ESCI upf.
International Business

T2.1 | Basic tools for data visualization

ggplot2

Marta Coronado Zamora
25 September 2024

Keep in touch

Theory lessons

Marta Coronado Zamora	Jose F. Sánchez
marta.coronado@prof.esci.upf.edu	jose.sanchez@prof.esci.upf.edu
@geneticament	@JFSanchezBioinf
Institut Botànic de Barcelona (CSIC-CMCNB)	Germans Trias i Pujol Research Institute (IGTP)

Practical lessons

Adrià Auladell
adria.auladell@ibe.upf-csic.es
Institut de Biologia Evolutiva (UPF-CSIC)

Theory session dynamics

Content

- Theory
- Examples and short exercises in class (✎) - complete and submit to aul@-ESCI

  Bring your laptop to theory sessions!

Interactive R documents

`R code` can be executed within RStudio!

```
value ← 2
value + 3
```

```
## [1] 5
```

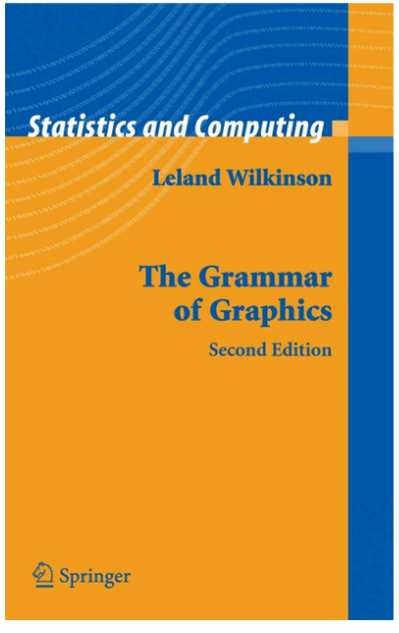
Get started!

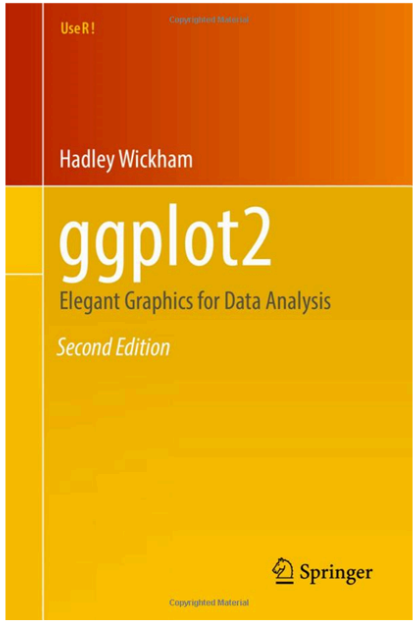
Graphics with `ggplot2`

The grammar of graphics

Original grammar

Wilkinson, Leland. The grammar of graphics. Springer Science & Business Media, 2006.





Adapted to R in the `ggplot2` package

Hadley Wickham. ggplot2: elegant graphics for data analysis. Springer, 2009.

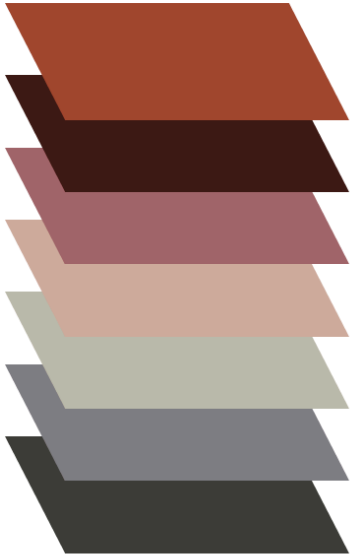
"The grammar tells us that a **statistical graphic is a mapping from data to aesthetic attributes** (colour, shape, size) **of geometric objects** (points, lines, bars). The plot may also contain **statistical transformations** of the data and is drawn on a specific **coordinate system**. **Facetting** can be used to generate the same plot for different subsets of the dataset. It is the combination of these independent components that make up a graphic."

Syntax

In `ggplot2` there are different components we can add to a plot:

```
ggplot(data = <DATA>,
    mapping = aes(<MAPPINGS>)) +

    <GEOM_FUNCTION>(stat = <STAT>,
    position = <POSITION>) +

    <SCALE_FUNCTION>() +

    <COORDINATE_FUNCTION>() +

    <FACET_FUNCTION>() +

    <THEME_FUNCTION>
```

Describes all the non-data ink	**Theme**
Rows and columns of sub-plots	**Facets**
Plotting space for the data	**Coordinates**
Statistical models and summaries	**Statistics**
Shapes used to represent the data	**Geometries**
Scales onto which data is mapped	**Aesthetics**
The actual variables to be plotted	**Data**

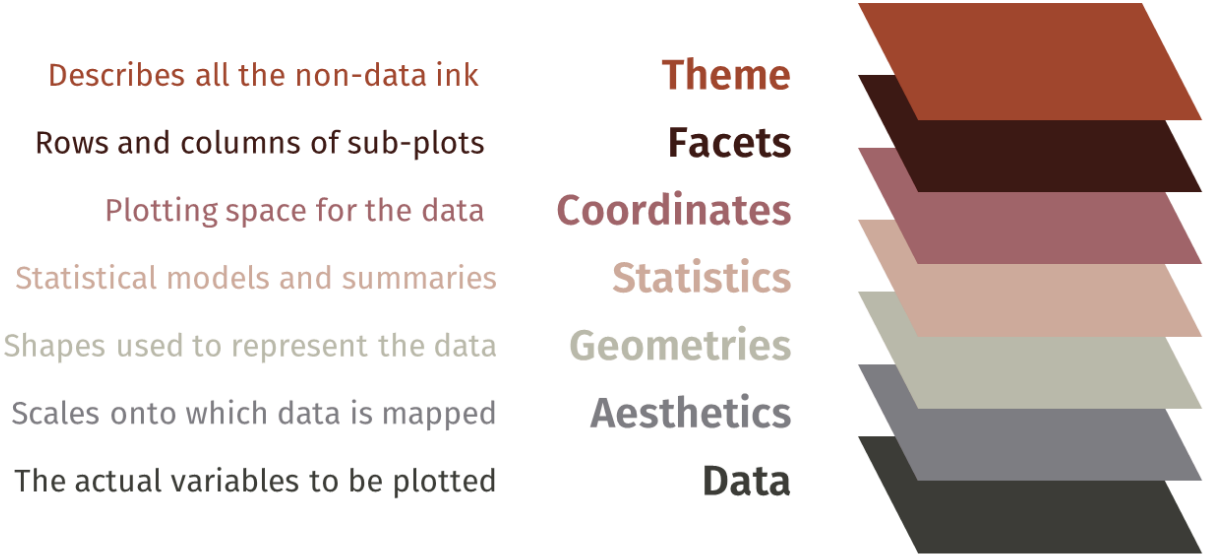


Grammar of Graphics:
A layered approach to elegant visuals

Components

The layered grammar defines a plot as a combination of:

- **Layers**
  - Data
  - Aesthetic mapping
  - Geometric objects
  - Statistical transformation
  - Position adjustment
- **Scales**
- **Coordinate system**
- **Faceting specification**
- **Theme** (not in the original grammar)



Grammar of Graphics:
A layered approach to elegant visuals

Layers

Layers are responsible for creating the objects that we perceive on the plot.

- **Data**
- **Aesthetic mapping**
- **Geometric objects**
- **Statistical transformation**
- **Position adjustment**

Syntax:

```
ggplot(data = data,  mapping = aes(x = var1, y = var2, colour = var3)) +
  layer(geom = "point", stat = "identity", position = "identity")
```

Layers: Data

The data layer specifies the data being plotted. Must be an **R data frame** object.

```
head(iris, 5)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
```

```
ggplot(data = iris)
```


We get a blank square because we have not added any other layers yet!

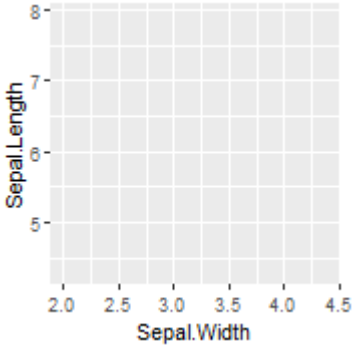
Layers: mapping data onto aesthetics

The `iris` data frame has different columns: **aesthetics**.

The aesthetic layer, or aes for short, specifies how we want to map our data onto the scales of the plot (such as the `x` and `y` coordinates).

In `ggplot2` the aesthetic layer is specified using the `aes()` function.

```
ggplot(data = iris, mapping = aes(x = Sepal.Width, y = Sepal.Length))
```



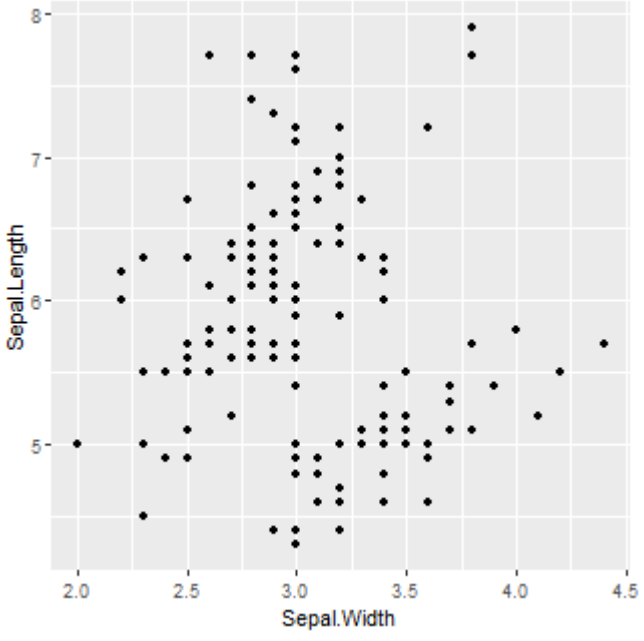
The graph shows variable and scales of `Sepal.Length` mapped onto the `x`-axis and `Sepal.Width` on the `y`-axis. What are we missing?

A **geom**!

Layers: mapping data onto aesthetics

The **geom** is the geometric object to use display the data.

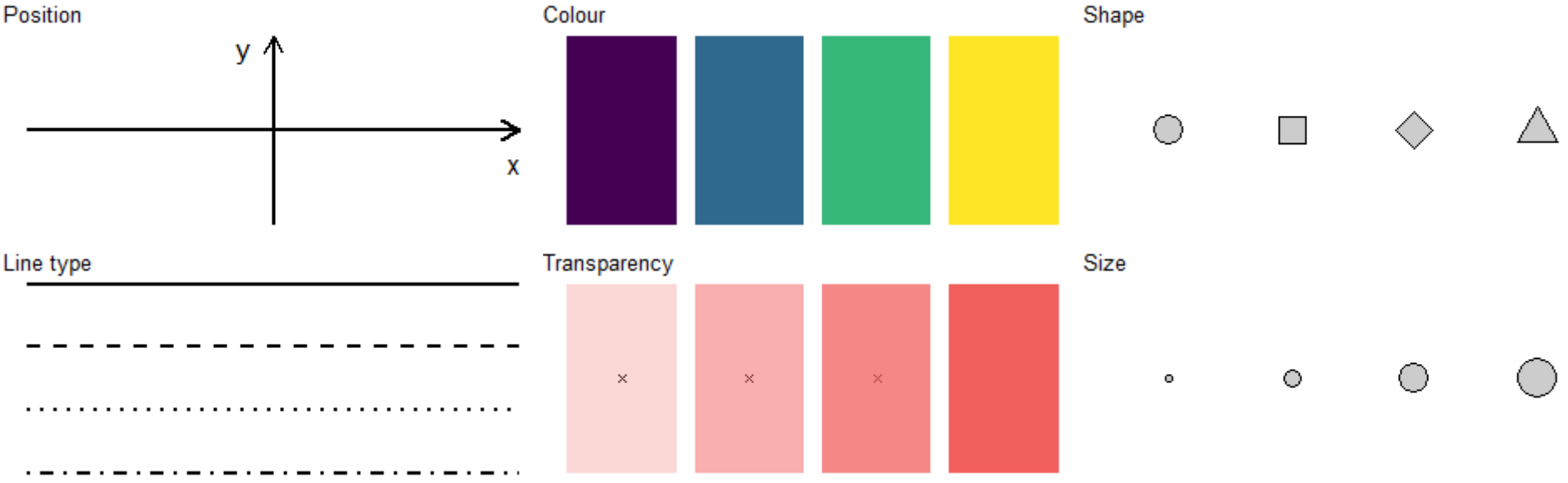
```
ggplot(data = iris, mapping = aes(x = Sepal.Width, y = Sepal.Length)) +
  geom_point()
```



Now we have a scatterplot of the relationship between `Sepal.Length` and `Sepal.Width`.

Layers: mapping data onto aesthetics

Main aesthetics types:

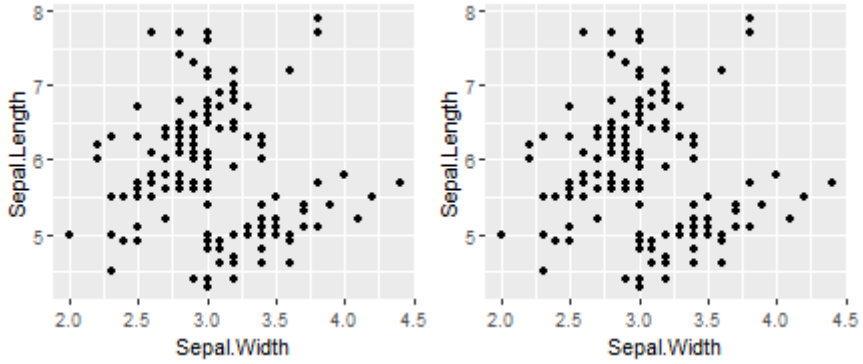


✎ **Exercise | Which aesthetic attributes can a continuous variable be mapped to? And a discrete variable?**

Layers: mapping data onto aesthetics

Data and mapping can be defined in the initial `ggplot()` call or in the layer (`geom_` or `stat_`)

```
# Default data and mapping used by the layer
ggplot(data = iris, mapping = aes(x = Sepal.Width, y = Sepal.Length)) +
  geom_point()
# No default defined, data and mapping in layer
ggplot() +
  geom_point(data = iris, mapping = aes(x = Sepal.Width, y = Sepal.Length))
```



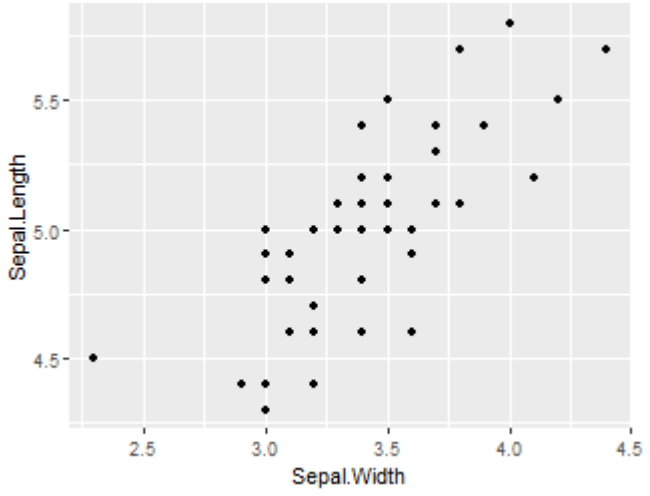
Exercise | With the code that generate the previous figure, experiment with the colour, size, transparency (`alpha`) and shape aesthetics.

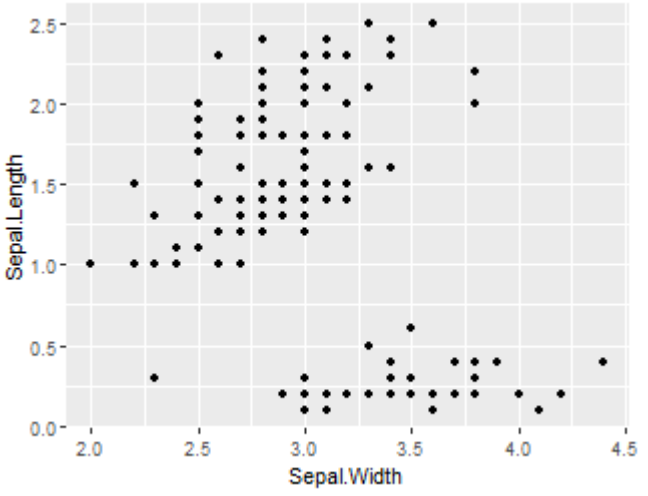
Answer:

Layers: mapping data onto aesthetics

You can override data or mapping in layer.

```
p ← ggplot(data = iris, mapping = aes(x = Sepal.Width, y = Sepal.Length))

# Override data in layer
small_iris ← iris[iris$Species %in% "setosa", ]
p + geom_point(data = small_iris)

# Override mapping in layer (or remove y = NULL or add colour = Species)
p + geom_point(aes(y = Petal.Width))
```





ℹ Note that even though we are mapping `Petal.Width` the graph stills shows `Sepal.Length` in the `y`-axis.

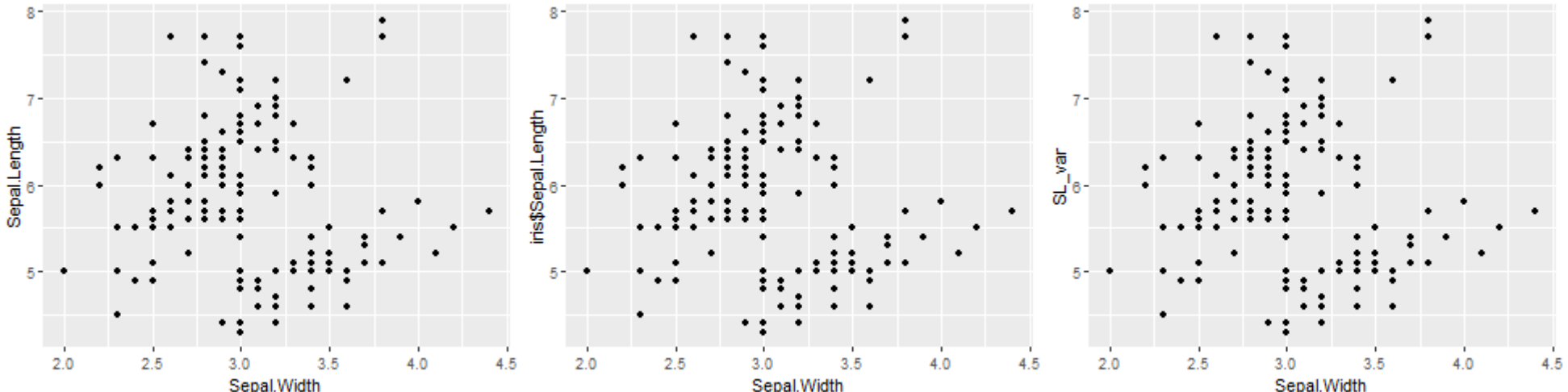
Layers: mapping data onto aesthetics

All variables used in the plot should be in the data.

```
# Correct
ggplot(iris, aes(x = Sepal.Width, y = Sepal.Length)) + geom_point()

# Not recommended
ggplot(iris, aes(x = Sepal.Width, y = iris$Sepal.Length)) + geom_point()
```

```
## Warning: Use of `iris$Sepal.Length` is discouraged.
## i Use `Sepal.Length` instead.
```

```
SL_var ← iris$Sepal.Length
ggplot(iris, aes(x = Sepal.Width, y = SL_var)) + geom_point()
```



Layers: mapping data onto aesthetics

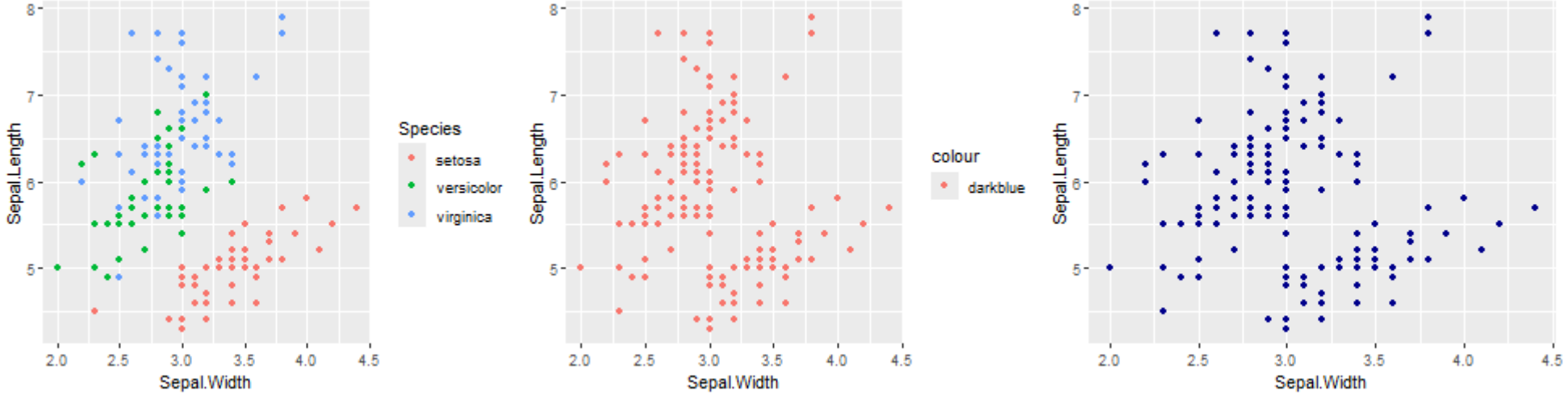
Exercise | Simplify the following code.

```
ggplot(mpg) +
  geom_point(aes(mpg$displ, mpg$hwy))

ggplot() +
  geom_point(mapping = aes(y = hwy, x = cty), data = mpg) +
  geom_smooth(data = mpg, mapping = aes(cty, hwy))

ggplot(diamonds, aes(carat, price)) +
  geom_point(aes(log(brainwt), log(bodywt)), data = msleep)
```

Layers: mapping data onto aesthetics

Mapping (variable) vs. setting (constant)

```
p ← ggplot(iris, aes(Sepal.Width, Sepal.Length))

p + geom_point(aes(colour = Species)) # Map Species to colour
p + geom_point(aes(colour = "darkblue")) # Map "darkblue" to colour
p + geom_point(colour = "darkblue") # Set colour to "darkblue"
```

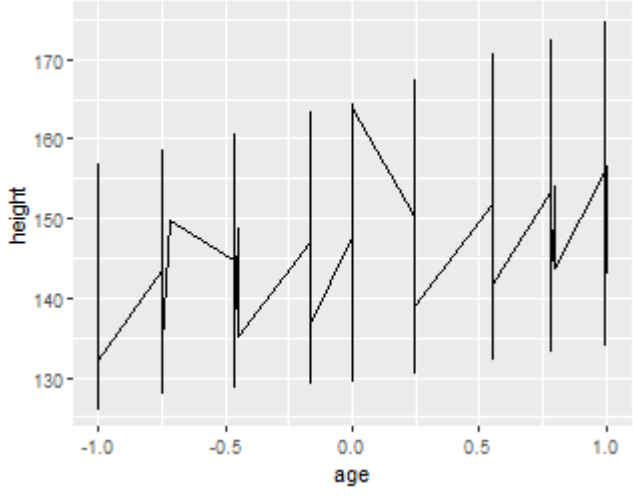


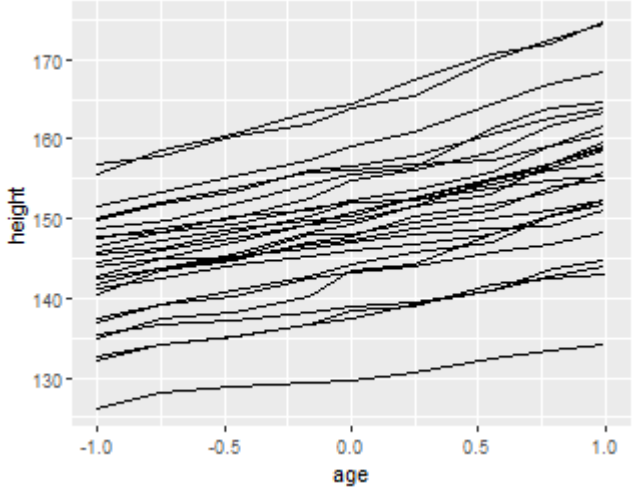
Layers: mapping data onto aesthetics

Group aesthetic: by default it is the combination of discrete variables (except position).

Multiple groups, one aesthetic:

```
ggplot(nlme::Oxboys, aes(age, height)) +
  geom_line()

ggplot(nlme::Oxboys, aes(age, height, group = Subject)) + # or colour = Subject
  geom_line()
```



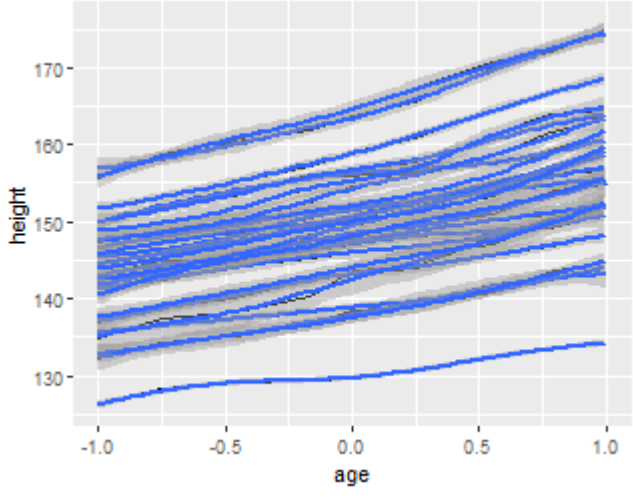


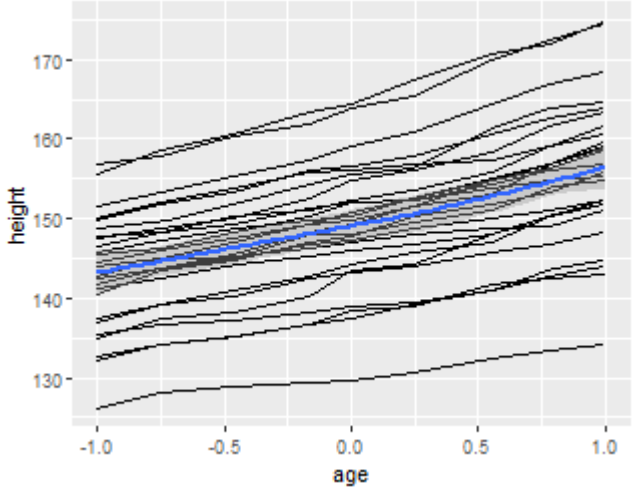
Layers: mapping data onto aesthetics

Group aesthetic: by default it is the combination of discrete variables (except position).

Different groups on different layers:

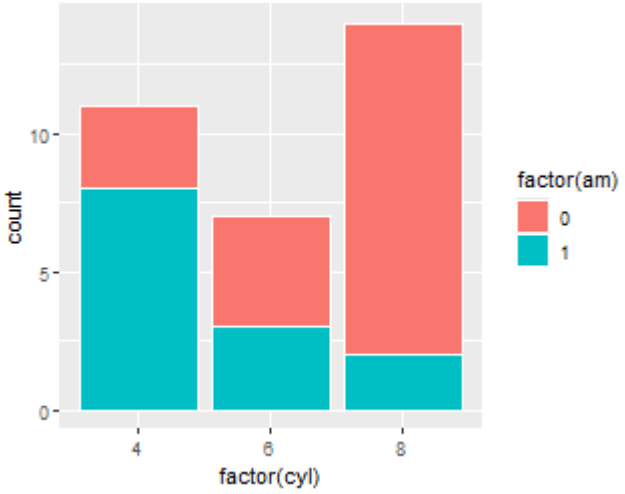
```
ggplot(nlme::Oxboys, aes(age, height, group = Subject)) +
  geom_line() + geom_smooth(aes(group = Subject))

ggplot(nlme::Oxboys, aes(age, height, group = Subject)) +
  geom_line() + geom_smooth(aes(group = 1)) # o group = NULL
```





Layers: mapping data onto aesthetics

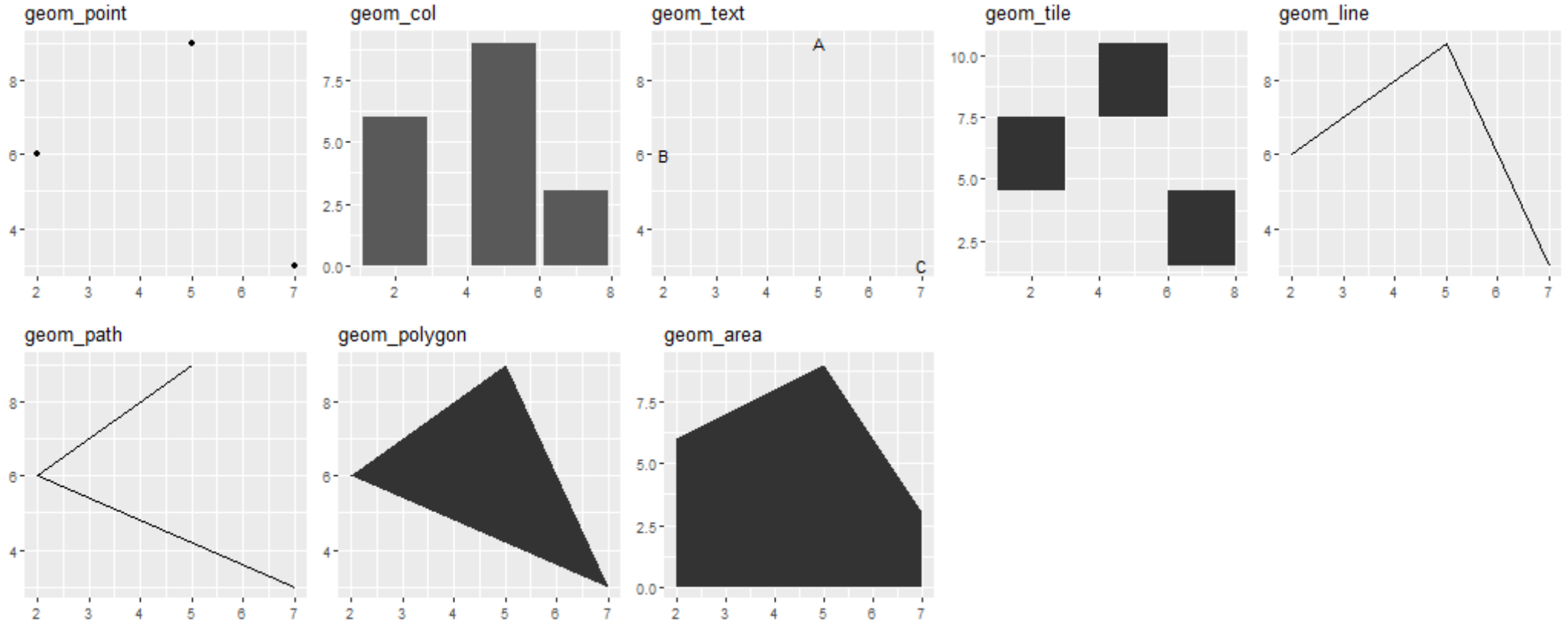
✎ **Exercise | Are the observations grouped in the following plot? If so, what variables are used for the grouping?**



Layers: mapping data onto aesthetics

- Aesthetic attributes:
  - **Position**: x, y, xmin, xmax, ymin, ymax, xend, yend
  - **Colour**: colour, fill, alpha
  - **Differentiation**: shape, size, linetype
  - **Grouping**: group
- Each geom understands a different set of aesthetics
- Each geom requires some aesthetics (`?geom_*`)

Layers: geometric objects



✎ **Exercise | Which geoms are collective? (multiple observations share a geometrical object)**

Layers: geometric objects

- Full list of geom_* functions:
  - ggplot2 reference
  - ggplot2 cheat sheet, sorted by type

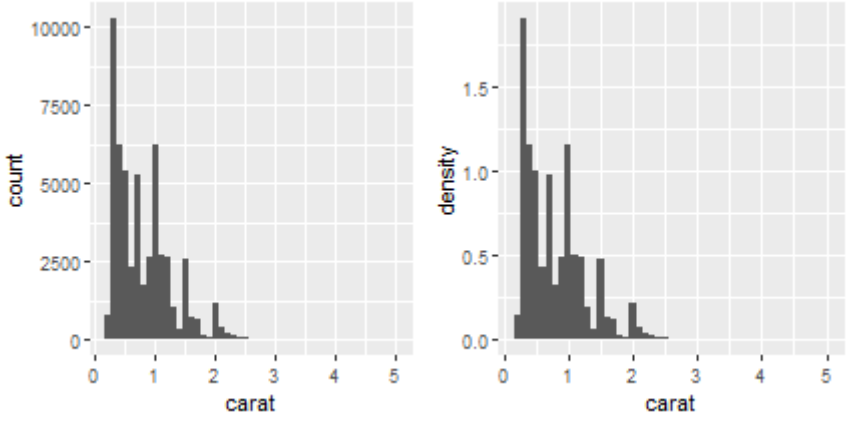
Layers: statistical transformation

The statistics layer allows you plot statistical values calculated from the data. Transforms the data, typically by summarising it in some manner.

A stat takes a dataset as input and returns a dataset as output, and so a stat can add new variables to the original dataset.

dataset → statistical transformation → dataset

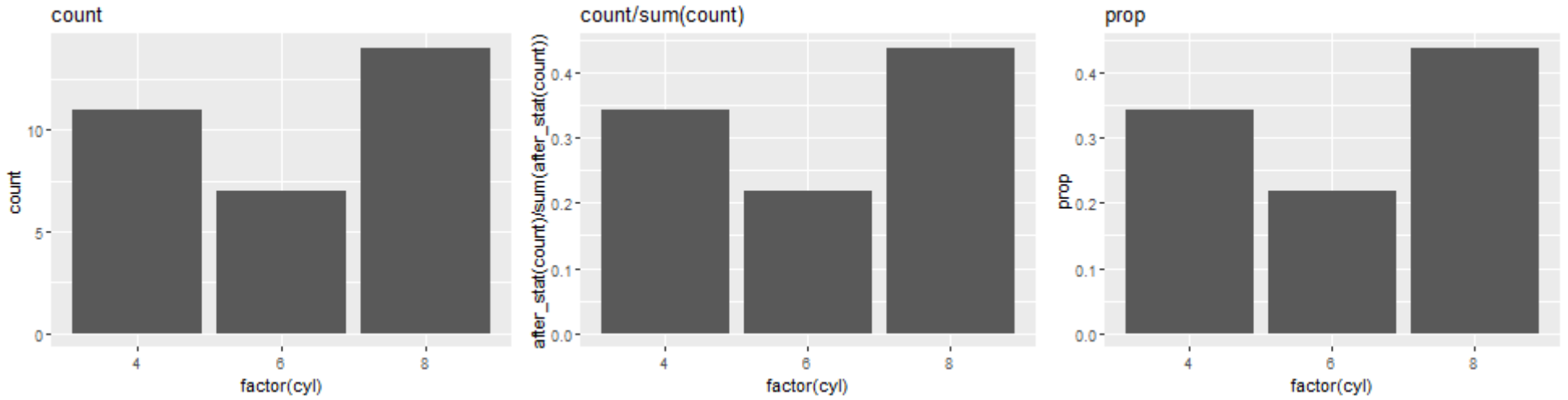
```
ggplot(diamonds, aes(carat)) + geom_histogram(aes(y=after_stat(count)), binwidth = 0.1)
ggplot(diamonds, aes(carat)) + geom_histogram(aes(y=after_stat(density)), binwidth = 0.1)
```



New variables are accessed with `after_stat(var)` notation or `stat()` (v3.4.0+)

Layers: statistical transformation

```
p ← ggplot(mtcars, aes(x = factor(cyl)))

p + geom_bar() + labs(title = "count") # uses stat = "bin" by default
p + geom_bar(aes(y = after_stat(count)/sum(after_stat(count)))) +
  labs(title = "count/sum(count)")
# p + geom_bar(aes(y = stat(count/sum(count))))
p + geom_bar(aes(y = after_stat(prop), group = 1)) + labs(title = "prop")
```



ℹ Note that we need to specify `group = 1` (or `group = NULL`), otherwise the proportion are calculated within each group and therefore all proportions would be 1.

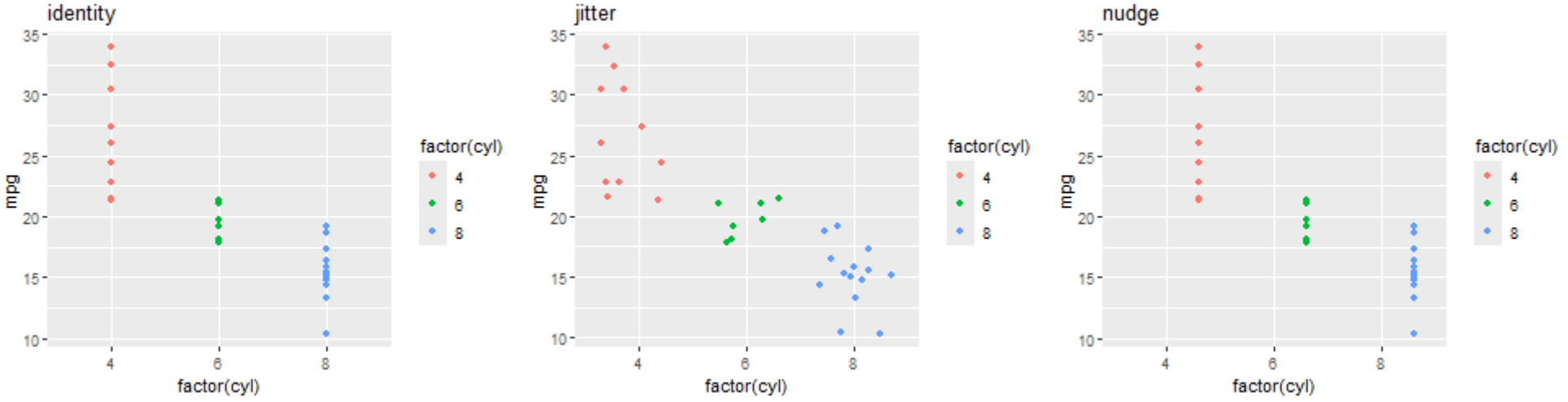
Layers: statistical transformation

- Full list of stat_* functions:
  - `ggplot2` reference: geoms and stats
  - `ggplot2` cheat sheet
- New variables from statistical transformations:
  - `stat_count`: count, prop
  - `stat_bin` or `stat_bin2d`: count, density
  - `stat_boxplot`: width, ymin, ymax, middle
  - `stat_smooth`: y, ymin, ymax, se
  - `stat_summary`: value

Layers: position adjustment

Position adjustments apply minor tweaks to the position of elements within a layer.

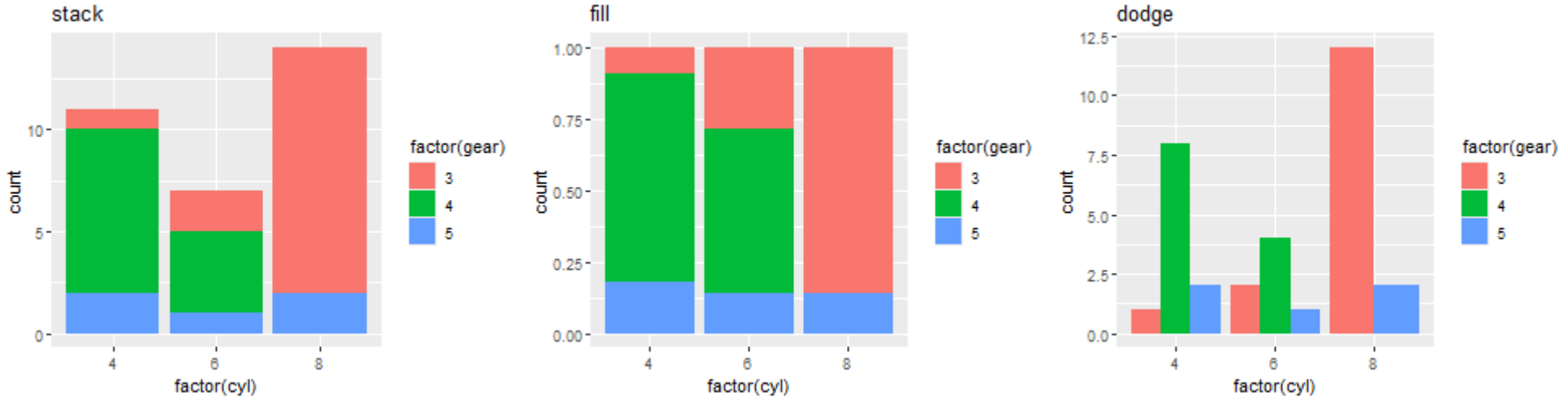
```
p ← ggplot(mtcars, aes(x = factor(cyl), y = mpg, colour = factor(cyl)))

p + geom_point() + labs(title = "identity") # default point position
p + geom_point(position = "jitter") + labs(title = "jitter")
p + geom_point(position = position_nudge(x = 0.3)) + labs(title = "nudge")
```



Layers: position adjustment

Position adjustments apply minor tweaks to the position of elements within a layer.

```
b ← ggplot(mtcars, aes(x = factor(cyl), fill = factor(gear)))

b + geom_bar() + labs(title="stack") # default bar position
b + geom_bar(position = "fill") + labs(title="fill")
b + geom_bar(position = "dodge") + labs(title="dodge")
```



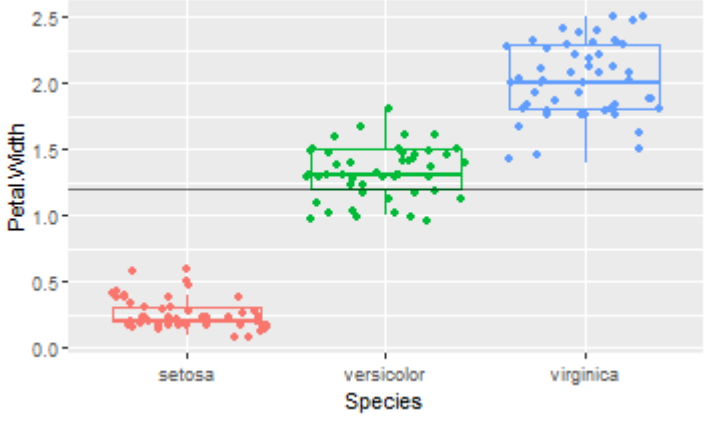
Layers: data + aes + geom + stat + position

fun	aes	geom	stat	position
geom_point	x, y, alpha, colour, size, shape...	point	identity	identity
geom_line	x, y, linetype...	line	identity	identity
geom_ribbon	x, ymin, ymax...	ribbon	identity	identity
geom_col	x, y, ...	col	identity	stack
geom_text	x, y, label, angle, fontface...	text	identity	identity
stat_ecdf	x, colour, linetype	step	ecdf	identity

Layers: type of layers and annotations

We can add additional **metadata**, context and annotations, that help to give meaning to the raw data or highlight important features.

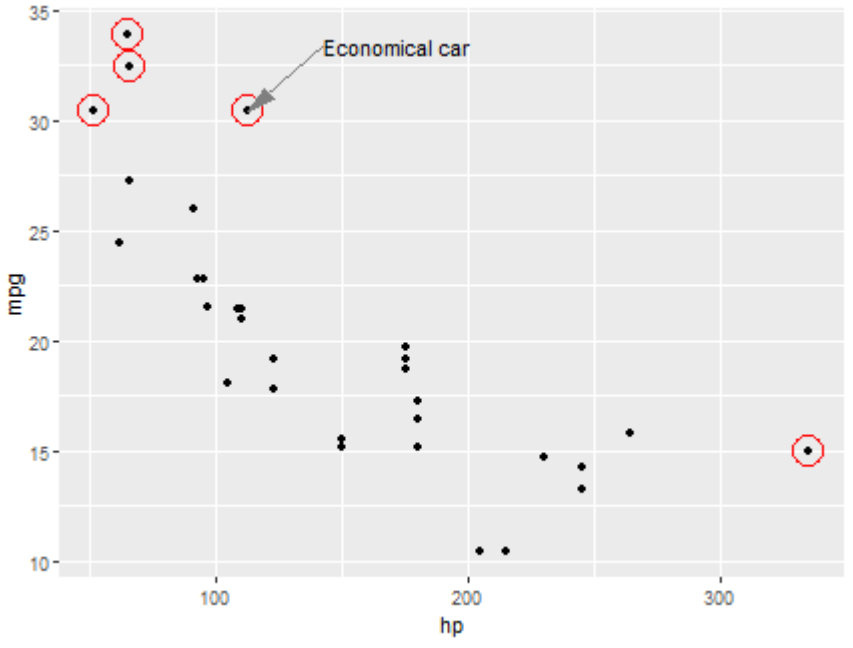
```
ggplot(iris, aes(x = Species, y = Petal.Width, colour = Species)) +
  # Show data
  geom_jitter(show.legend = FALSE) +

  # Summarize
  geom_boxplot(fill = NA, show.legend = FALSE) +

  # Annotate
  geom_hline(yintercept = mean(iris$Petal.Width), alpha = 0.5)
```



Layers: type of layers and annotations

- Common `geom_*` functions with metadata:
  - `geom_text()`, `geom_rect()`, `geom_point()`
  - `geom_line()`, `geom_path()`, `geom_segment()`

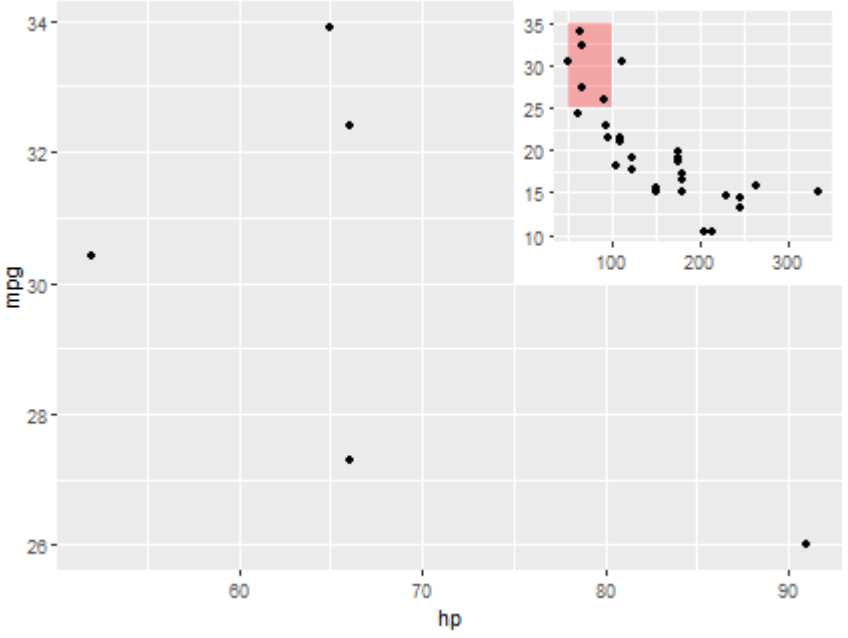
```
p ← ggplot(mtcars, aes(hp, mpg)) + geom_point(
p +
  # Add red cercles
  geom_point(data = mtcars[mtcars$hp > 300 | mtc
             colour = "red", shape = 1, size = 7
  # Draw an arrow
  geom_segment(data = mtcars[mtcars$hp > 100 & m
               aes(xend = hp + 30, yend = mpg +
               arrow = arrow(ends = "first", typ
               colour = "gray50") +
  # Add text
  geom_text(data = mtcars[mtcars$hp > 100 & mtca
            aes(label = "Economical car"), posit
```



Layers: type of layers and annotations

- Special layers that act as normal geoms if parameters are within `aes()` or as annotations if outside.
  - `geom_vline()`, `geom_hline()`, `geom_abline()`
- Special layers that don't inherit global settings
  - `annotate()`, `annotation_custom()`

```
# Represent mpg vs. hp and annotate a subset of
p_overview ← ggplot(mtcars, aes(hp, mpg)) +
  annotate(geom = "rect", xmin = 50, ymin = 25,
  geom_point() +
  labs(x = NULL, y = NULL)
# Represent the subset of points
p_subset ← ggplot(mtcars[mtcars$hp <100 & mtcar
  geom_point()
# Add an inset with the full data
p_subset +
  annotation_custom(ggplotGrob(p_overview), xmir
```



Layers: type of layers and annotations

Exercise | Exercise: what type of annotation is the most appropriate for the following cases:

A. To highlight some points

B. To add the expected linear trend

C. To write some text

Scales

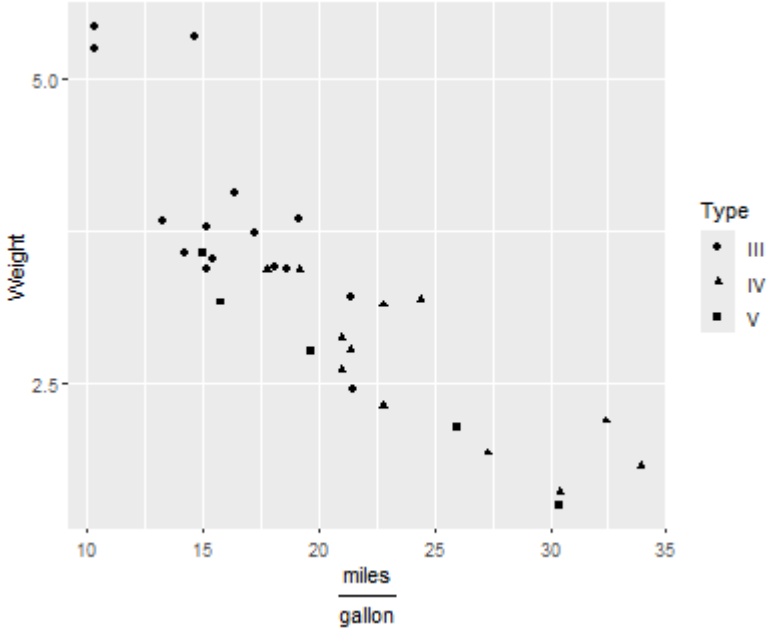
Scales control the mapping from data to aesthetics. They take your data and turn it into something that you can perceive visually: e.g., size, colour, position or shape.

- Types of scales
  - Axis: `x` and `y` position
  - Legend: the other aesthetics (fill, alpha, shape...)
- Default scales
  - `scale_<aes>_continuous()`: map continious values to visual values
  - `scale_<aes>_discrete()`: map discrete values to visual values
- Manual scales for discrete variables
  - `scale_<aes>_manual(values = c(1, 2, 4))`: map pdiscrete values to manualy chosen visual values

Scales

In all scales you can define: `name`, `breaks` and `labels`.

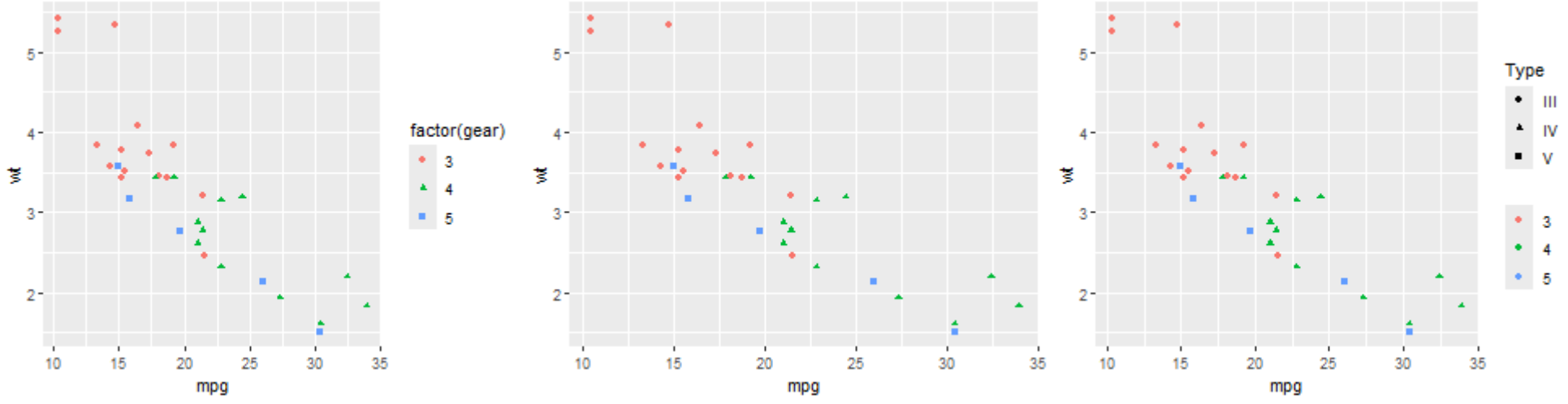
```
ggplot(mtcars, aes(x = mpg, y = wt, shape = factor(gear))) +
  geom_point() +
  scale_y_continuous(name = "Weight", breaks = c(2.5,5)) +
  scale_x_continuous(name = quote(frac(miles, gallon))) +
  scale_shape_discrete(name = "Type", labels = c("III","IV","V"))
```



Scales: legends

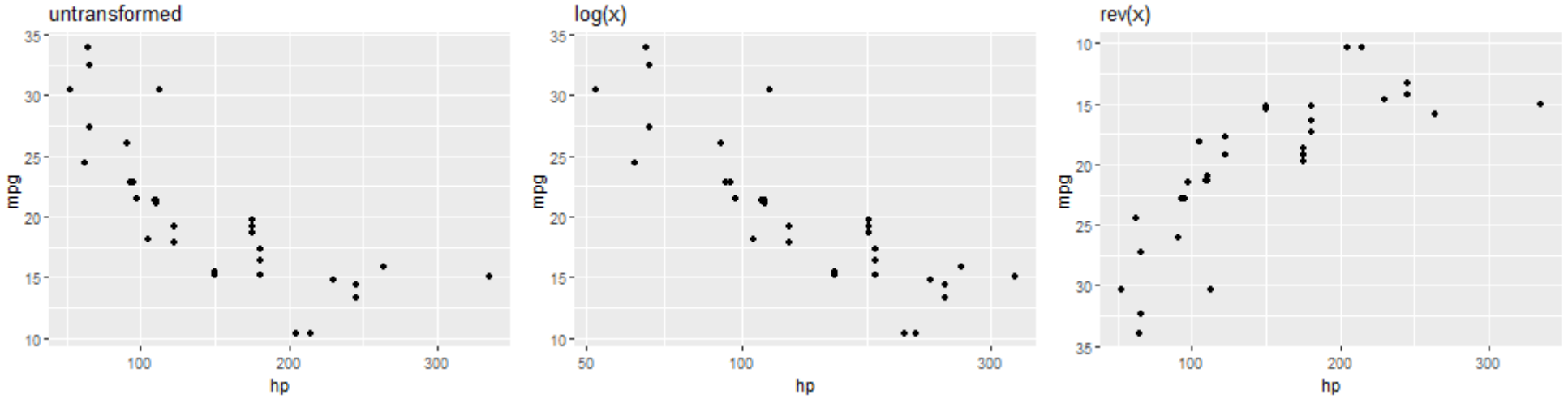
- Can be turned on/off
- Need the same name, breaks and labels to be merged

```
p ← ggplot(mtcars, aes(x = mpg, y = wt, shape = factor(gear), colour = factor(gear)))
p + geom_point()
p + geom_point(show.legend = FALSE)
p + geom_point() +
  scale_shape_discrete(name = "Type", labels = c("III","IV","V")) +
  scale_colour_discrete(name = NULL)
```

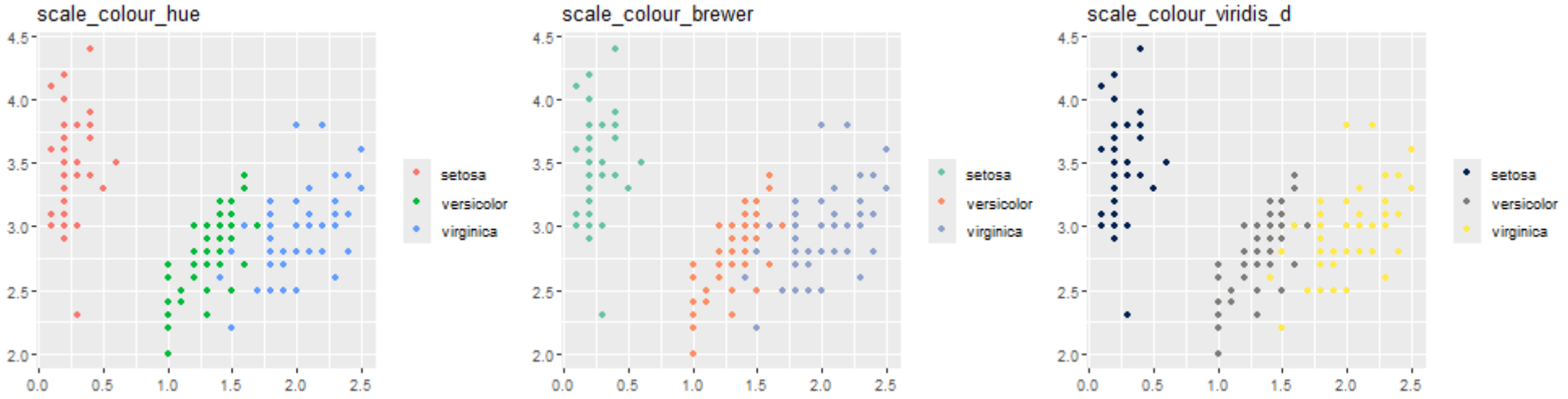


Scales: continuous axes

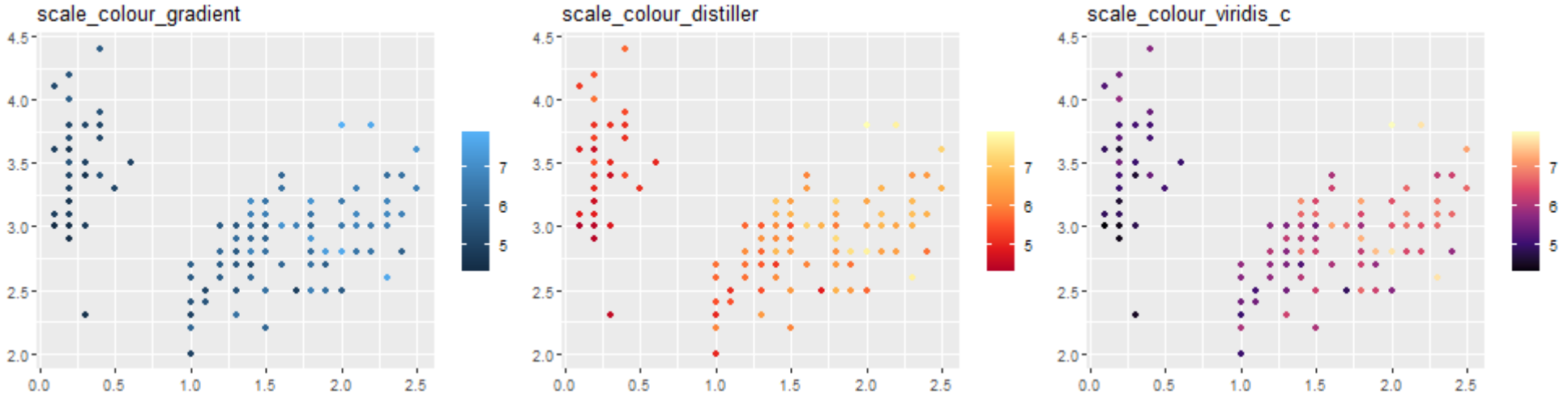
```
p ← ggplot(mtcars, aes(hp, mpg)) +
  geom_point()
p + labs(title = "untransformed")
p + scale_x_continuous(trans = "log10") + labs(title = "log(x)")
p + scale_y_reverse() + labs(title = "rev(x)")
```



Scales: colour scales for discrete variables

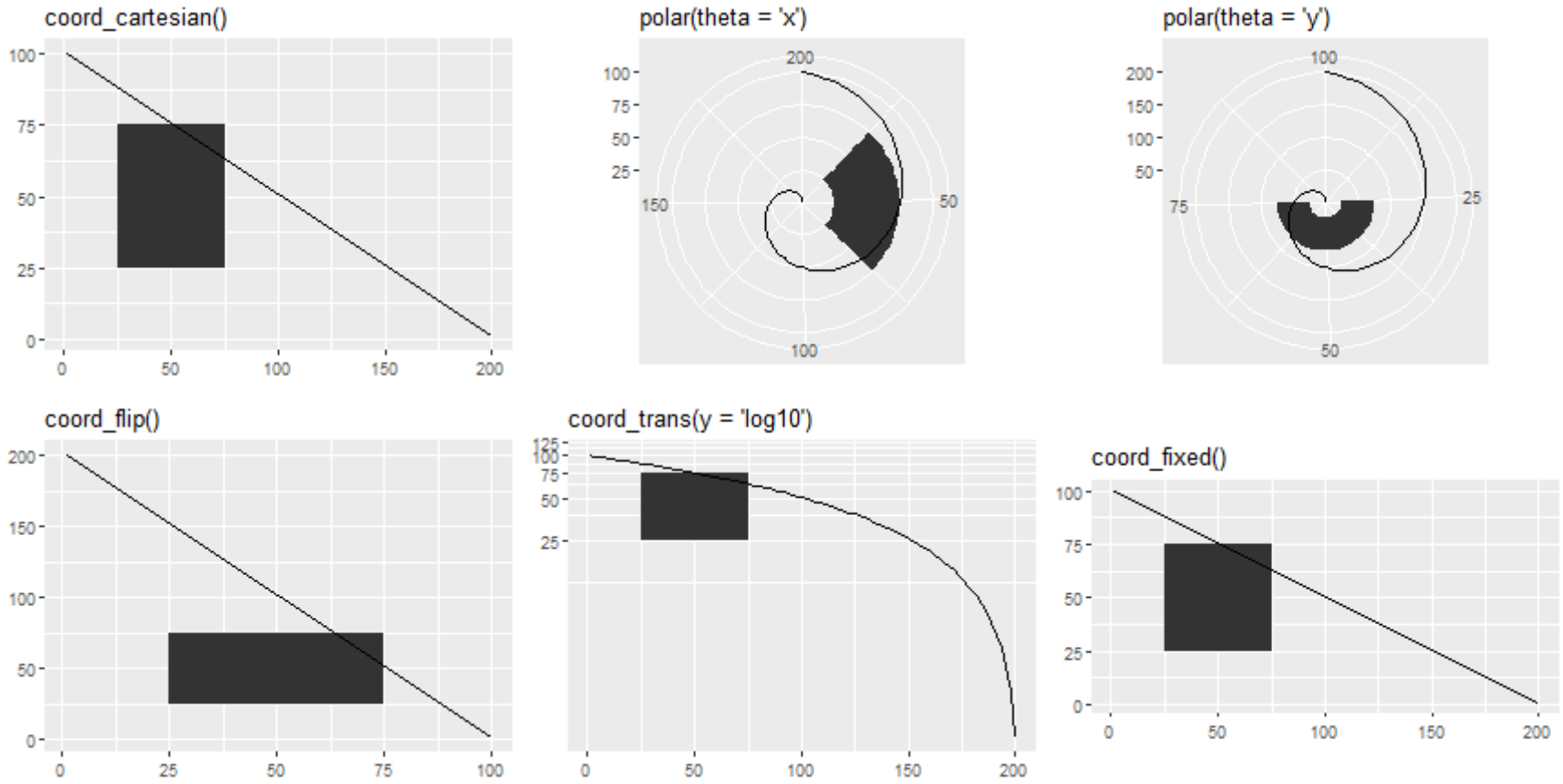


Scales: colour scales for continuous variables



Coordinate system

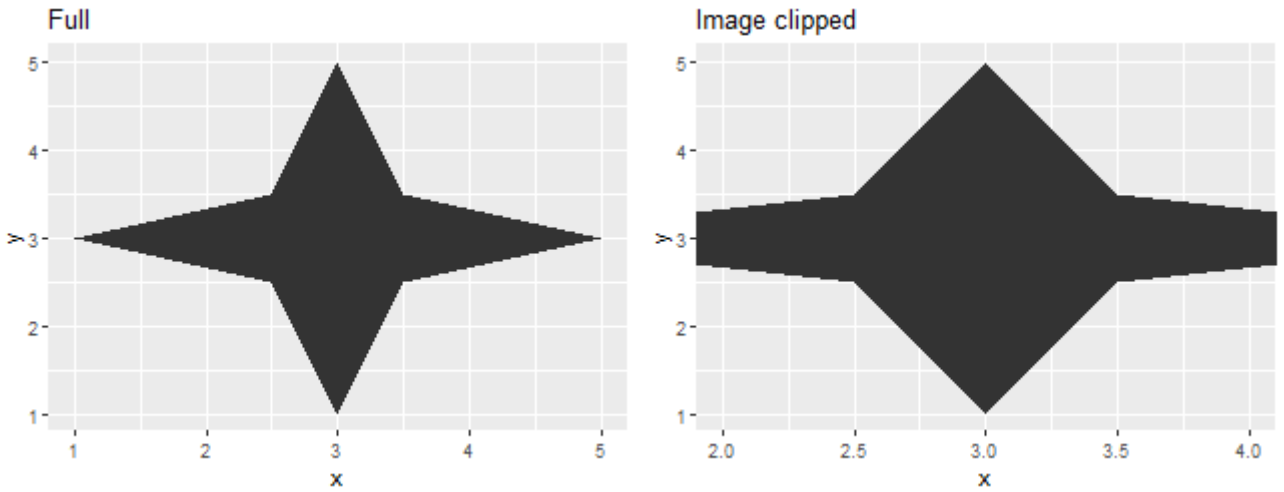
The coordinate component allows you to adjust the `x` and `y` coordinates.



Coordinate system

Zoom

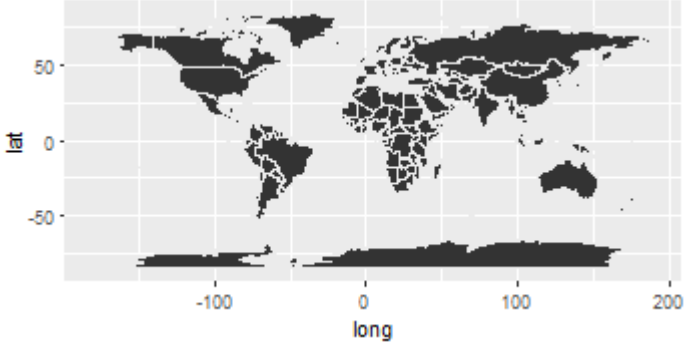
```
p ← ggplot(data = data.frame(x = c(1,2.5,3,3.5,5,3.5,3,2.5,1),
                             y = c(3,3.5,5,3.5,3,2.5,1,2.5,3)),
           aes(x, y)) +
  geom_polygon()
p + labs(title = "Full")
p + coord_cartesian(xlim = c(2,4)) + labs(title = "Image clipped")
```



Coordinate system

Maps

```
# install.packages("maps"); library(maps)
if(require("maps")){
  worldmap ← map_data(map = "world")
  ggplot(worldmap, aes(x = long, y = lat, group = group)) +
    geom_polygon(fill = "gray20", colour = "gray92") +
    coord_quickmap()
}
```

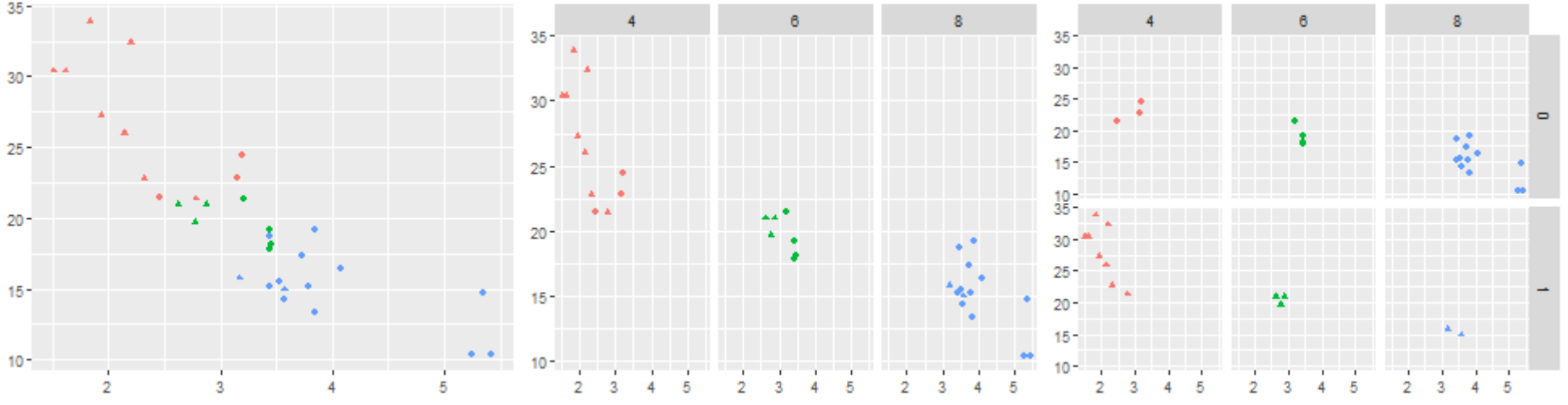


- `coord_map` can use projections from package `mapproj` (needs to be installed)

Facets

Faceting is a mechanism for automatically laying out multiple plots on a page. It splits the data into subsets, and then plots each subset into a different panel on the page. Such plots are often called small multiples.

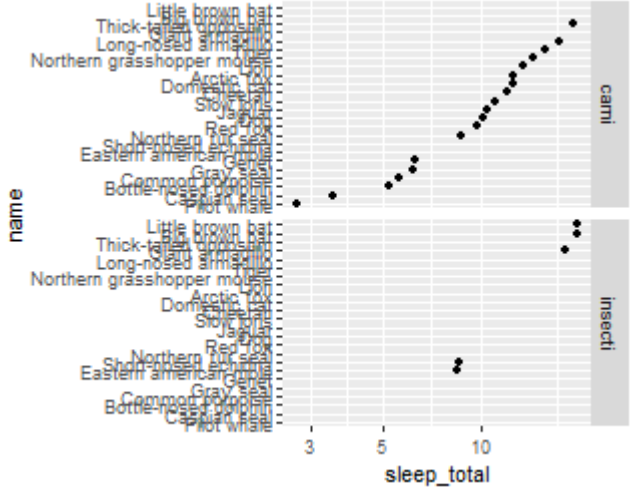
```
p ← ggplot(mtcars, aes(x = wt, y = mpg, colour = factor(cyl),
                       shape = factor(am))) +
  geom_point(show.legend = FALSE) + labs(x = NULL, y = NULL)
p
p + facet_wrap(~factor(cyl)) # 1d ribbon of panels that is wrapped into 2d
p + facet_grid(factor(am) ~ factor(cyl)) # 2d grid of panels defined by variables which form the rows and
```



Facets

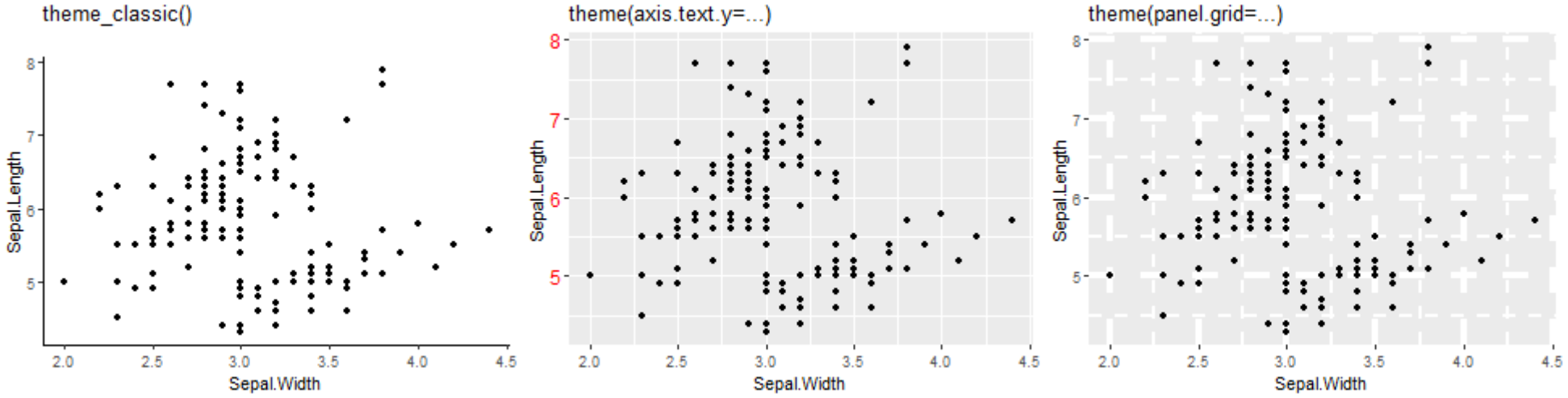
`facet_grid` have two additional arguments: `scales` and `space`.

```
msleep$name ← factor(msleep$name, levels = msleep$name[order(msleep$sleep_total)])
p ← ggplot(msleep[msleep$vore %in% c("carni","insecti"), ],
            aes(x = sleep_total, y = name)) +
  geom_point() + scale_x_log10()
p + facet_grid(vore~.)
p + facet_grid(vore~., scales = "free", space = "free")
```



Themes

- Complete themes `theme_*()`
- Theme elements `theme()`

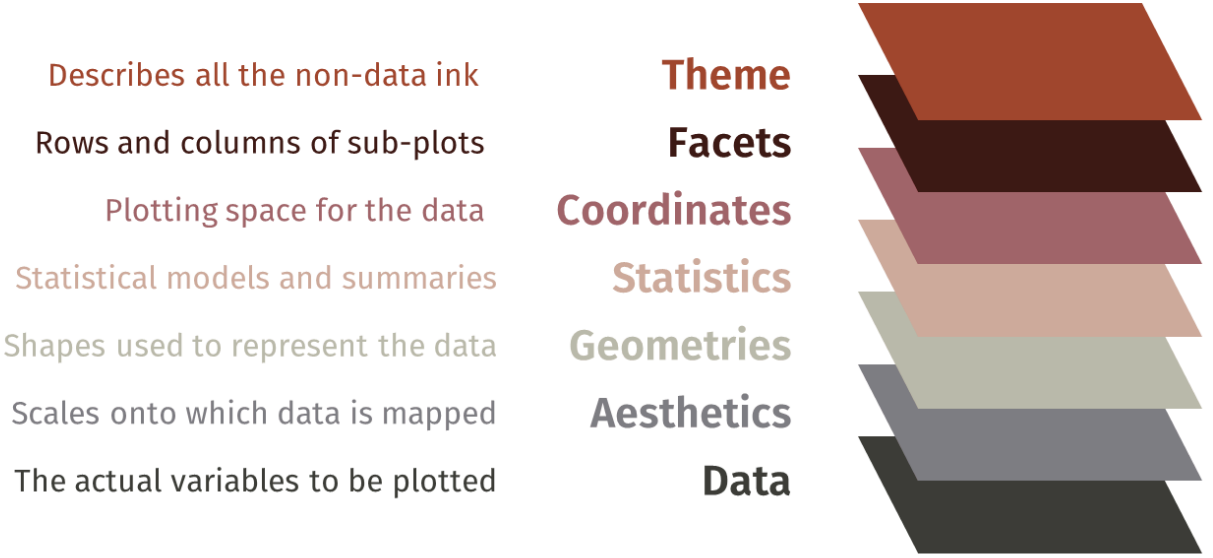


- Full list of theme elements: `?theme`

Components

The layered grammar defines a plot as a combination of:

- **Layers**
  - Data
  - Aesthetic mapping
  - Geometric objects
  - Statistical transformation
  - Position adjustment
- **Scales**
- **Coordinate system**
- **Faceting specification**
- **Theme** (not in the original grammar)



Grammar of Graphics:
A layered approach to elegant visuals

Components

Exercise | Update the plot to match the requirements below

```
p ← ggplot(msleep, aes(x = vore, y = sleep_total, colour = vore,
                       fill = vore)) +
  geom_violin(alpha = 0.2, colour = NA) +
  geom_jitter(width = 0.3)
```

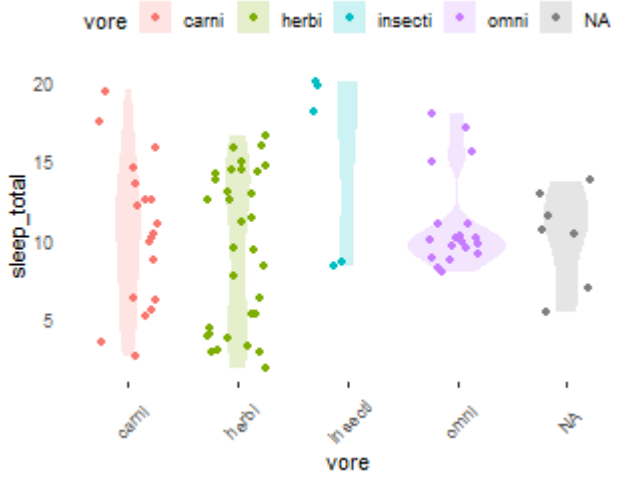
- Legend on top of the plot
- White background
- No ticks in y axis
- Axis labels rotated to an angle of 45 degrees

Note: check `?theme` options

Components

Exercise | Update the plot to match the requirements below

```
p +
  theme(legend.position = "top",
        panel.background = element_rect(fill = "white"),
        axis.ticks.y = element_blank(),
        axis.text.x = element_text(angle = 45, vjust = 0.5))
```



Upload `T2.1_slides.Rmd` with the completed exercises (text included) to aul@-ESCI