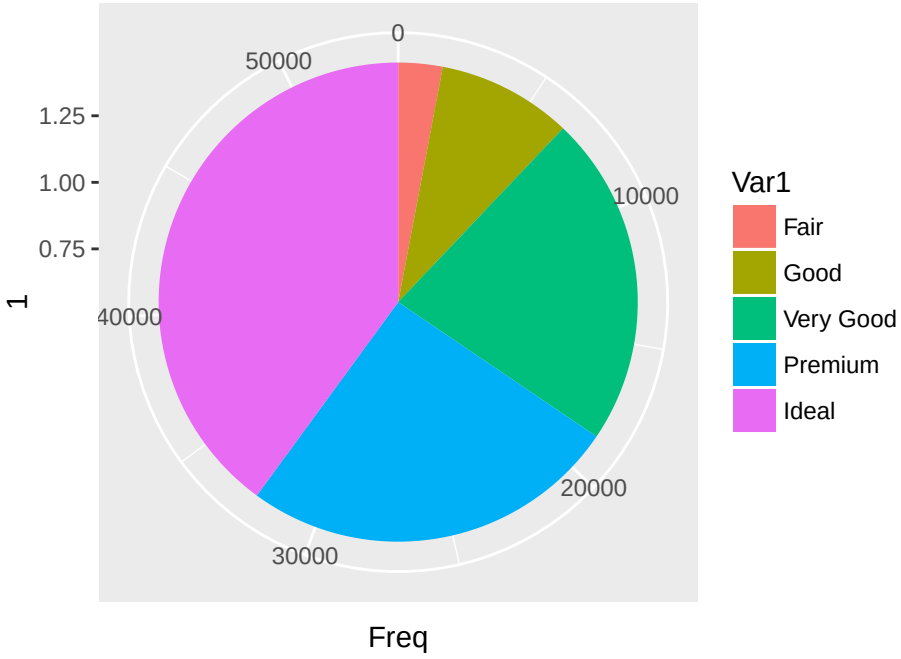
```
##        Var1  Freq
## 1      Fair  1610
## 2      Good  4906
## 3 Very Good 12082
## 4   Premium 13791
## 5     Ideal 21551
```

```
piep0 <- ggplot(cuts_counts, aes(x=1, y=Freq, fill=Var1)) +
        geom_bar(stat="identity")
piep0
```



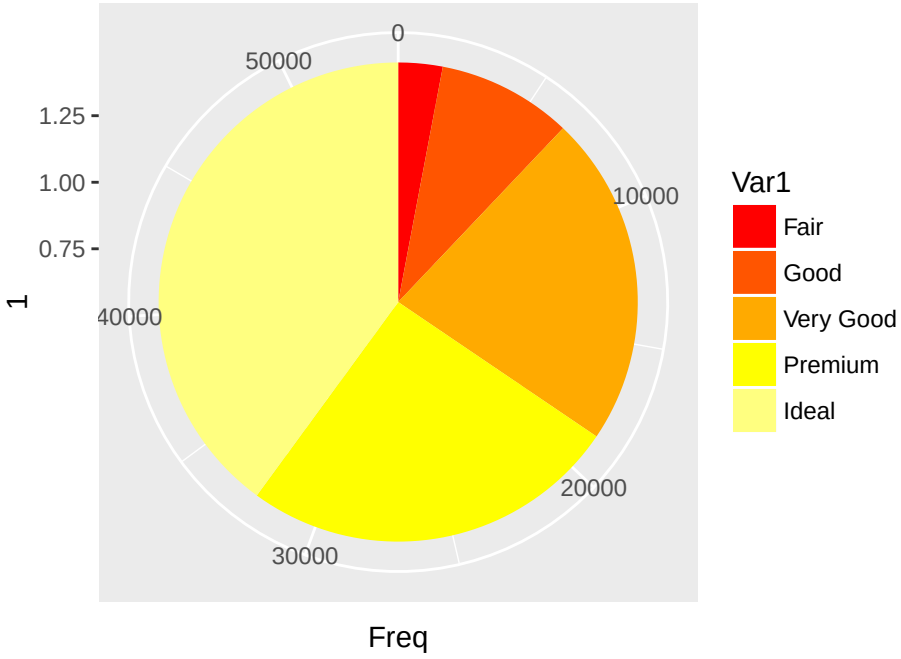
The convertion into a pie plot is done by transforming the coordinates to the polar coordinate system (most commonly used for pie charts):

```
# theta: variable to map the angle to
# start: offset of starting point from 12 o'clock in radians
piep1 <- piep0 + coord_polar(theta="y", start=0)
piep1
```



Change the default colors to the ones of your choice, the same way as we did for the barplot:

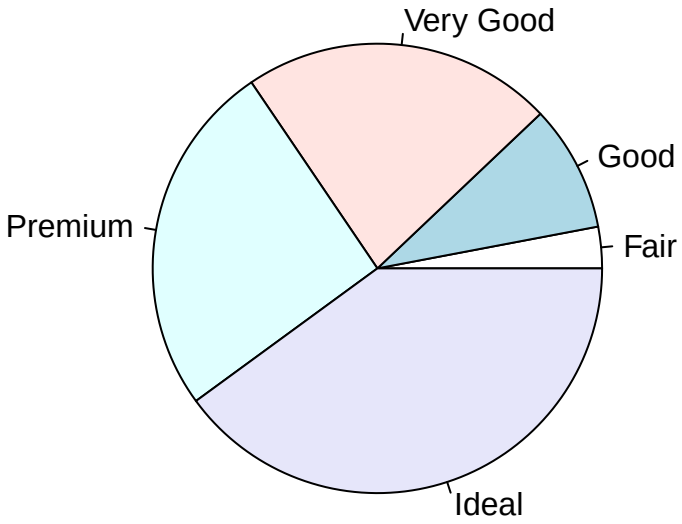
```
piep2 <- piep1 + scale_fill_manual(values=heat.colors(5))
piep2
```



Please note that it is rather quick and easy to produce a pie plot with the basic plotting function pie()

It is simply done with:

```
pie(table(diamonds$cut))
```

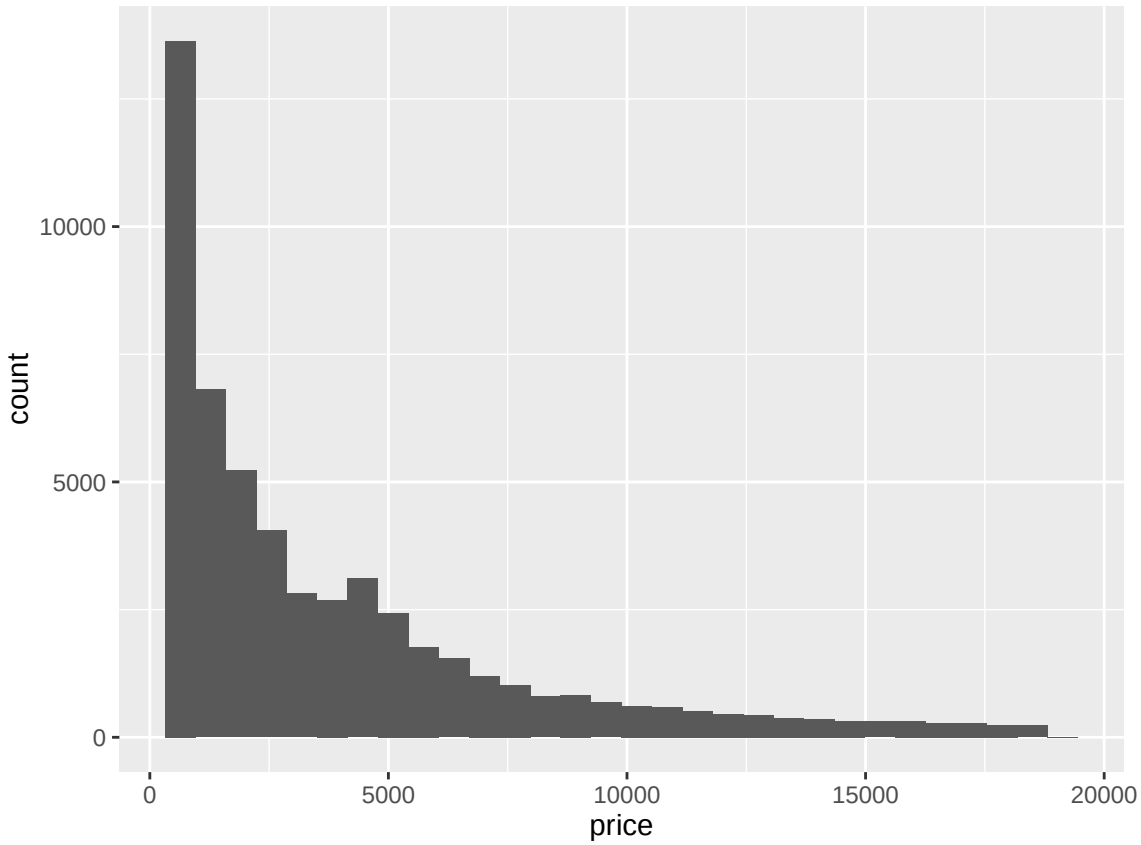


Histograms: geom_hist()

> A histogram is a plot that lets you discover, and show, the underlying frequency distribution of a set of continuous data. (Wikipedia)

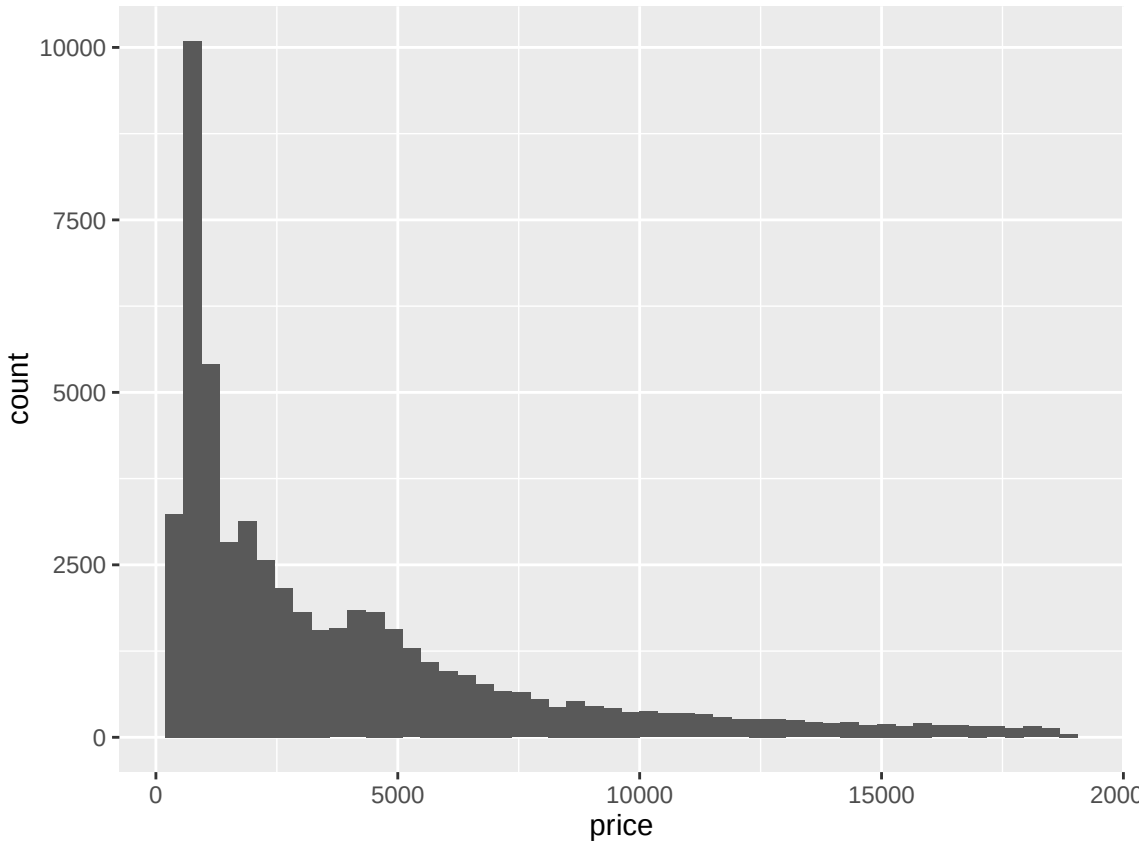
Let's start with the most basic histogram: we will display the diamonds price distribution:

```
histp0 <- ggplot(data=diamonds, aes(x=price)) + geom_histogram()
histp0
```



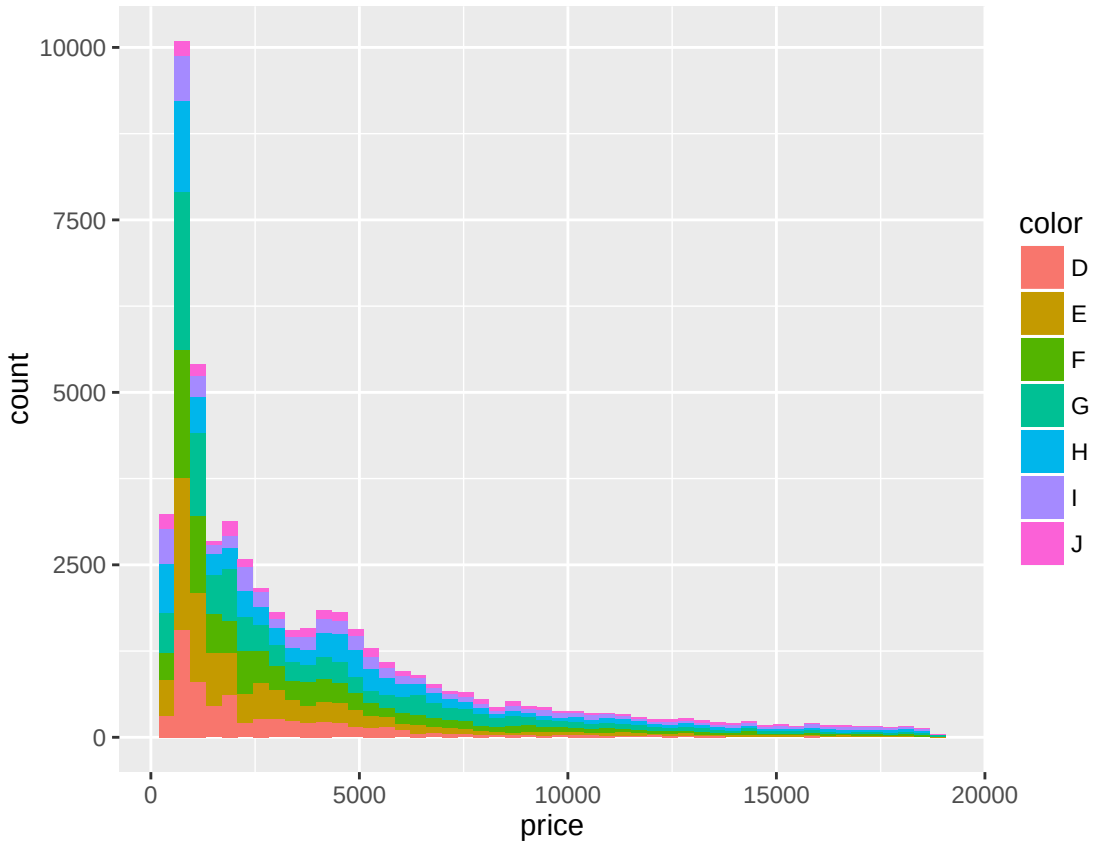
We can change the bin size (default to 30) to get higher resolution:

```
histp1 <- ggplot(data=diamonds, aes(x=price)) + geom_histogram(bins=50)
histp1
```



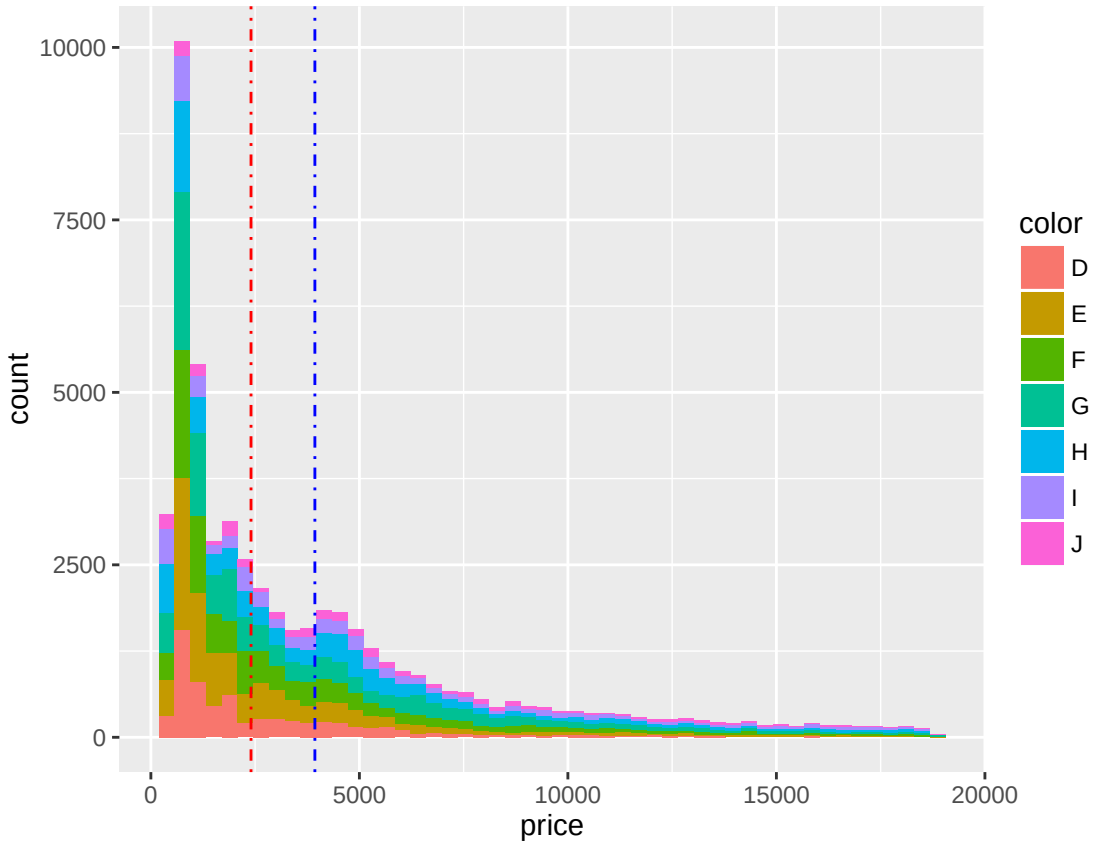
Now let's take into account the **color** of the diamond (J and D being the worst and best colors, respectively). Add this parameter as a ggplot aesthetic:

```
histp2 <- ggplot(diamonds, aes(x=price, fill=color)) +  geom_histogram(bins=50)
histp2
```



Add a **red vertical line** at the median price value, and a **blue vertical line** at the average price value using **geom_vline** layers:

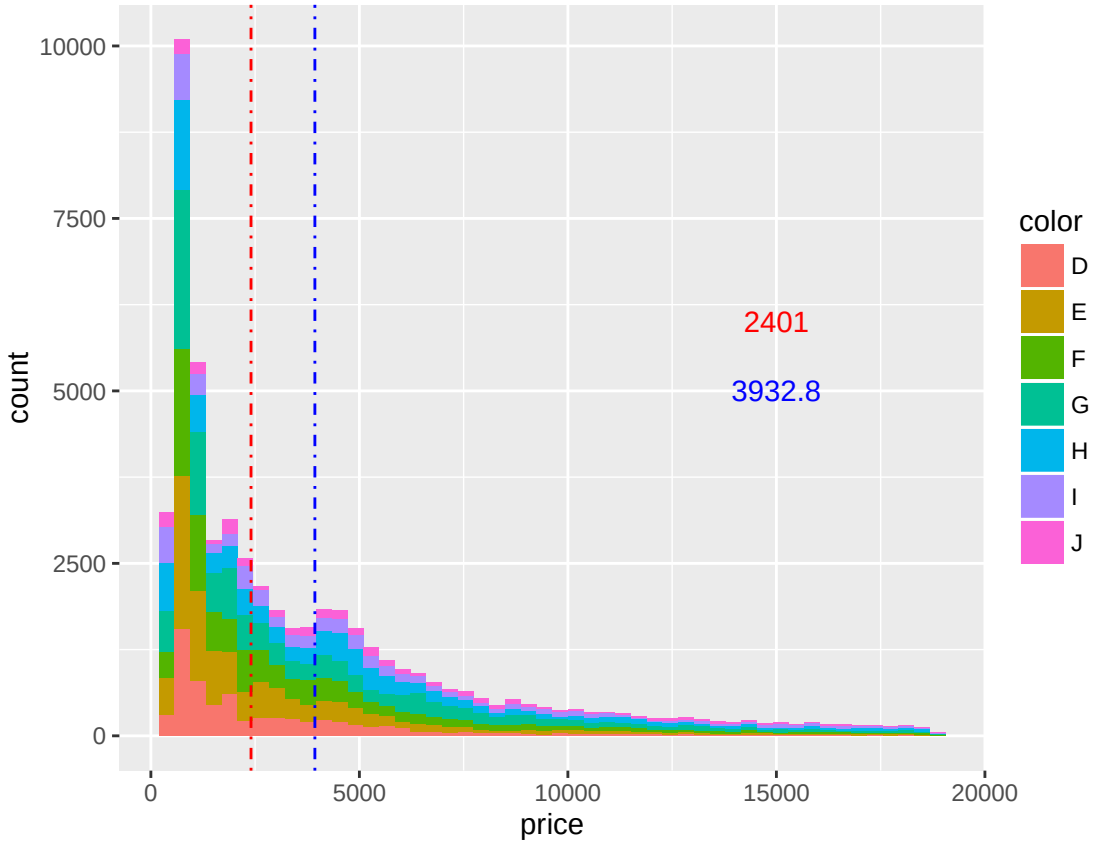
```
# Calculate median and average/mean, and round them to 2 decimals.
median_price <- round(median(diamonds$price),2)
mean_price <- round(mean(diamonds$price),2)

# geom_vline() for vertical lines: xintercept is their starting coordinate on the x axis
histp3 <- histp2 +
  geom_vline(xintercept=median_price, linetype="dotdash", colour="red", size=0.5) +
  geom_vline(xintercept=mean_price, linetype="dotdash", colour="blue", size=0.5)
histp3
```


10000
7500
5000
2500
0
count
0
5000
10000
15000
20000
price
color
D
E
F
G
H
I
J

Exercice 4

Add text to histp3 that displays the mean and the median values

We want to obtain this plot:



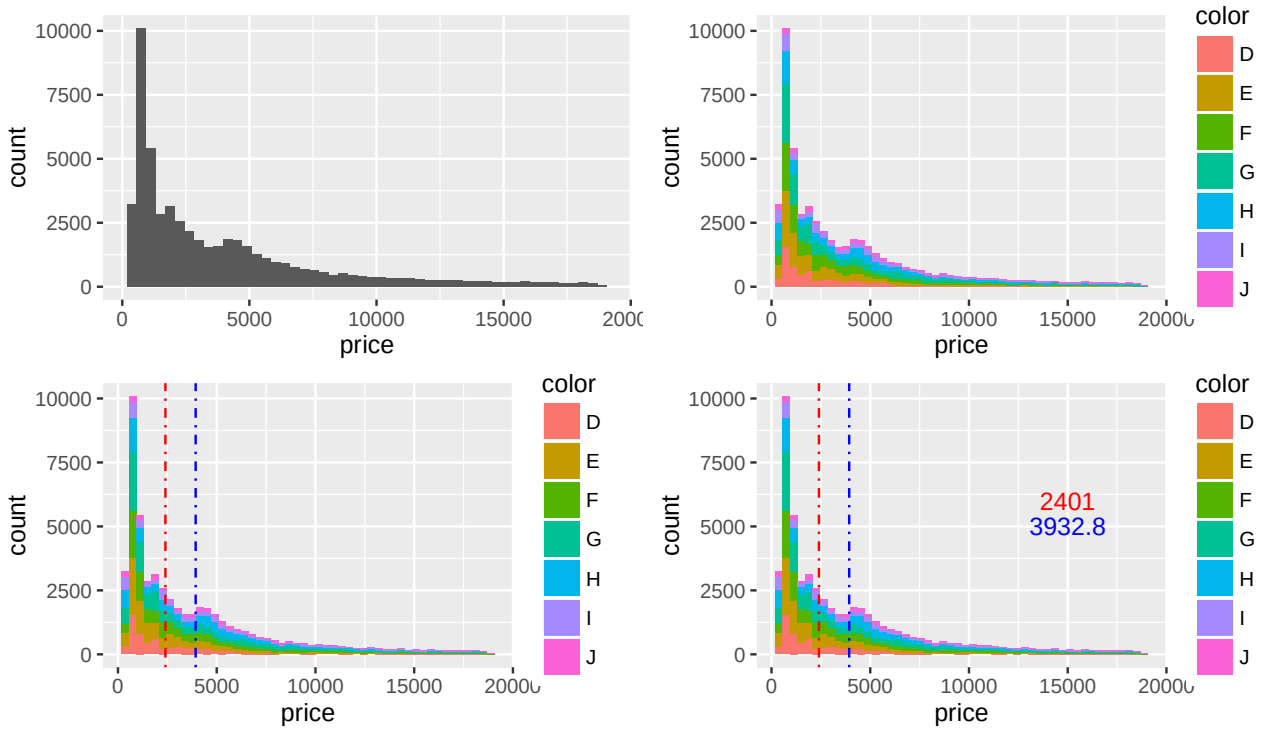
You will use:

- annotate layer, more specifically the following parameters:
  - geom
  - x and y
  - label
  - color

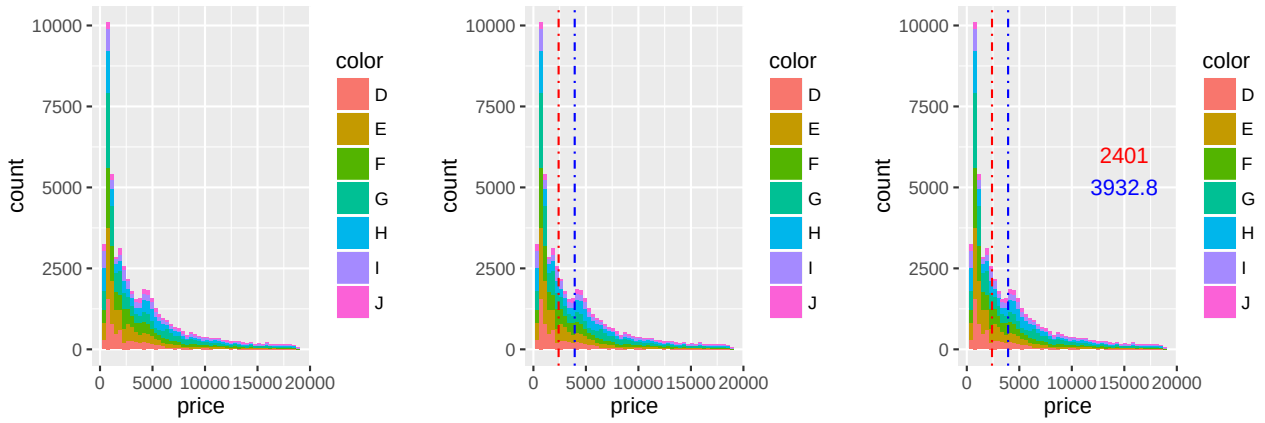
Function **grid.arrange** from **gridExtra** package allows to display several plots in one page.

Using grid.arrange, plots are organized in the page in a matrix-like manner of **ncol** columns and **nrow** rows.

```
grid.arrange(histp1,
             histp2,
             histp3,
             histp4,
             ncol=2,
             nrow=2)
```




```
grid.arrange(histp2,
             histp3,
             histp4,
             ncol=3,
             nrow=1)
```



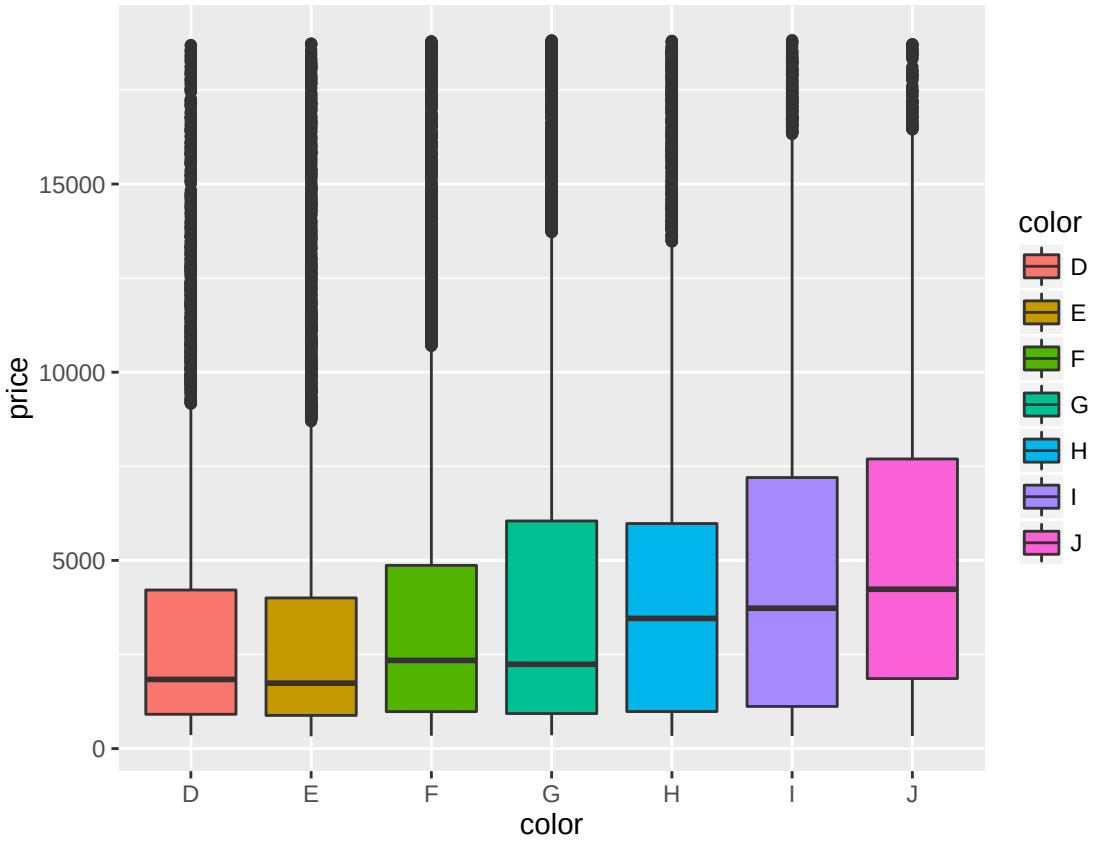
Boxplots: geom_boxplot()

> In descriptive statistics, a boxplot is a convenient way of graphically depicting groups of numerical data through their quartiles. (Wikipedia)

We will produce here boxplots of the distribution of diamond prices according to their colors.

Basic boxplot, colored given the diamond color:

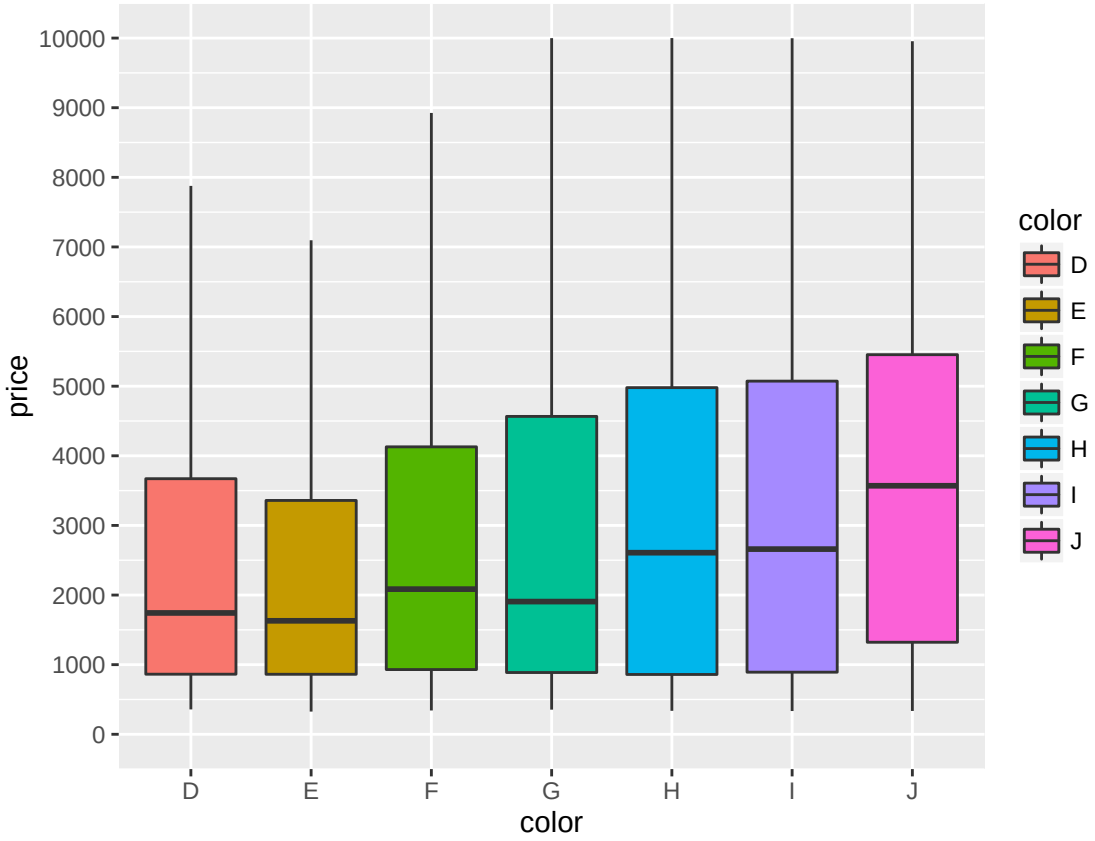
```
boxp0 <- ggplot(diamonds, aes(x=color, y=price, fill=color)) +
  geom_boxplot()
boxp0
```



For more clarity we might want to remove the outliers, reduce the y axis and add more breaks to it:

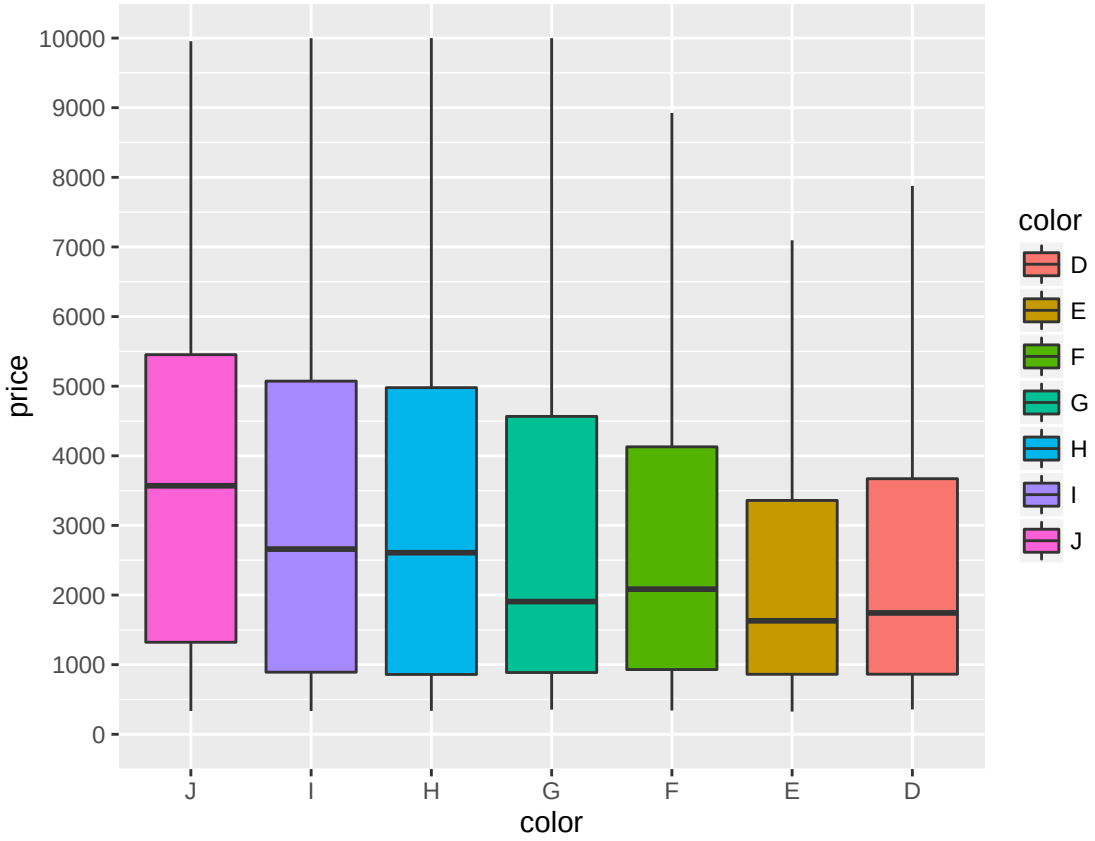
```
# outlier.size=NA in geom_boxplot(): ignore outliers
# scale_y_continuous(): modify y axis.
  # limits: reducing the axis
  # breaks: customizing number of breaks
boxp1 <- ggplot(diamonds, aes(x=color, y=price, fill=color)) +
  geom_boxplot(outlier.size=NA) +
  scale_y_continuous(limits=c(0,10000), breaks=seq(0,10000,1000))
boxp1
```

```
## Warning: Removed 5222 rows containing non-finite values (stat_boxplot).
```



We can manually reorder the x axis (colors):

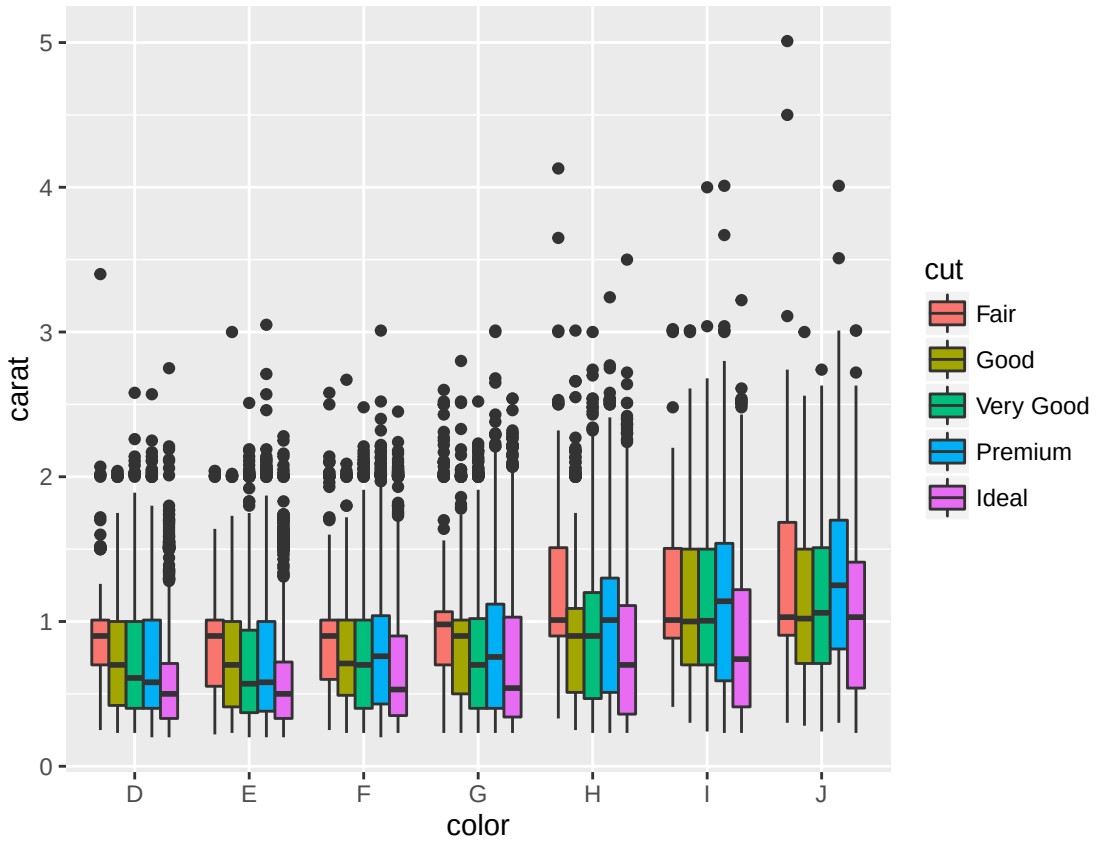
```
boxp2 <- boxp1 +
  scale_x_discrete(limits=c("J","I","H","G","F", "E", "D"))
boxp2
```



We will next take into account the **diamond cut** property of the diamonds.

If that property is taken into account at the **initial aesthetics level**, each of the previous boxplot will be divided into several boxplots representing the different diamonds cuts:

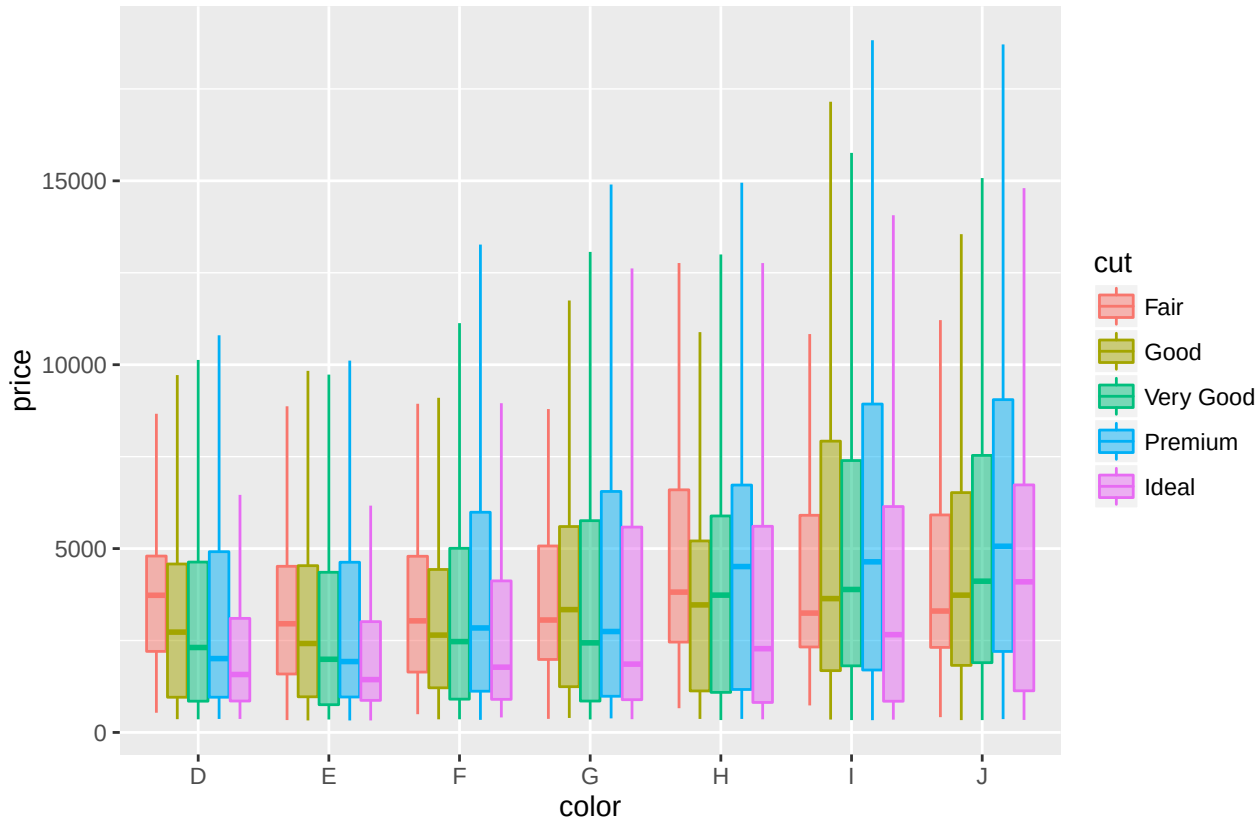
```
boxp3 <- ggplot(diamonds, aes(x=color, y=carat, fill=cut)) +
  geom_boxplot()
boxp3
```



Try to play a bit with:

- position_dodge: separate or make boxes closer.
- alpha: control boxes transparency.
- color: control colors of the lines for each box (versus fill that controls the inside color of the boxes).

```
boxp4 <- ggplot(diamonds, aes(x=color, y=price, fill=cut, color=cut)) +
  geom_boxplot(outlier.size=NA, alpha=0.5, position = position_dodge(width = 0.8))
boxp4
```



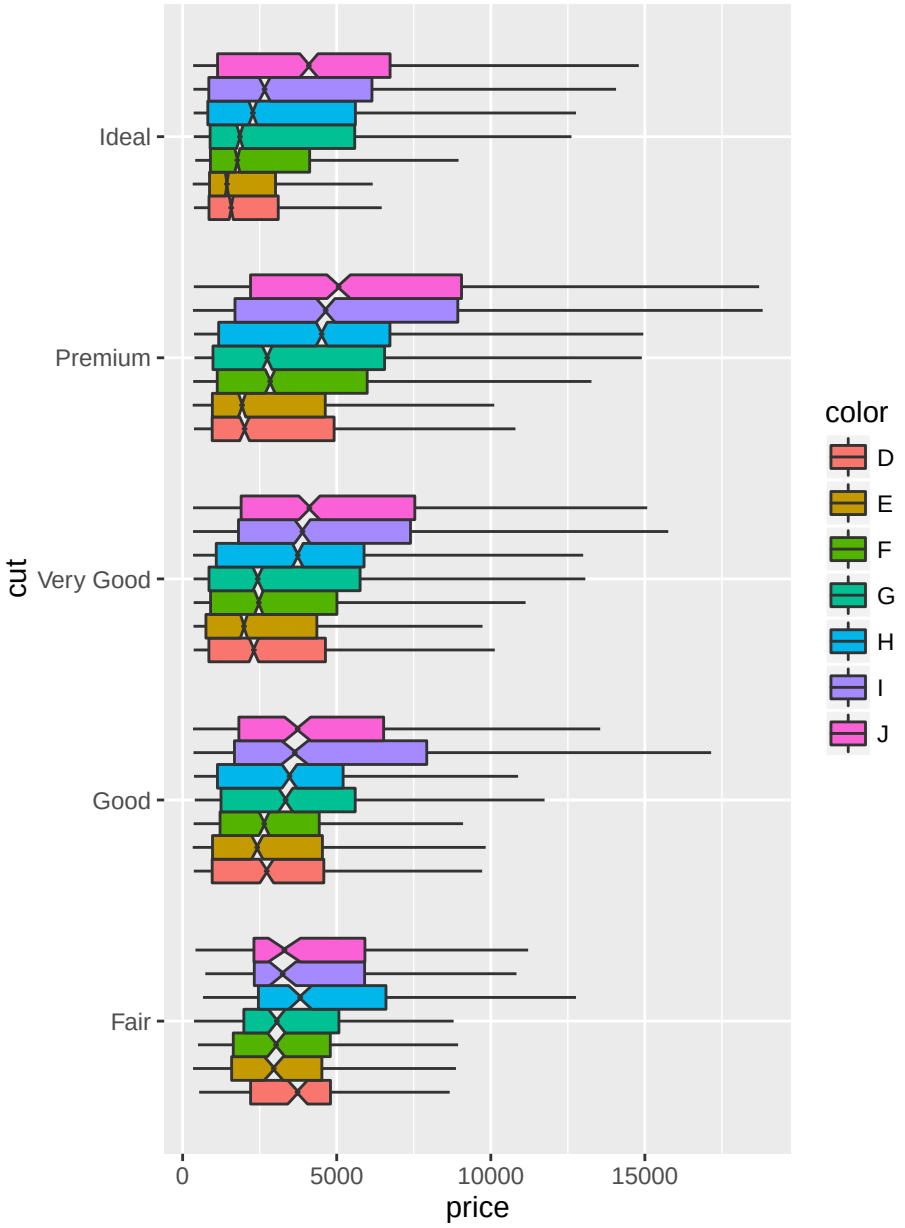
Finalizing: add previous parameters and flip coordinates (x as y, y as x):

```
boxp5 <- boxp4 +
  scale_y_continuous(limits=c(0,10000), breaks=seq(0,10000,1000)) +
  coord_flip()
boxp5
```


J
I
H
G
F
E
D
color
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
price
cut
Fair
Good
Very Good
Premium
Ideal

Exercise 5

Produce the following notched boxplot



Notches are used to compare groups; if the notches of two boxes do not overlap, this is evidence that the medians differ.

Additionally to what we have already seen, you will need:

- notch parameter from geom_boxplot()
- notchwidth parameter from geom_boxplot()
- coord_flip() layer

Scatter plots: geom_point()

> A scatter plot is a type of plot or mathematical diagram using Cartesian coordinates to display values for typically two variables for a set of data. (Wikipedia)

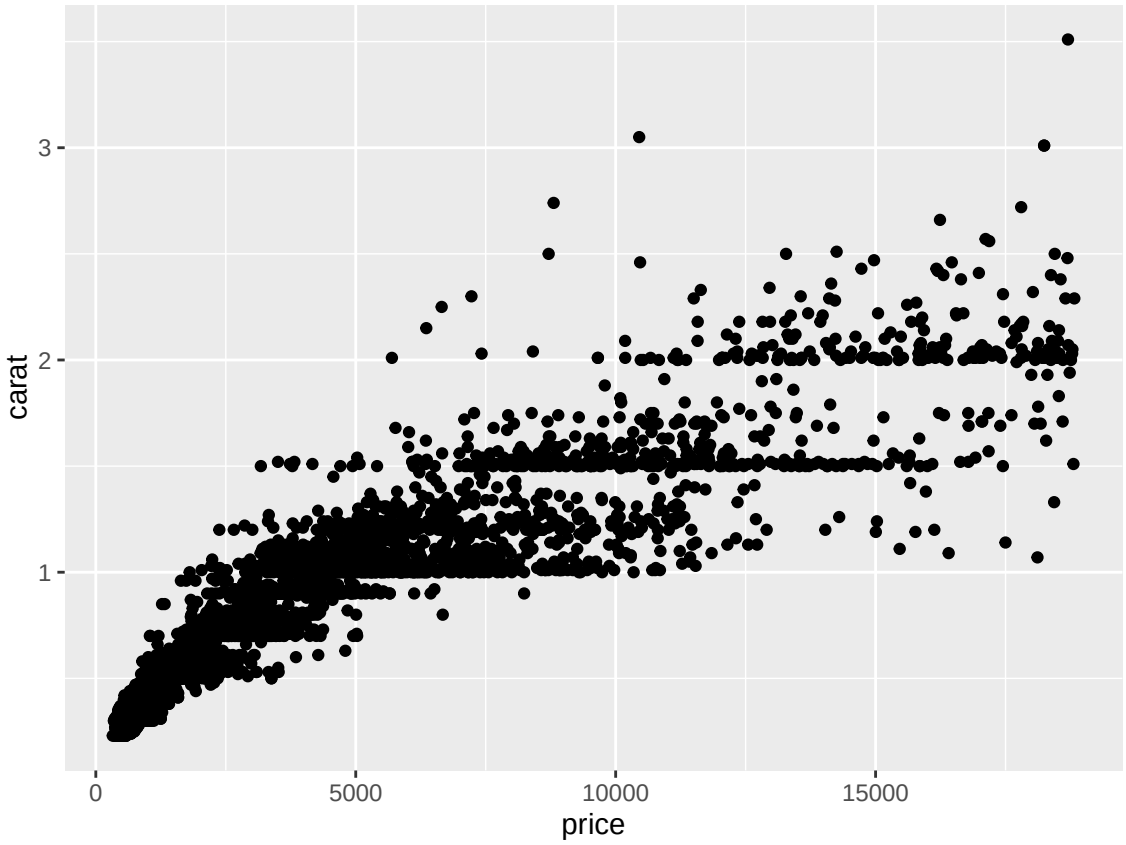
We will use a random subset of 5000 diamonds (only for a less busy plot) for producing the scatter plots.

```
# Take a 5000 diamond subset (randomly using the sample function)
diamonds2 <- diamonds[sample(nrow(diamonds), 5000),]
dim(diamonds2)
```

```
## [1] 5000   10
```

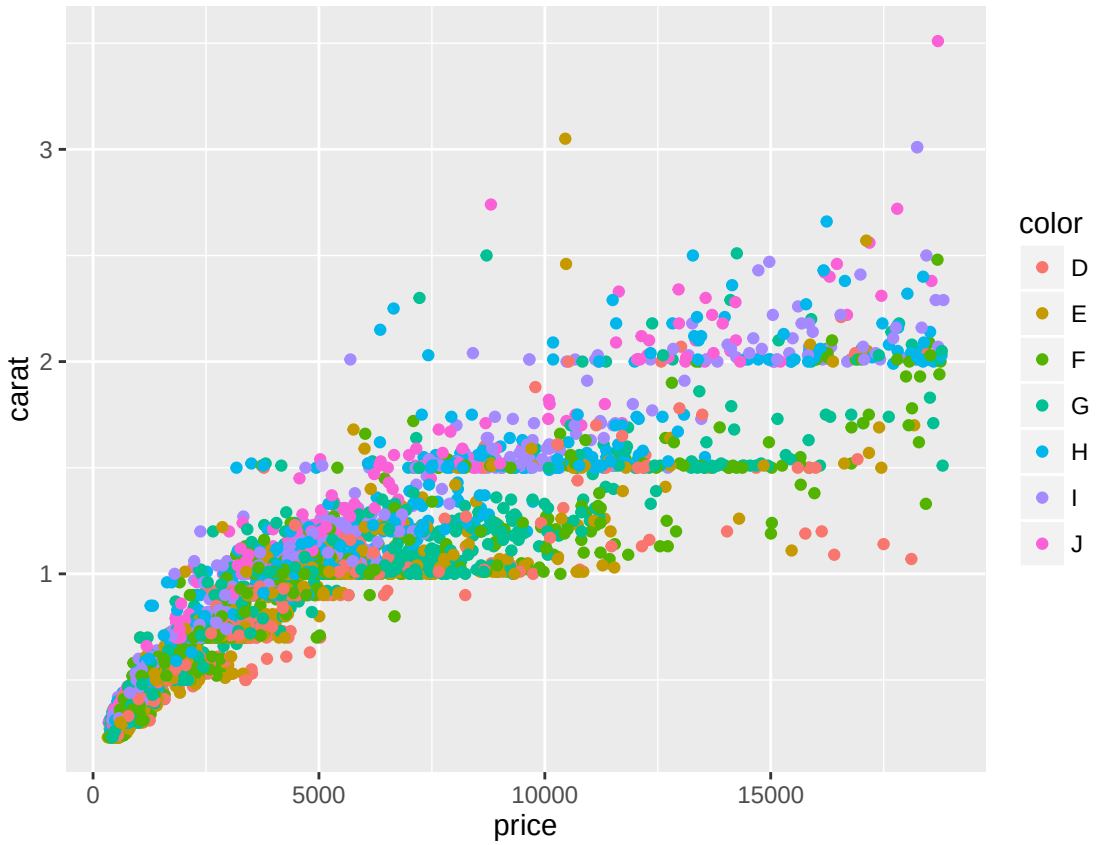
We will plot the **carat** versus **price** of each diamond. Each point in the following scatter plots represents a single diamond.

```
scatp0 <- ggplot(diamonds2, aes(x=price, y=carat)) +
    geom_point()
scatp0
```



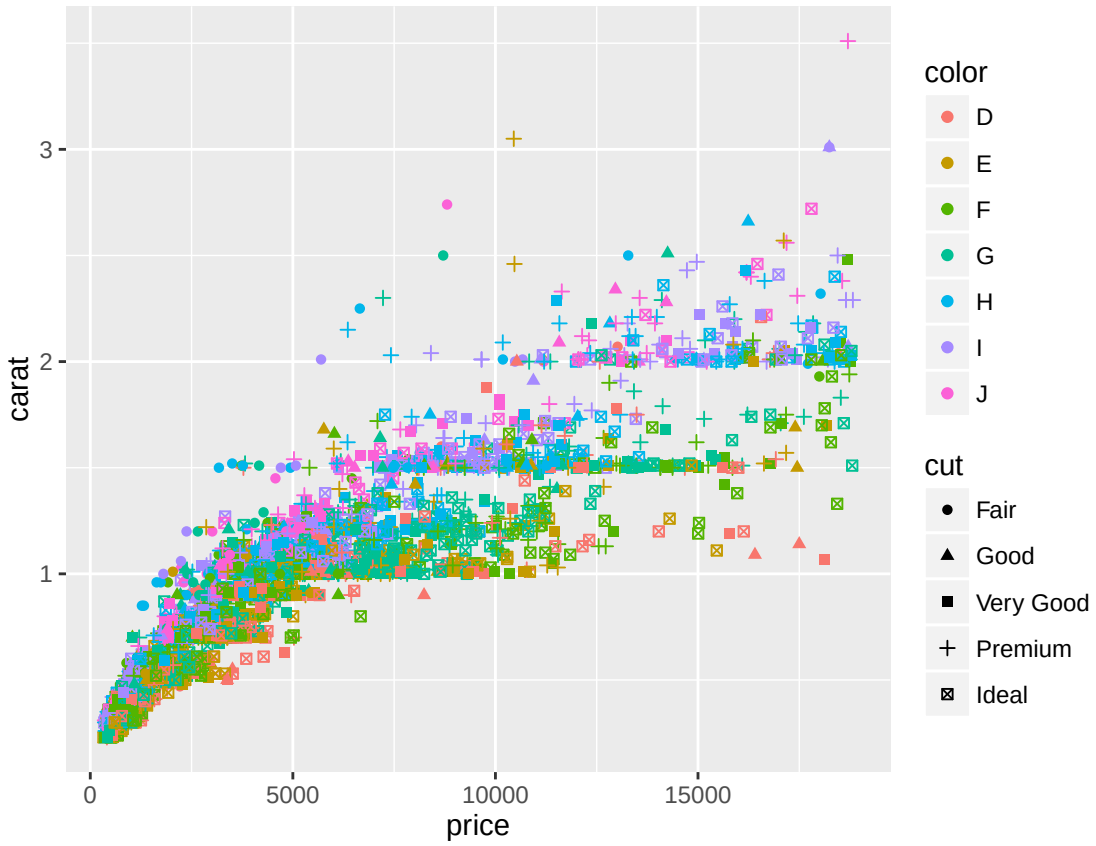
Add an information to the graph: color the points according to the diamond color:

```
scatp1 <- ggplot(diamonds2, aes(x=price, y=carat)) +
    geom_point(aes(colour=color))
scatp1
```



Change points shape according to the diamond shape:

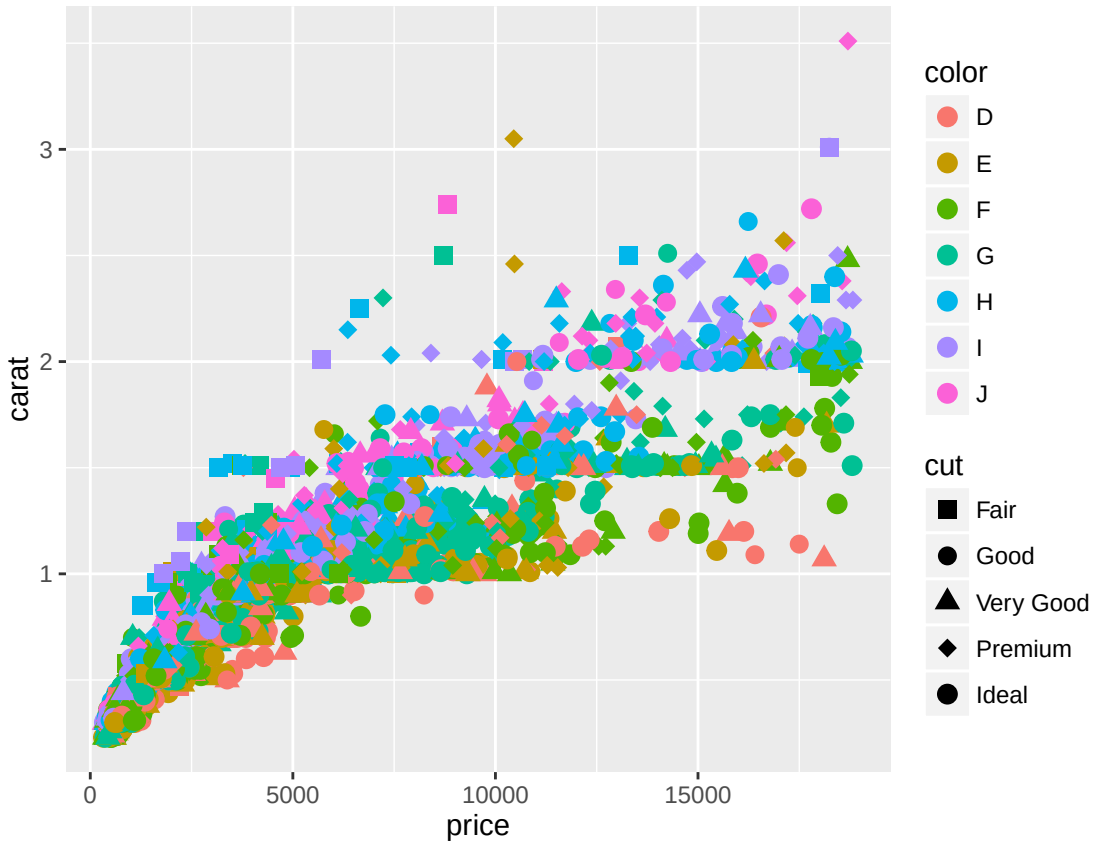
```
scatp2 <- ggplot(diamonds2, aes(x=price, y=carat)) +
    geom_point(aes(colour=color, shape=cut))
scatp2
```



Note how legends are being added as you had layers of information to the plot.

You might want to control the **shape** and **size** of the points (refer to the point shape table introduced in the basic plotting):

```
# size in geom_point(): point size
# scale_shape_manual(): control of the shapes
scatp3 <- ggplot(diamonds2, aes(x=price, y=carat)) +
  geom_point(aes(colour=color, shape=cut), size=3) +
  scale_shape_manual(values=15:19)
scatp3
```



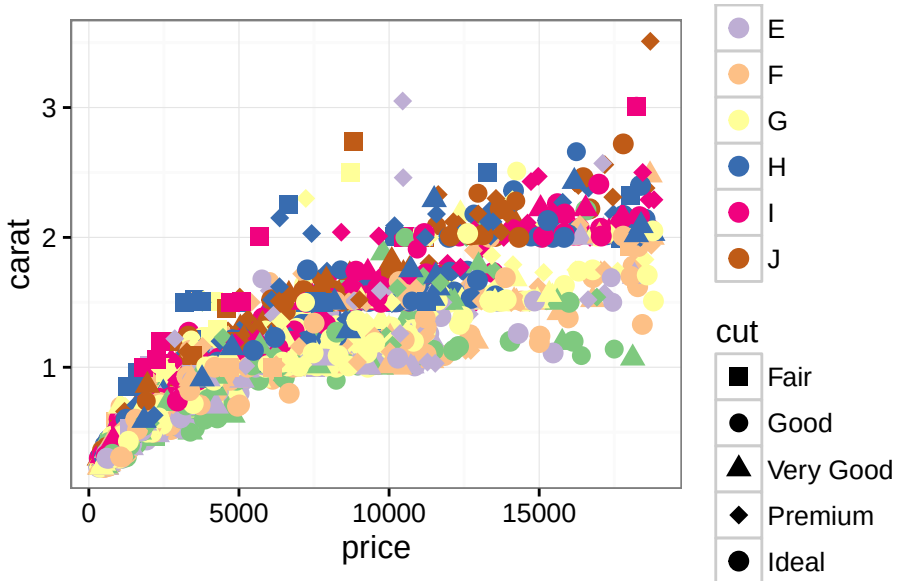
We will use the RColorBrewer package previously introduced to color our plot using different palettes.

Palettes from that package are the following ones:

Blues BuGn BuPu GnBu Greens Greys Oranges OrRd PuBu PuBuGn PuRd Purples RdPu Reds YlGn YlGnBu YlOrBr YlOrRd BrBG PiYG PRGn PuOr RdBu RdGy RdYlBu RdYlGn Spectral Accent Dark2 Paired Pastel1 Pastel2 Set1 Set2 Set3

Exercise 6

Modify scatp3 and give the palettes a try to obtain the following plot (or any plot that you like best):

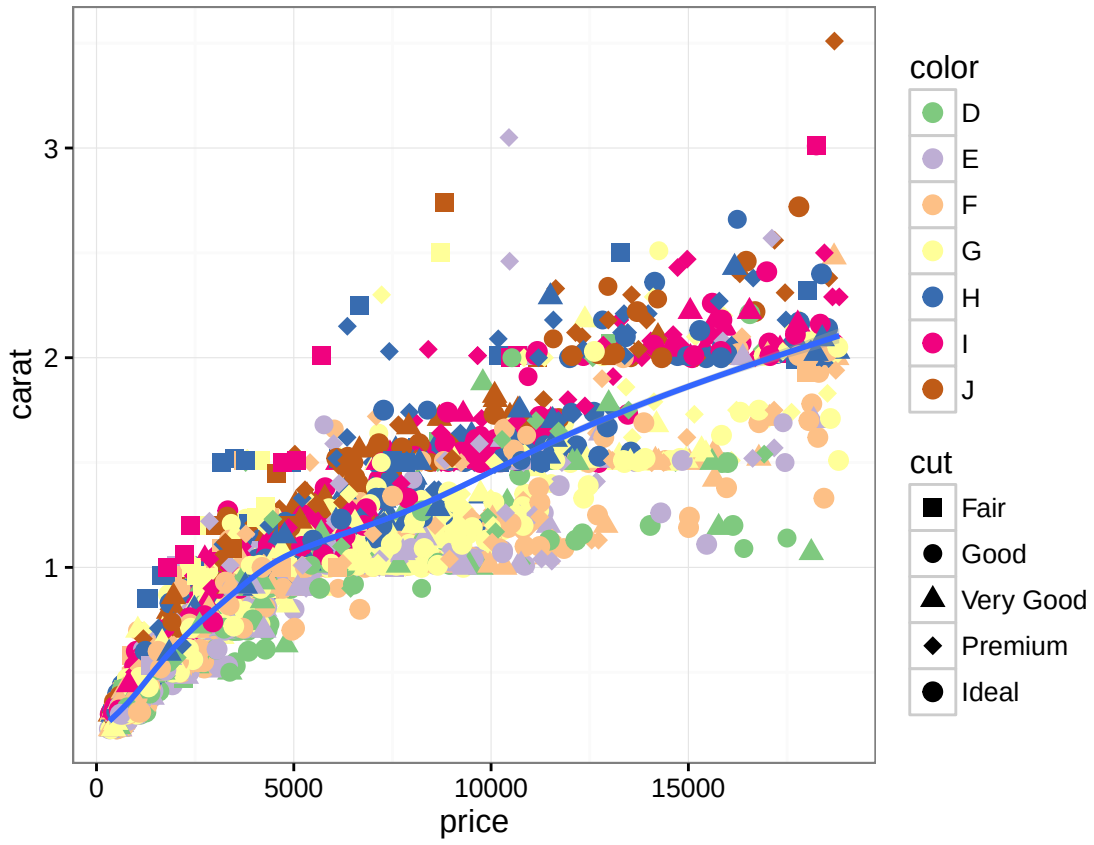


You will need more layers:

- scale_colour_brewer: change color scheme
- theme_bw: remove the grey background

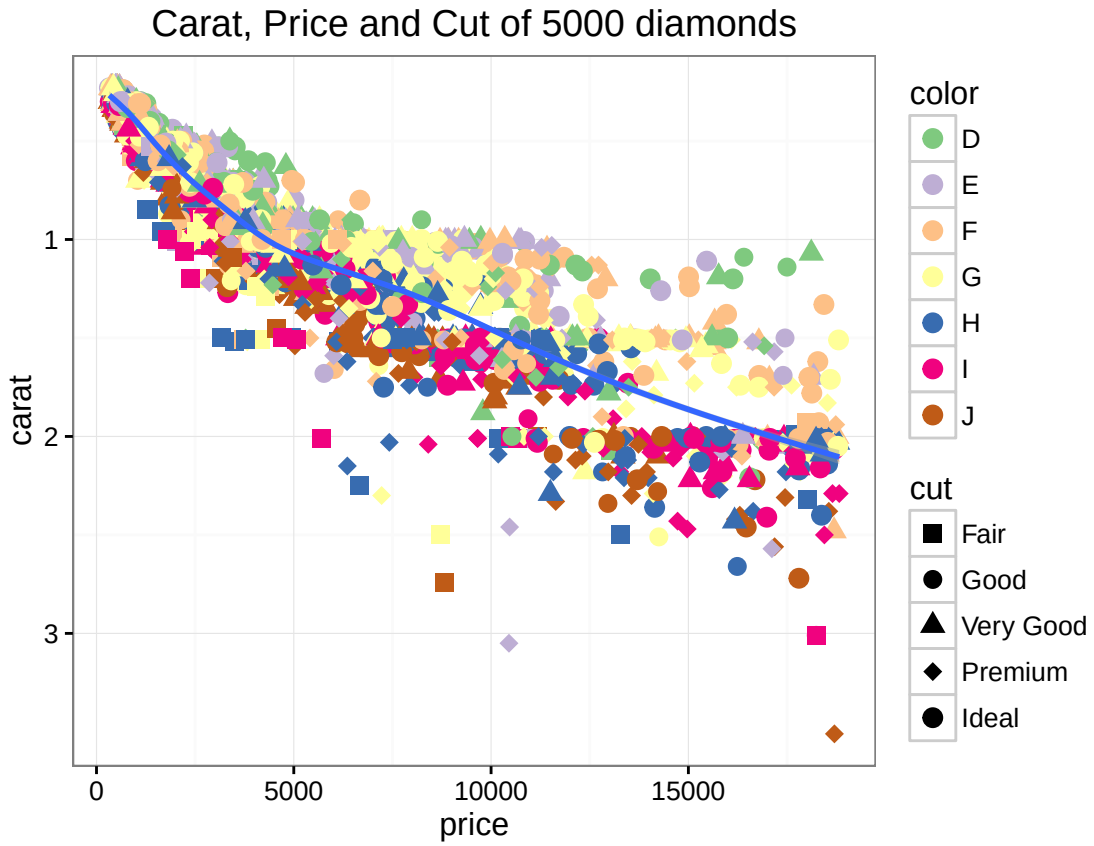
We will now add a linear regression line to scatp4:

```
scatp5 <- scatp4 + geom_smooth()
scatp5
```



Finally, we will reverse the y axis, and add a title to the plot:

```
scatp6 <- scatp5 +
  scale_y_reverse() +
  ggtitle("Carat, Price and Cut of 5000 diamonds")
scatp6
```



We can save some scatter plots into a jpeg file, and **increase the dimensions of the output file** (by default for png/pdf/tiff formats, width=height=480 pixels):

```
jpeg("all_scatter_plots.jpg", width=1000, height=500)
grid.arrange(scatp1,
             scatp2,
             scatp3,
             scatp4,
             scatp5,
             scatp6,
             nrow=2,
             ncol=3)
dev.off()
```

The process is similar to save graphicals in **pdf, png, bmp or tiff formats:**

```
pdf("scatter1.pdf", width=7, height=7)
  scatp1
dev.off()

png("scatter2.png", width=480, height=480)
  scatp2
dev.off()

tiff("scatter3.tiff", width=480, height=480)
  scatp3
```

```
dev.off()

bmp("scatter4.bmp", width=480, height=480)
  scatp4
dev.off()
```

Additional "non ggplots"

Some plots are easily produced using additional packages.

Here, we will introduce:

- heatmap.2 from the gplots package
- venn.diagram from the VennDiagram package

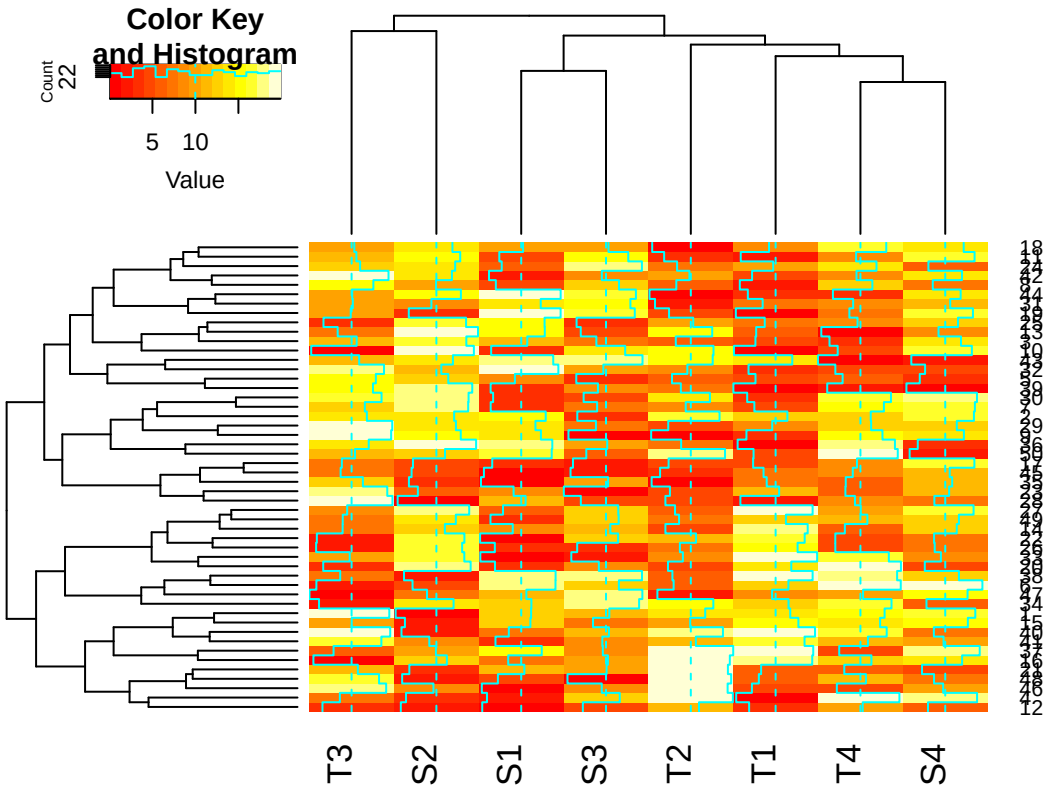
Heatmaps: heatmap.2()

We will produce a heatmap from a random number matrix:

```
# seq(from=1, to=2, by=0.5): generate sequence from 1 to 2 with 0.5 steps (1.0 1.5 2.0)
# sample(x, size=n): randomly select n elements from x
# paste(x, y, sep="_"): pastes x, y (as many elements as needed) together,
  # separated here with "_"
in_heat <- matrix(sample(seq(0, 20, 0.001), 400),
                  nrow=50,
                  ncol=8,
                  dimnames=list(1:50, c(paste("S", 1:4, sep=""), paste("T", 1:4, sep=""))))
```

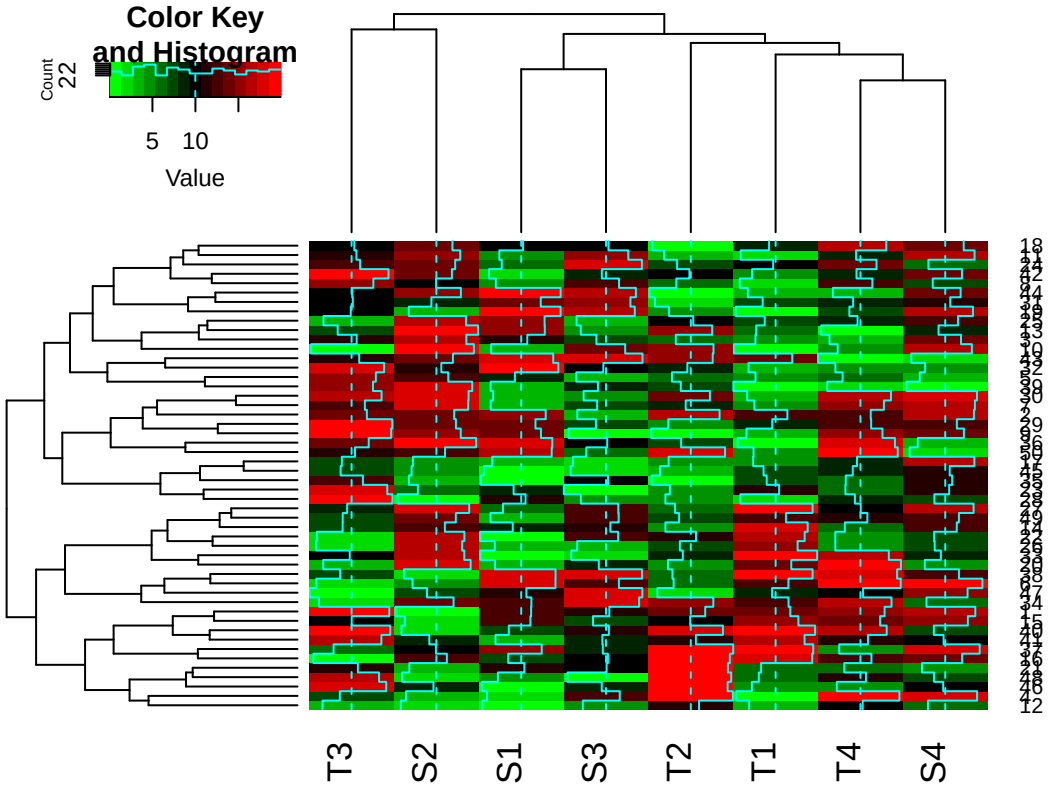
Let's see what the default heatmap looks like:

```
heatmap.2(in_heat)
```

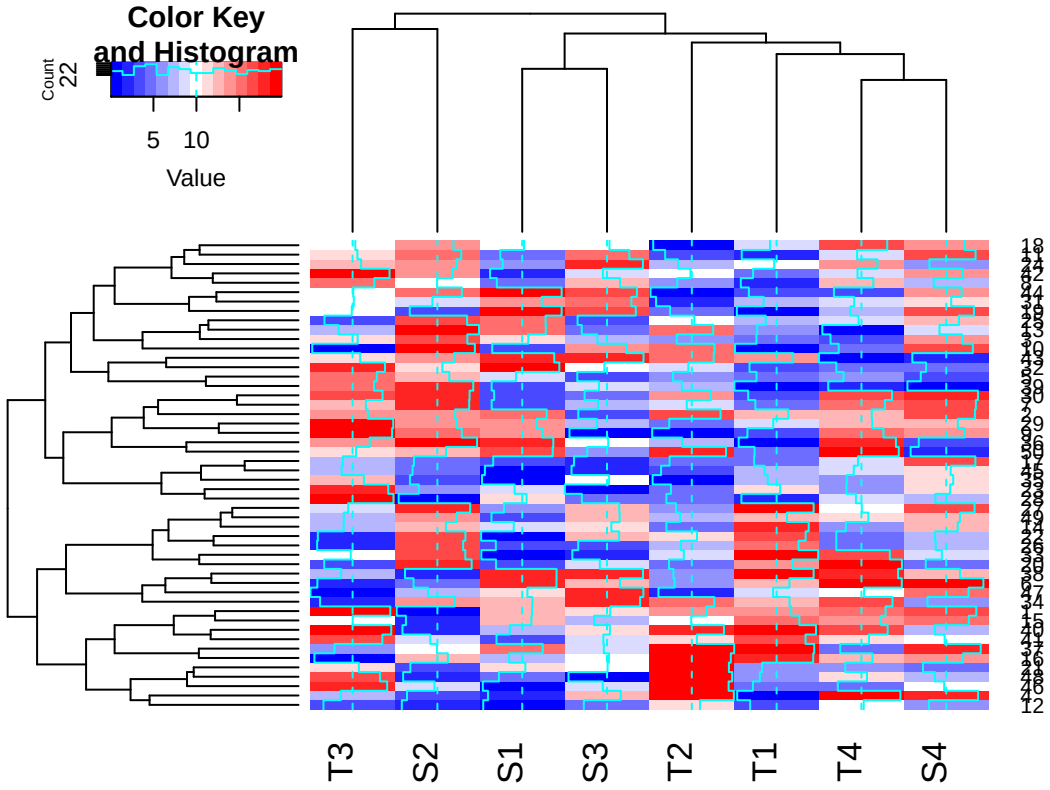


We can change the color scheme to green-red blue-red

```
heatmap.2(in_heat, col="greenred")
```



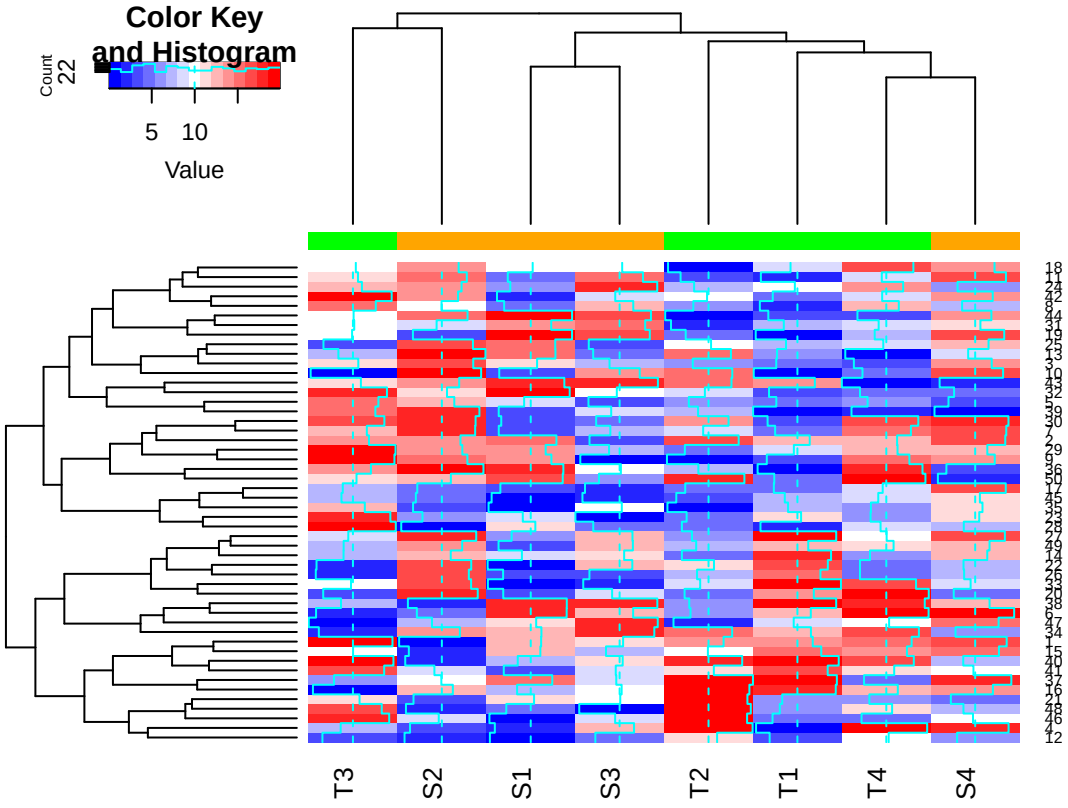
```
heatmap.2(in_heat, col="bluered")
```



Now imagine that you are analyzing an experiment, and samples T (1, 2 and 3) come from a different experimental group as samples S (1, 2 and 3).

Apart from the names, it would be interesting to represent experimental groups with color boxes, especially for large experiments.

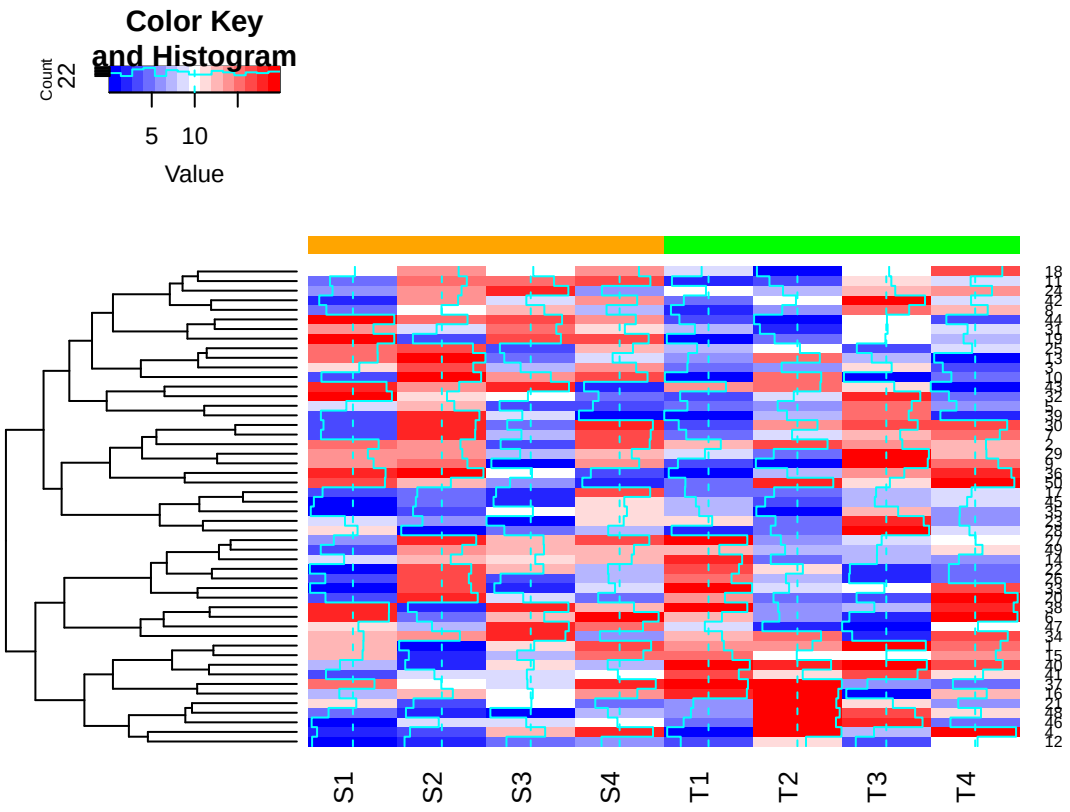
```
# color vector
mycol <- colnames(in_heat)
# grep(pattern, x): search for matches to pattern within each element of x
mycol[grep("T", mycol)] <- "green"
mycol[grep("S", mycol)] <- "orange"
# ColSideColors: add colored boxes.
heatmap.2(in_heat, col="bluered",
          ColSideColors=mycol)
```



By default, heatmap.2 uses hierarchical clustering to reorder rows and columns of the input matrix.

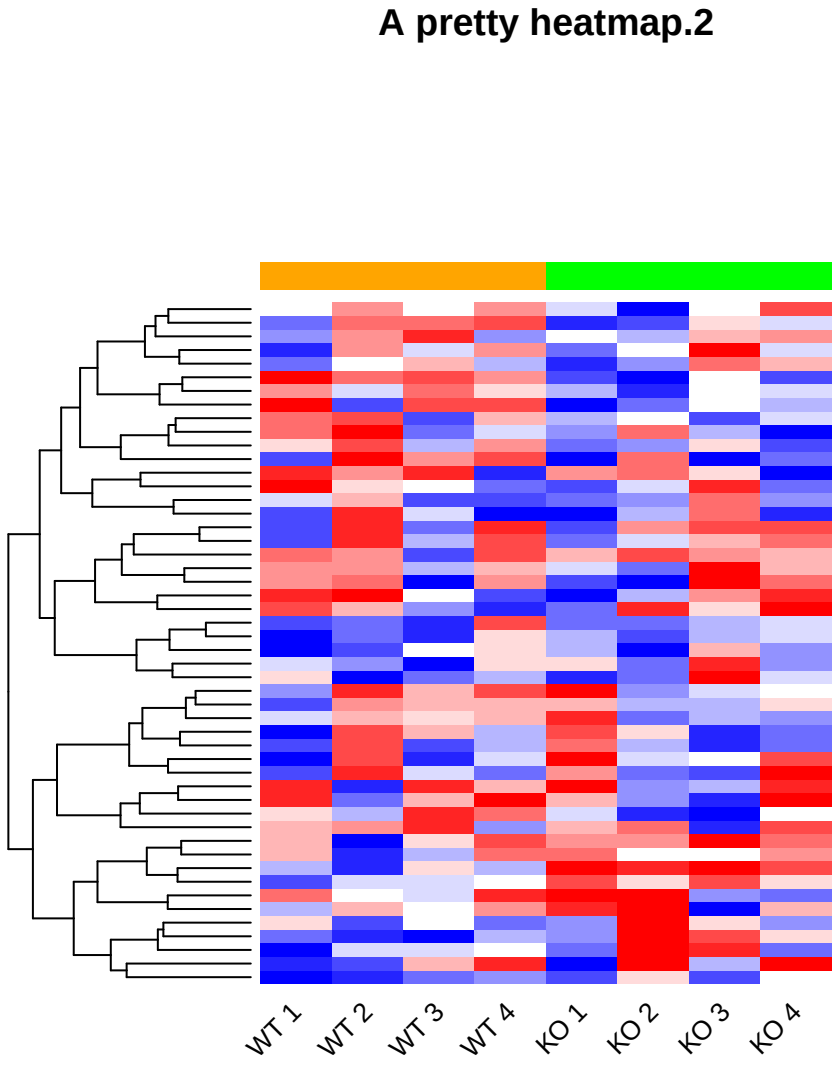
Clustering of either the rows, the columns, can be turned off, so as to keep the matrix the way it is inputted (rows, columns, or both):

```
# Rowv: if and how the row dendrogram should be reordered
# Colv: if and how the column dendrogram should be reordered
heatmap.2(in_heat, col="bluered",
          ColSideColors=mycol,
          Colv=FALSE,
          dendrogram="row",
          Rowv=TRUE)
```


Color Key
and Histogram
Count
22
5
10
Value
S1
S2
S3
S4
T1
T2
T3
T4

Exercise 7

Create the following heatmap:



- Have a look at the (very exhaustive) help page of the function: ?heatmap.2
- You will need:
  - dendrogram
  - Colv
  - labCol and labRow
  - strCol
  - trace
  - key
  - main

Venn Diagrams: venn.diagram()

> A Venn diagram (also known as a set diagram or logic diagram) is a diagram that shows all possible logical relations between a finite collection of different sets (Wikipedia).

Let's first create 2 random vectors:

```
vec1 <- sample(seq(0,100, 1), 40)
vec1
```

```
##  [1]   5  80  58  69  90  84  44  45  32  17  42  97  24  77  52  57  30
## [18]  50  37   9   2  89  29  28  56  15   4  34  60  14  18  94  39  91
## [35]  41   0 100  10  48  72
```

```
vec2 <- sample(seq(0,100, 1), 30)
vec2
```

```
##  [1] 57 21 76 22 81 40 14 54 38 31 89  5 58 73 59 67 12 25 29 26 19  9 10
## [24] 88 41 74 32  2 28 87
```

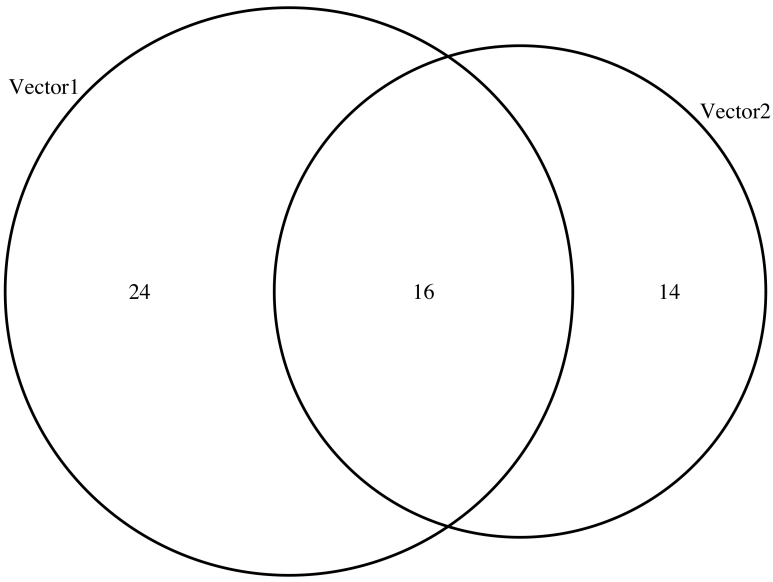
Function venn.diagram from package VennDiagram takes a list as an input. We will create a list with names containing both vectors.

```
mylist <- list(Vector1=vec1, Vector2=vec2)
mylist
```

```
## $Vector1
##  [1]   5  80  58  69  90  84  44  45  32  17  42  97  24  77  52  57  30
## [18]  50  37   9   2  89  29  28  56  15   4  34  60  14  18  94  39  91
## [35]  41   0 100  10  48  72
##
## $Vector2
##  [1] 57 21 76 22 81 40 14 54 38 31 89  5 58 73 59 67 12 25 29 26 19  9 10
## [24] 88 41 74 32  2 28 87
```

Let's plot the most basic Venn diagram. Each component of the list represents a circle, and the plot will be saved in the VennDiagram.tiff file in the current directory.

```
venn.diagram(mylist,
             filename="VennDiagram.tiff")
```

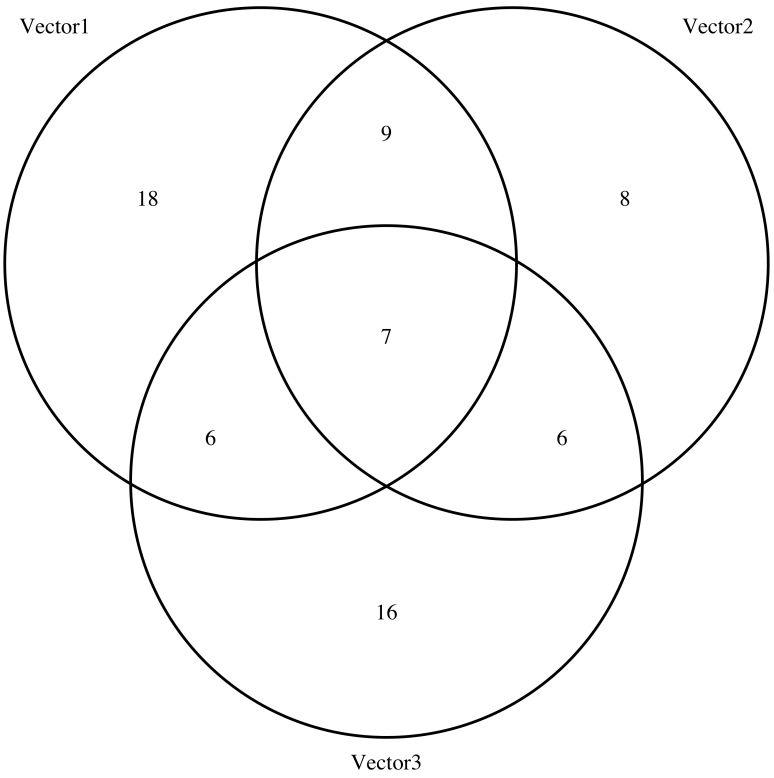


This is a two-set Venn diagram.

venn.diagram works with list up to 5 vectors!

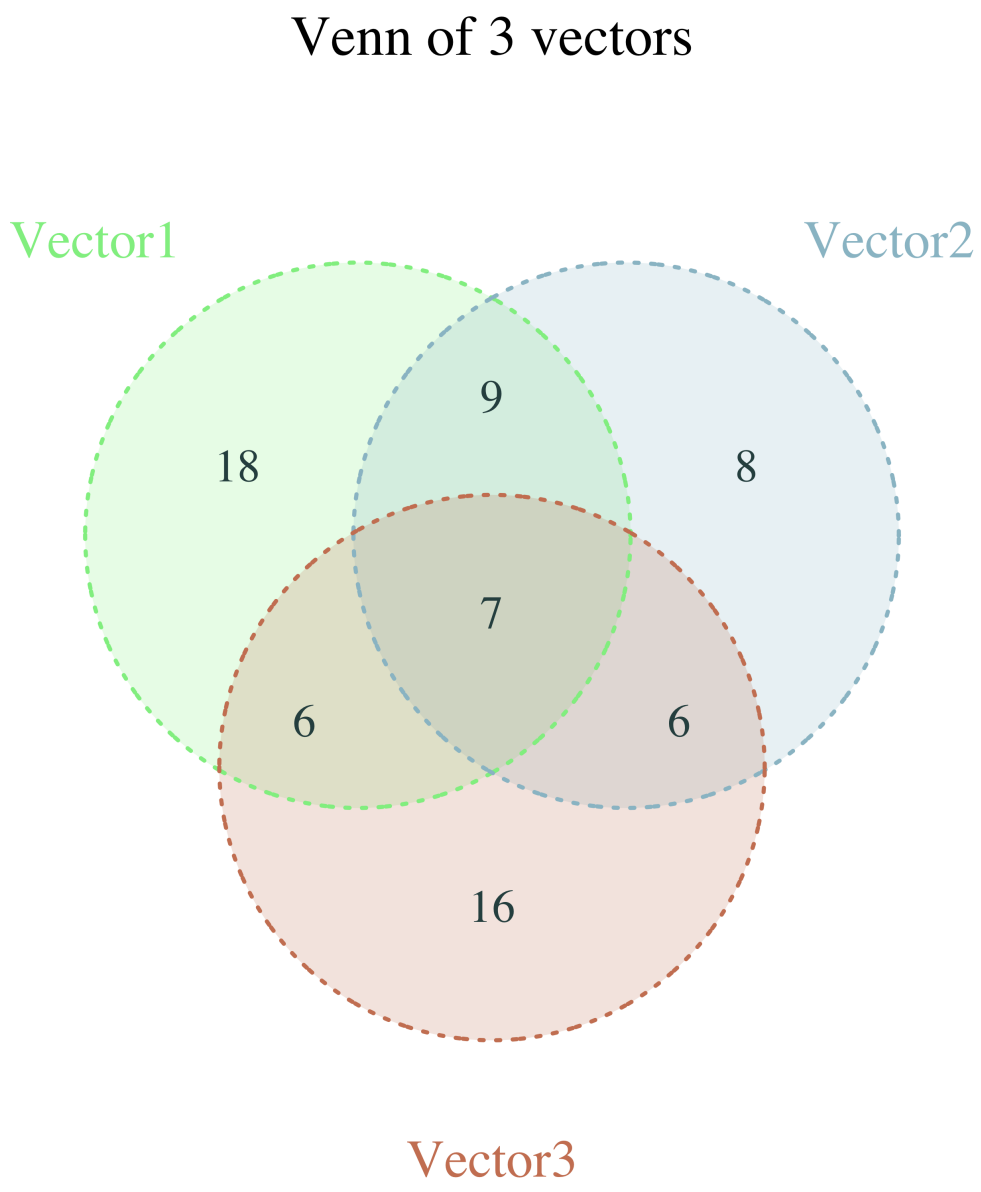
Let's try a three-set Venn:

```
# Additional vector
vec3 <- sample(seq(0,100, 1), 35)
# new list containing the 5 vectors
mylist3 <- list(Vector1=vec1, Vector2=vec2, Vector3=vec3)
# Produce Venn
venn.diagram(mylist3,
             filename="VennDiagram_3.tiff")
```


Vector1
Vector2
9
18
8
7
6
6
16
Vector3

Exercise 8

Try to produce the following plot:



Using the following paramaters from venn.diagram:

- Control the coloring:
  - cat.col
  - label.col
  - col
  - fill
  - alpha
- Control the text sizes:
  - main.cex
  - cat.cex

  - cex

- And more:
  - margin
  - lty
  - cat.dist
  - main

END