3
Scales
The diamonds data is part of ggplot2

```r
glimpse(diamonds)
```

```r
## Rows: 53,940
## Columns: 10
## $ carat <dbl> 0.23, 0.21, 0.23, 0.29, 0.31, 0.24, 0.24, 0.26, 0.22, 0.23, 0....
## $ cut <ord> Ideal, Premium, Good, Premium, Good, Very Good, Very Good, Ver...
## $ clarity <ord> SI2, SI1, VS1, VS2, SI2, VVS2, VVS1, SI1, VS2, VS1, SI1, VS1, ...
## $ depth <dbl> 61.5, 59.8, 56.9, 62.4, 63.3, 62.8, 62.3, 61.9, 65.1, 59.4, 64...
## $ table <dbl> 55, 61, 65, 58, 58, 57, 57, 55, 61, 61, 55, 56, 61, 54, 62, 58...
## $ price <int> 326, 326, 327, 334, 335, 336, 336, 337, 337, 338, 339, 340, 34...
## $ x <dbl> 3.95, 3.89, 4.05, 4.20, 4.34, 3.94, 3.95, 4.07, 3.87, 4.00, 4....
## $ y <dbl> 3.98, 3.84, 4.07, 4.23, 4.35, 3.96, 3.98, 4.11, 3.78, 4.05, 4....
## $ z <dbl> 2.43, 2.31, 2.31, 2.63, 2.75, 2.48, 2.47, 2.53, 2.49, 2.39, 2....
```
Scales control the mapping from data to aesthetics.

```
g <- ggplot(diamonds, aes(carat, price) ) + geom_hex()
```

```g + scale_y_continuous() + scale_x_continuous()
g + scale_x_reverse() + scale_y_continuous(trans="log10")
g + scale_y_log10() + scale_x_sqrt()
```
<table>
<thead>
<tr>
<th>scales</th>
</tr>
</thead>
<tbody>
<tr>
<td>scale_alpha, scale_alpha_continuous, scale_alpha_binned, scale_alpha_discrete, scale_alpha_ordinal, scale_alpha_datetime, scale_alpha_date</td>
</tr>
<tr>
<td>Alpha transparency scales</td>
</tr>
<tr>
<td>scale_x_binned, scale_y_binned</td>
</tr>
<tr>
<td>Positional scales for binning continuous data (x &amp; y)</td>
</tr>
<tr>
<td>scale_colour_brewer, scale_fill_brewer, scale_colour_distiller, scale_fill_distiller, scale_colour_fermenter, scale_fill_fermenter, scale_color_brewer, scale_color_distiller, scale_color_fermenter</td>
</tr>
<tr>
<td>Sequential, diverging and qualitative colour scales from ColorBrewer</td>
</tr>
<tr>
<td>scaleColour_continuous, scaleFill_continuous, scaleColour_binned, scaleFill_binned, scaleColour_continuous, scaleColour_binned</td>
</tr>
<tr>
<td>Continuous and binned colour scales</td>
</tr>
</tbody>
</table>
Guide: an axis or a legend

- The scale creates a `guide`: an `axis` or `legend`.
- So to modify these you generally use `scale_*` or other handy functions (guides, labs, xlab, ylab and so on).
g +
  scale_y_continuous(name = "Price",
                   breaks = c(0, 10000),
                   labels = c("0", "More\n than\n 10K")) +
  geom_hline(yintercept = 10000, color = "red", size = 2)
g +
  scale_y_continuous(
    label = scales::dollar_format()
  )
g +

scale_fill_continuous(
  breaks = c(0, 10, 100, 1000, 4000),
  trans = "log10"
)
g +
  scale_fill_continuous(
    guide = "none"
  )
Alternative control of guides

g +

ylab("Price") + # Changes the y axis label
labs(x = "Carat", # Changes the x axis label
    fill = "Count") # Changes the legend name

g + guides(fill = "none") # remove the legend

Qualitative palettes

designed for categorical variable with no particular ordering

colorspace::hcl_palettes("Qualitative", plot = "TRUE", n = 7)
Sequential palettes

designed for ordered categorical variable or number going from low to high (or vice-versa)

colourspace::hcl_palettes("Sequential", plot = "TRUE", n = 7)
Diverging palettes

designed for ordered categorical variable or number going from low to high (or vice-versa) with a neutral value in between

colorspace::hcl_palettes("Diverging", plot = "TRUE", n = 7)
RGB color space
made for screen projection

Code adapted from https://github.com/Golobro/rgbcolorslder
HCL color space
made for human visual system

Color conversion using https://github.com/gka/chroma.js
Interactively choose/create a palette using the HCL color space.

```r
library(colorspace)
hcl_wizard() # OR choose_palette()
```
Choose your palette > Export > R > Copy the command

If you use R the preferred way to handle color palettes is to use `choose_palette()` or `hclwizard()` on your local machine. Both graphical user interfaces return an R color palette function. However, you can also use the function call below to use the current palette in your R scripts.

```
## Custom color palette
sequential_hcl(n = 7, h = c(300, 200), c = c(60, NA, 0), l = c(25, 95), power = c(0.7, 1.3), register = )
```

Custom color palettes can also be registered to be able to call custom palettes by name. If the optional argument register = "Custom-Palette" is set (where "Custom-Palette" is the name of your new palette) the palette will be added to the list of available color palettes. Existing palettes can also be overruled. Note that the graphical interfaces (choose_palette()) will not use custom palettes. These register-calls can also be added to your local ~/.Rprofile.

```
## Register custom color palette
colorspace::sequential_hcl(n = 7, h = c(300, 200), c = c(60, NA, 0), l = c(25, 95), power = c(0.7, 1.3), register = "Custom-Palette")
```
Registering your own palette

```r
library(colorspace)
# register your palette
sequential_hcl(n = 7,
    h = c(300, 200),
    c = c(60, 0),
    l = c(25, 95),
    power = c(2.1, 0.8),
    register = "my-set")
# now generate from your palette
sequential_hcl(n = 3,
    palette = "my-set")

## [1] "#6B0077" "#7C8393" "#F1F1F1"
```

## Combining with ggplot:

```r
ggplot(penguins,
    aes(bill_length_mm, fill = species)) +
    geom_density(alpha = 0.6) +
    # notice here you don't need to specify the n!
    scale_fill_discrete_sequential(palette = "my-set")
```

```
```
Manually selecting colors

```r
# Manually selecting colors for the density plot

# Load the penguins dataset
library(tidyverse)
library(ggplot2)
penguins <- ggplot2:::ggplot_data(penguins)

# Create the ggplot object

g <- ggplot(penguins, aes(bill_length_mm, fill = species)) +
   geom_density(alpha = 0.6) +
   scale_fill_manual(
     breaks = c("Adelie", "Chinstrap", "Gentoo"),  # optional but makes it more robust
     values = c("darkorange", "purple", "cyan4"))

g
```
Check that it's colour blind friendly!

cols <- c("darkorange", "purple", "cyan4")

g # original

g + scale_fill_manual(values = deutan(cols))

g + scale_fill_manual(values = protan(cols))

g + scale_fill_manual(values = deutan(cols))

g + scale_fill_manual(values = tritan(cols))
Open part1-exercise-03.Rmd

15:00
devtools::session_info()

---

Session info 🎨 🌍 🎳

## Settings

<table>
<thead>
<tr>
<th>setting</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>R version 4.1.2 (2021-11-01)</td>
</tr>
<tr>
<td>os</td>
<td>macOS Big Sur 10.16</td>
</tr>
<tr>
<td>system</td>
<td>x86_64, darwin17.0</td>
</tr>
<tr>
<td>ui</td>
<td>X11</td>
</tr>
<tr>
<td>language</td>
<td>(EN)</td>
</tr>
<tr>
<td>collate</td>
<td>en_AU.UTF-8</td>
</tr>
<tr>
<td>ctype</td>
<td>en_AU.UTF-8</td>
</tr>
<tr>
<td>tz</td>
<td>Australia/Melbourne</td>
</tr>
<tr>
<td>date</td>
<td>2021-11-30</td>
</tr>
</tbody>
</table>

---

These slides are licensed under [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/)