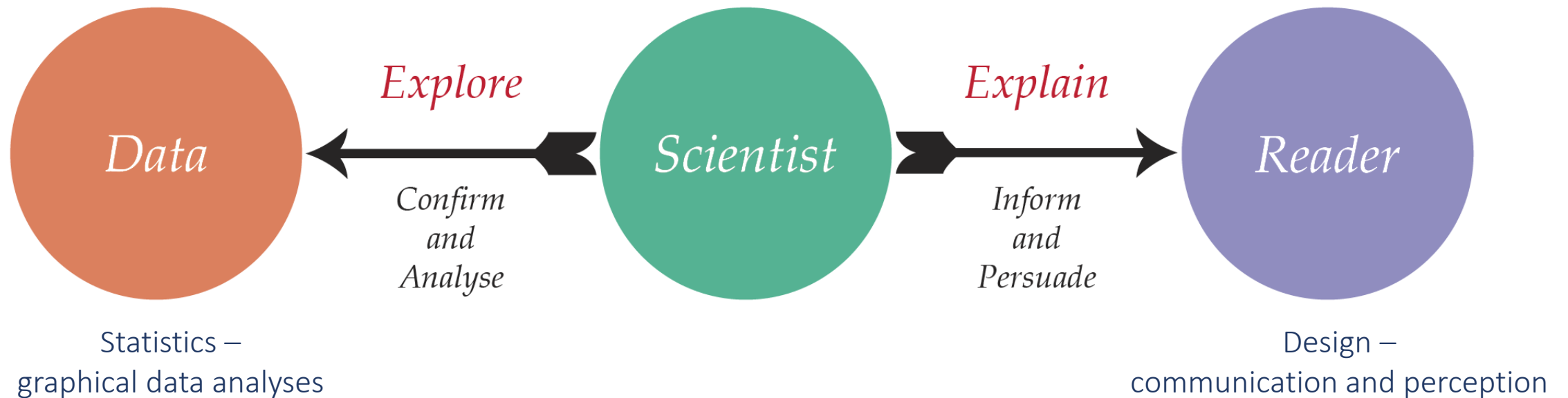


- Data visualisation = statistics and design
- Who is intended audience?
 - exploratory vs explanatory





Hadley Wickham

Why ggplot2 not base R plotting?

- Function call is unified = *ggplot*
 - *ggplot(data=data , aes(x=x, y=y)) + geom()*
- Base R plot:
 - Function calls not unified
 - Plot is an image (vs *ggplot* object)
 - Must add legend yourself

The quick brown fox jumps over the lazy dog

<i>Article</i>	<i>The</i>	<i>A</i>	<i>The</i>
<i>Adjective</i>	<i>quick brown</i>	<i>rabid red</i>	
<i>Noun</i>	<i>fox</i>	<i>fox</i>	<i>Hunter</i>
<i>Verb</i>	<i>jumps</i>	<i>bit</i>	<i>shot</i>
<i>Preposition</i>	<i>over</i>		
<i>Article</i>	<i>the</i>	<i>the</i>	<i>the</i>
<i>Adjective</i>	<i>lazy</i>	<i>friendly</i>	<i>rabid red</i>
<i>Noun</i>	<i>dog.</i>	<i>dog.</i>	<i>fox.</i>



The grammar of graphics

<i>Data</i>	{variables of interest}			
<i>Aesthetics</i>	<i>x-axis</i> <i>y-axis</i>	<i>colour</i> <i>fill</i>	<i>size</i> <i>labels</i>	<i>alpha</i> <i>shape</i>
<i>Geometries</i>	<i>point</i>	<i>line</i>	<i>histogram</i>	<i>bar</i>
<i>Facets</i>	<i>columns</i>	<i>rows</i>		
<i>Statistics</i>	<i>binning</i>	<i>smoothing</i>	<i>descriptive</i>	<i>inferential</i>
<i>Coordinates</i>	<i>cartesian</i>	<i>fixed</i>	<i>polar</i>	<i>limits</i>
<i>Themes</i>	<i>non-data ink</i>			

- Structure / format dictates type of graph you will plot
- Use `tidyr::gather`, `spread`, `separate` and `unite()`
- `iris.wide` = single variable split onto more than 1 column

```
iris.wide <- iris %>%
  dplyr::mutate(Flowe = 1:nrow(iris)) %>% #add column with unique IDs
  gather(key="Key", value="Value", -Species, -Flowe) %>%
  separate(col=Key, into=c("Part", "Measure"), sep="\\.") %>%
  spread(key=Measure, value=Value) %>%
  dplyr::select(-Flowe)
```

```
> print(iris)
```

```
# A tibble: 150 x 5
```

	Species	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
	<fct>	<dbl>	<dbl>	<dbl>	<dbl>
1	setosa	5.1	3.5	1.4	0.2
2	setosa	4.9	3	1.4	0.2
3	setosa	4.7	3.2	1.3	0.2
4	setosa	4.6	3.1	1.5	0.2
5	setosa	5	3.6	1.4	0.2
6	setosa	5.4	3.9	1.7	0.4
7	setosa	4.6	3.4	1.4	0.3
8	setosa	5	3.4	1.5	0.2
9	setosa	4.4	2.9	1.4	0.2
10	setosa	4.9	3.1	1.5	0.1

```
# ... with 140 more rows
```



```
> print(iris.wide)
```

```
# A tibble: 300 x 4
```

	Species	Part	Length	Width
	<fct>	<chr>	<dbl>	<dbl>
1	setosa	Petal	1.4	0.2
2	setosa	Sepal	5.1	3.5
3	setosa	Petal	1.4	0.2
4	setosa	Sepal	4.9	3
5	setosa	Petal	1.3	0.2
6	setosa	Sepal	4.7	3.2
7	setosa	Petal	1.5	0.2
8	setosa	Sepal	4.6	3.1
9	setosa	Petal	1.4	0.2
10	setosa	Sepal	5	3.6

```
# ... with 290 more rows
```

- iris.tidy = each *row* is single *obs*, each *column* is single *variable*

```
iris.tidy <- iris %>%
  gather(key="Key", value="value", -Species) %>%
  separate(col=Key, into=c("Part", "Measure"), sep="\\\.")
```

```
> print(iris)
# A tibble: 150 x 5
  species Sepal.Length Sepal.Width Petal.Length Petal.Width
  <fct>      <dbl>         <dbl>         <dbl>         <dbl>
1 setosa     5.1           3.5           1.4           0.2
2 setosa     4.9           3           1.4           0.2
3 setosa     4.7           3.2           1.3           0.2
4 setosa     4.6           3.1           1.5           0.2
5 setosa     5           3.6           1.4           0.2
6 setosa     5.4           3.9           1.7           0.4
7 setosa     4.6           3.4           1.4           0.3
8 setosa     5           3.4           1.5           0.2
9 setosa     4.4           2.9           1.4           0.2
10 setosa    4.9           3.1           1.5           0.1
# ... with 140 more rows
```



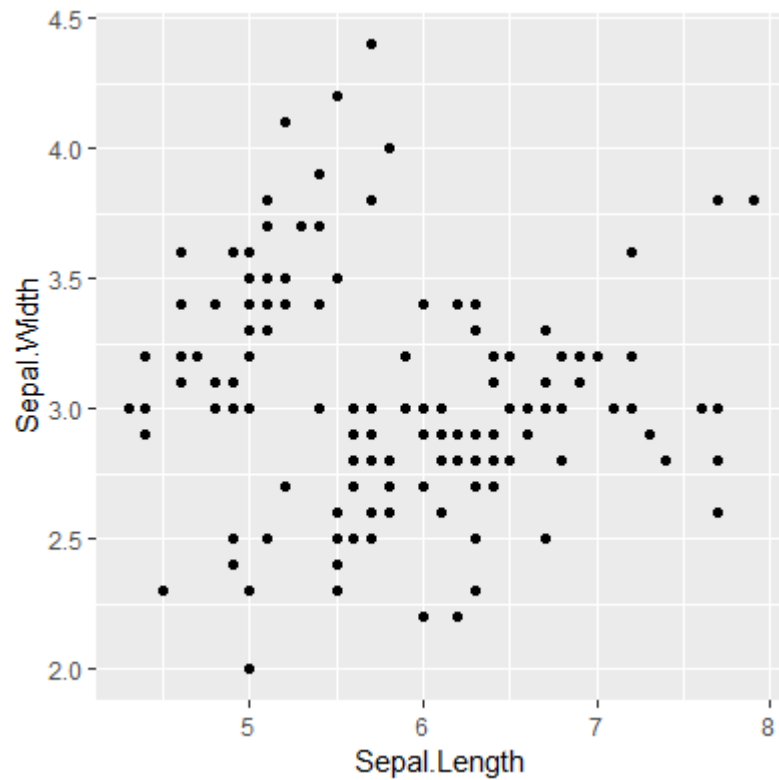
```
> print(iris.tidy)
# A tibble: 600 x 4
  species Part Measure value
  <fct>    <chr> <chr>    <dbl>
1 setosa Sepal Length  5.1
2 setosa Sepal Length  4.9
3 setosa Sepal Length  4.7
4 setosa Sepal Length  4.6
5 setosa Sepal Length  5
6 setosa Sepal Length  5.4
7 setosa Sepal Length  4.6
8 setosa Sepal Length  5
9 setosa Sepal Length  4.4
10 setosa Sepal Length  4.9
# ... with 590 more rows
```

```
ggplot(data=iris, aes(x=Sepal.Length, y=Sepal.Width)) +  
  geom_point() #inherits from original aes call
```

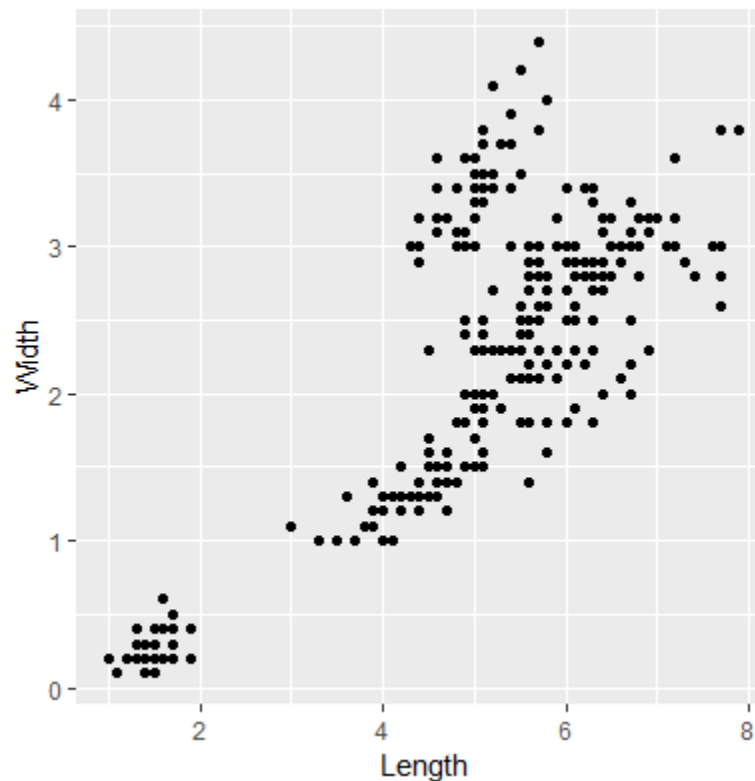
```
ggplot(iris.wide, aes(x=Length, y=width)) +  
  geom_point()
```

```
ggplot(iris.tidy, aes(x=Measure, y=Value)) +  
  geom_point(position = position_jitter(width=0.2))
```

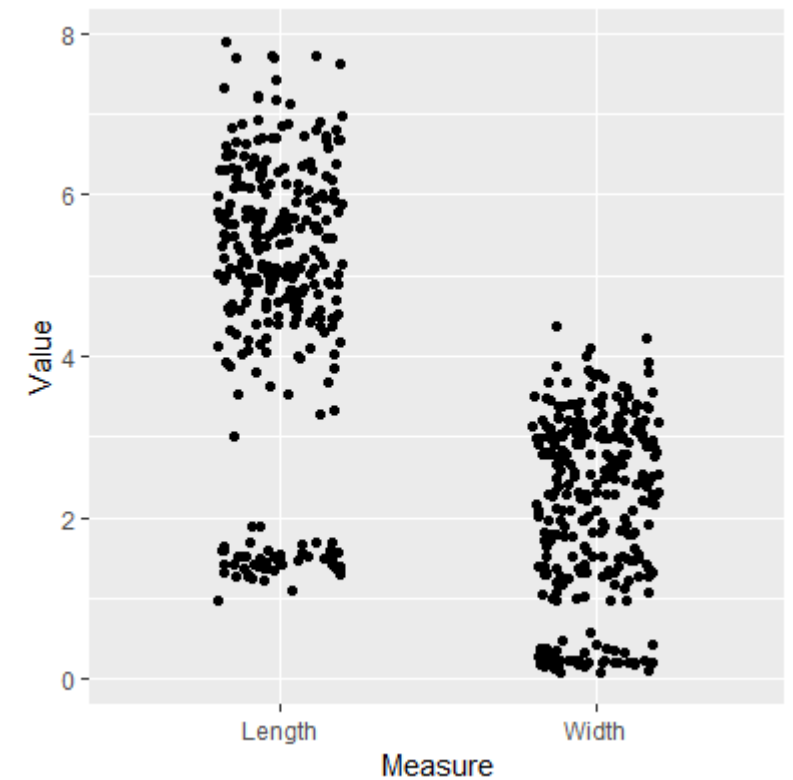
iris



































iris.wide



iris.tidy



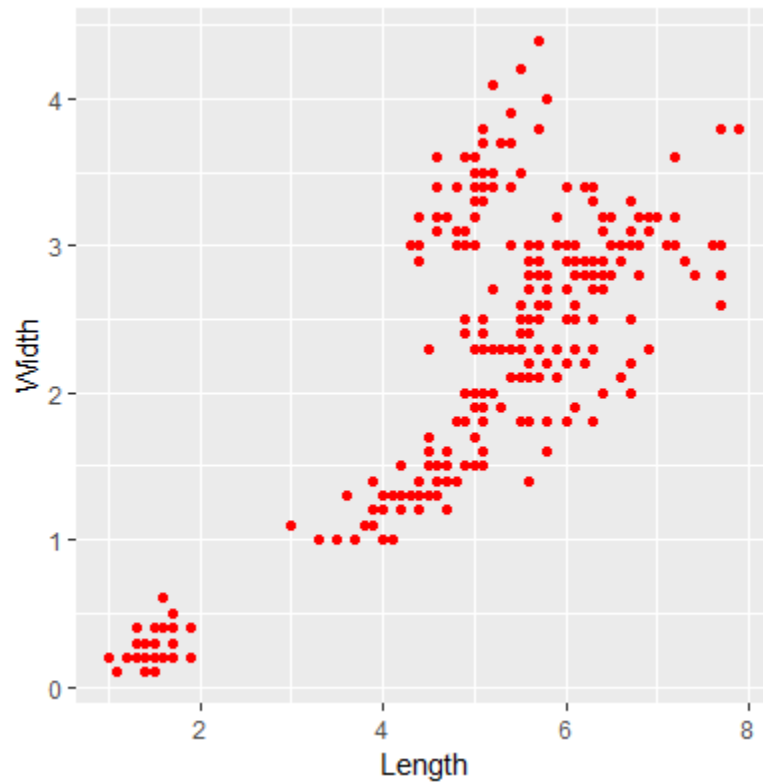
- | | | | | | |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | |
|  |  |  |  |  | |
| 5 | 6 | 7 | 8 | 9 | |
|  |  |  |  |  | |
| 10 | 11 | 12 | 13 | 14 | |
|  |  |  |  |  | |
| 15 | 16 | 17 | 18 | 19 | |
|  |  |  |  |  | |
| 20 | 21 | 22 | 23 | 24 | 25 |
|  |  |  |  |  |  |
- | | |
|--------------|--|
| 6.'twodash' |  |
| 5.'longdash' |  |
| 4.'dotdash' |  |
| 3.'dotted' |  |
| 2.'dashed' |  |
| 1.'solid' |  |
| 0.'blank' | |

Aesthetic	Description
x	X axis position
y	Y axis position
colour	Colour of dots, outlines of other shapes
fill	Fill colour
size	Diameter of points, thickness of lines
alpha	Transparency
linetype	Line dash pattern
labels	Text on a plot or axes
shape	Shape

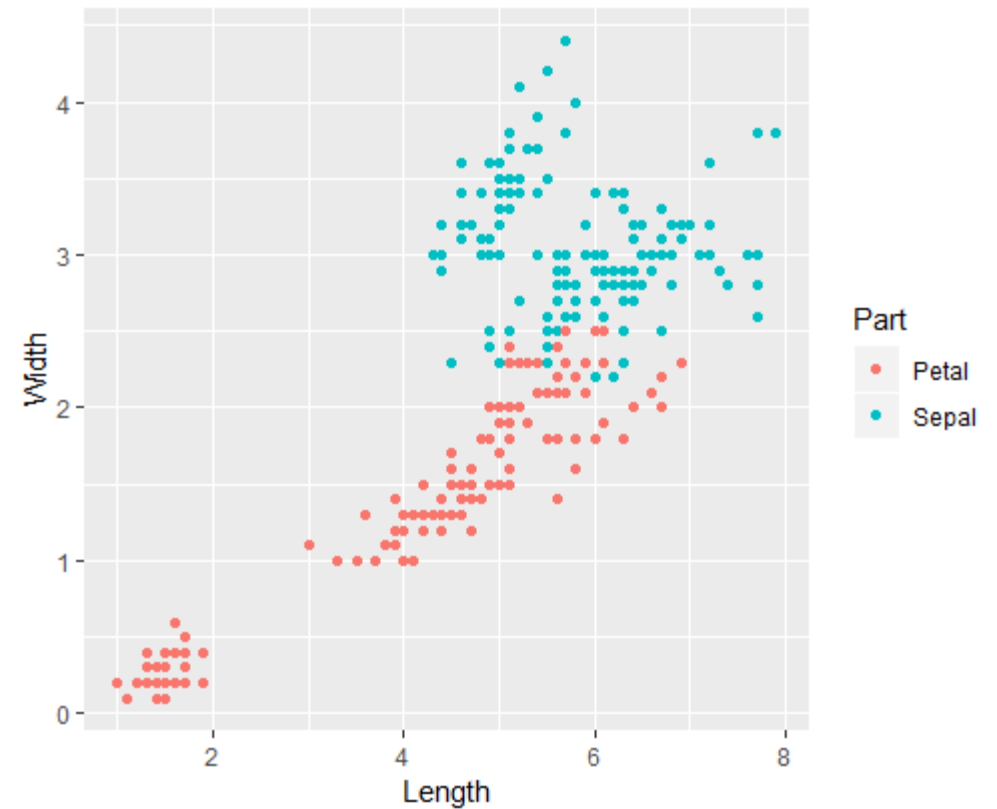
Attributes vs *Aesthetics*

Data

```
ggplot(iris.wide, aes(x=Length, y=width)) +  
  geom_point(col = "red")
```

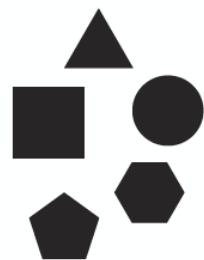


```
ggplot(iris.wide, aes(x=Length, y=width, col=Part)) +  
  geom_point()
```



Low *Efficiency in Decoding Separate Groups* High

Filled
Shapes



Sequential
Colours



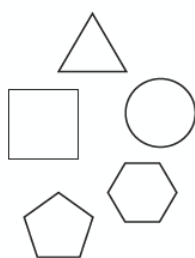
Qualitative
Colours



Hatching



Shape
Outlines



Labels

ANT1
FRG1 FRG2
Gapdh
DUX4

Line Width



Line Type



Line Colours



Aesthetics

Data



Modifying Aesthetics

- Positions – arrange overlapping geoms

- identity
- dodge
- stack
- fill
- jitter
- jitterdodge

- Scale functions – control how a plot maps data to visuals

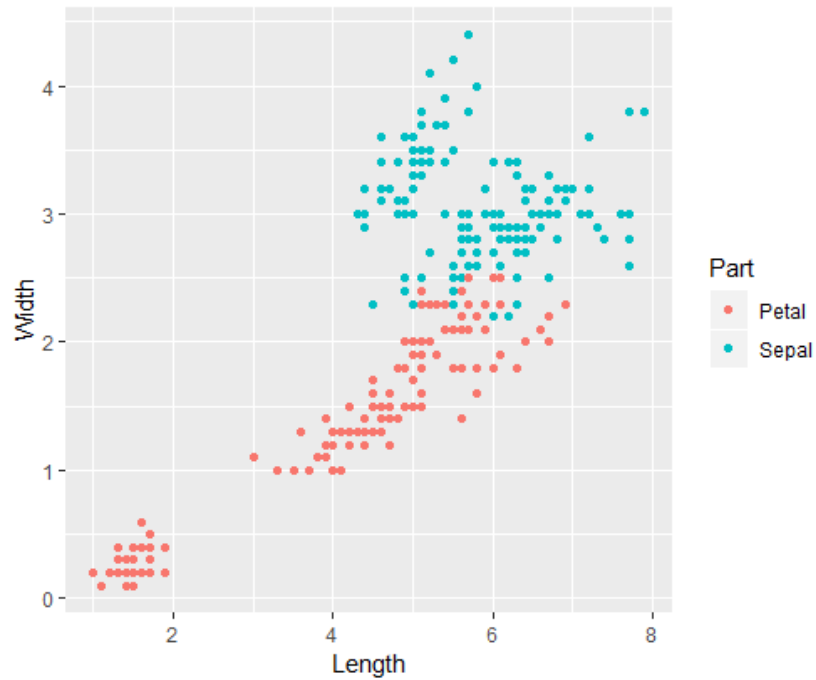
- `scale_*_continuous`
- `scale_*_discrete`
- `scale_*_manual`

Where * can be:

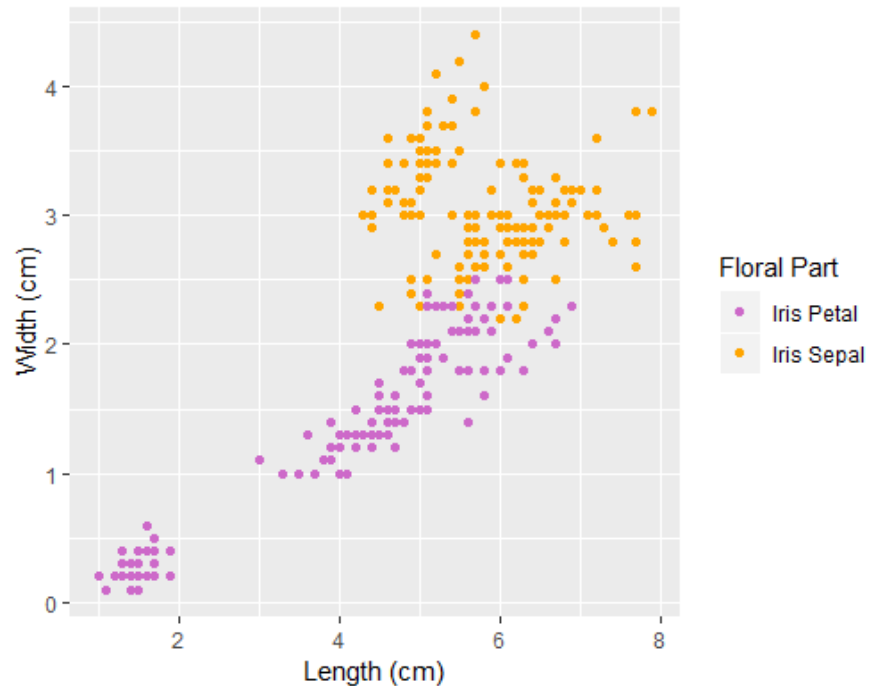
- x
- y
- colour
- fill
- size
- linetype
- shape etc...

Scaling *Aesthetics*

Data



```
ggplot(iris.wide, aes(x=Length, y=width, col=Part)) +  
  geom_point()
```

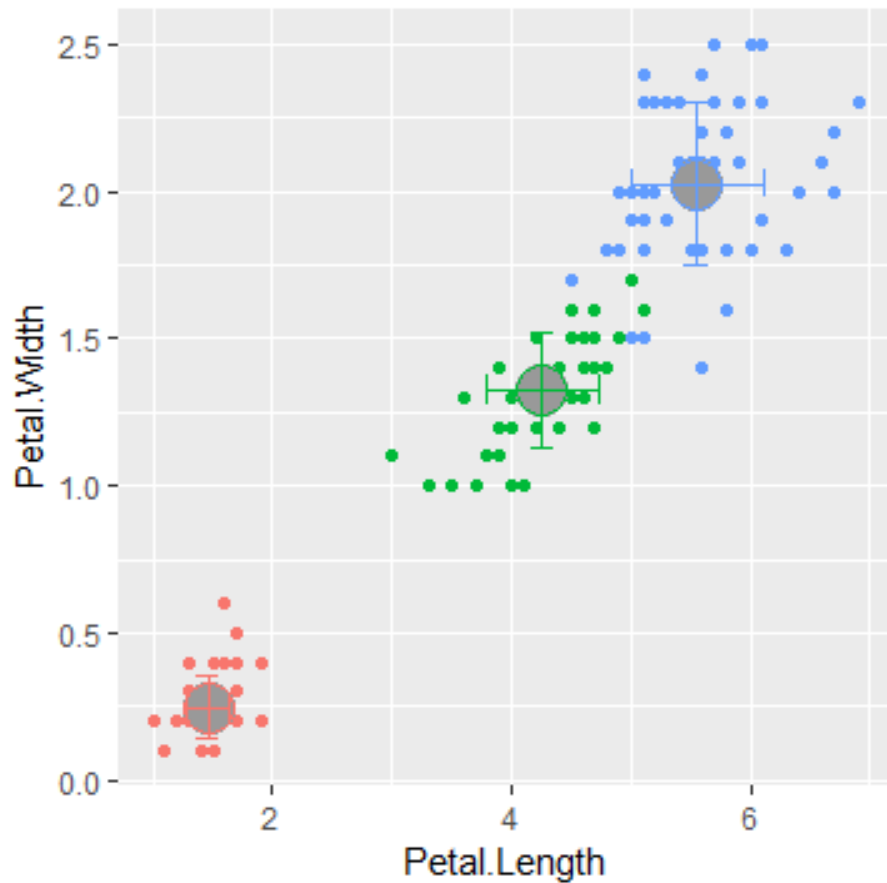


```
ggplot(iris.wide, aes(x=Length, y=width, col=Part)) +  
  geom_point() +  
  scale_colour_manual(values=c("orchid3", "orange1"),  
    labels=c("Iris Petal", "Iris Sepal"), name="Floral Part") +  
  xlab("Length (cm)") +  
  ylab("width (cm)")
```



- 37 geometries (e.g. point, line, bar, boxplot)
- Each geom has essential aes (usually x & y)
- aes inside geom – control each layer independently, and add different data sets
- ggplot2 summarises your data within plot code
- Many geoms have associated stat layer
- Two calls available (geom \leftrightarrow stat)
- geom or stat? – whatever is intuitive to you

stat_	geom_
stat_bin()	geom_histogram()
stat_bin()	geom_bar()
stat_bin()	geom_freqpoly()
stat_smooth()	geom_smooth()
stat_boxplot()	geom_boxplot()
stat_bindot()	geom_dotplot()
stat_bin2d()	geom_bin2d()
stat_binhex()	geom_hex()
stat_contour()	geom_contour()
stat_quantile()	geom_quantile()
stat_sum()	geom_count()

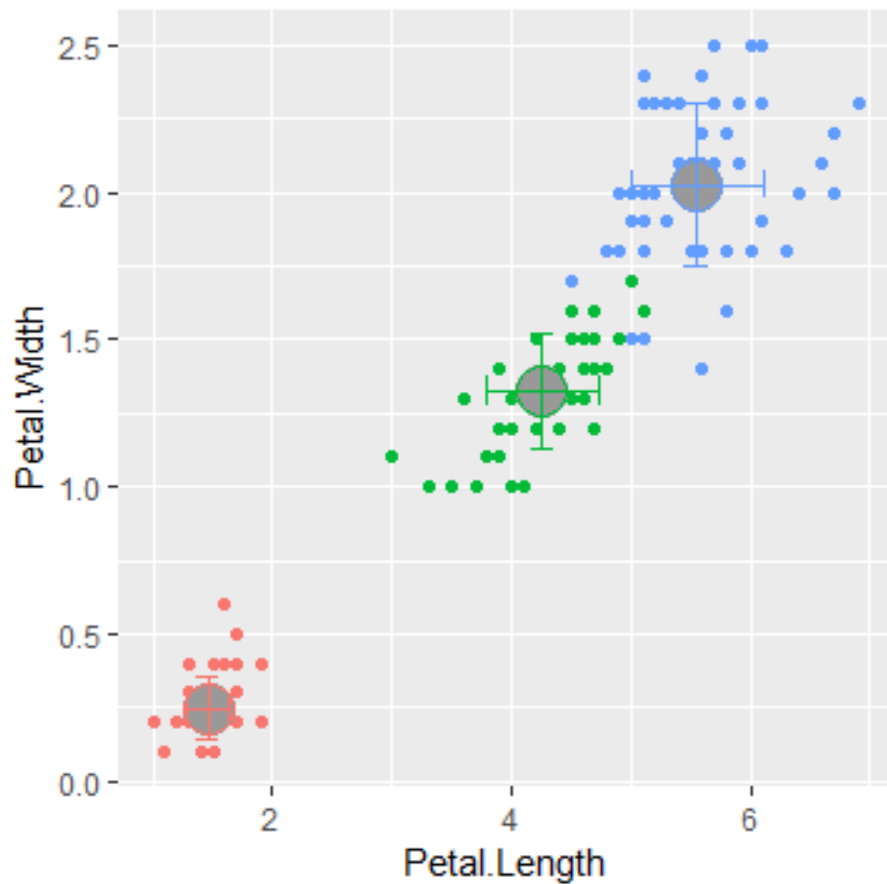


aes inside geom



```
iris.summary <- iris %>%
  group_by(Species) %>%
  summarise_all(funs(mean,sd)) #dplyr
```

```
# A tibble: 3 x 4
  Petal.Length_mean Petal.Length_sd Petal.width_mean Petal.width_sd
      <dbl>         <dbl>         <dbl>         <dbl>
1         1.46         0.174         0.246         0.105
2         4.26         0.470         1.33          0.198
3         5.55         0.552         2.03          0.275
```

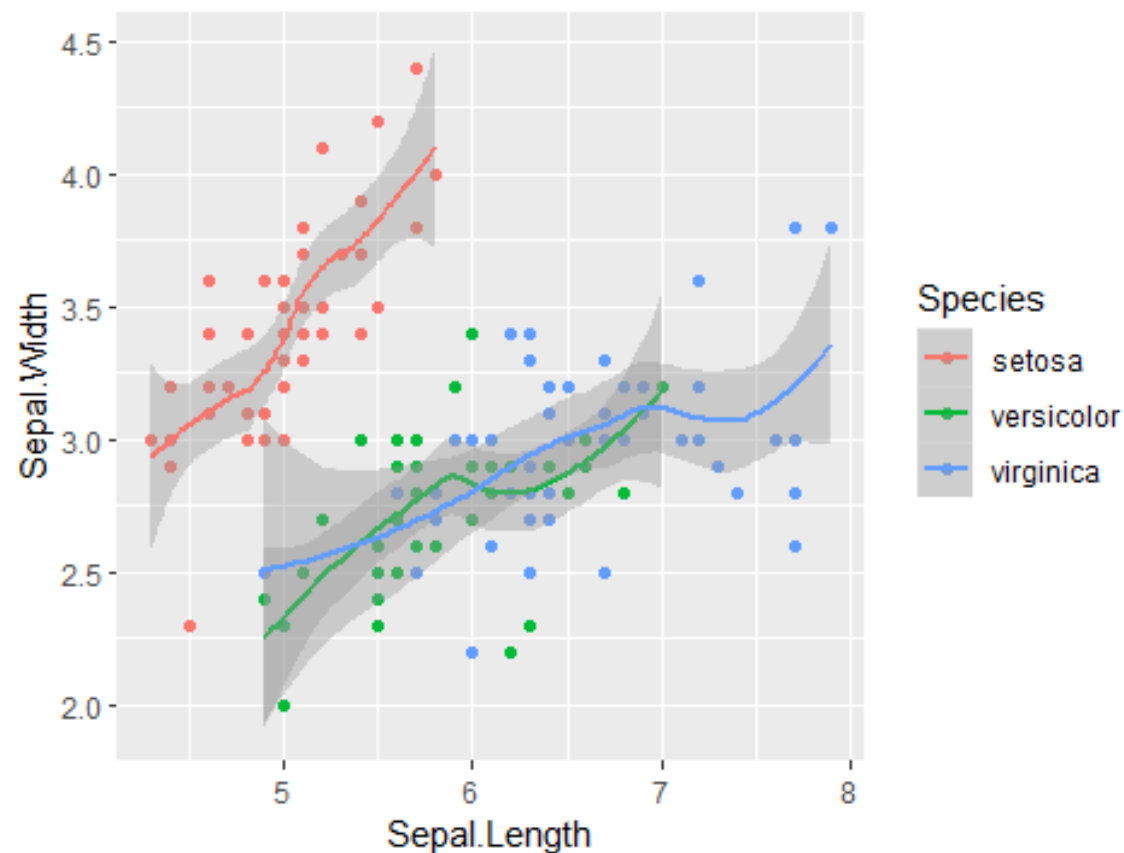


aes inside geom



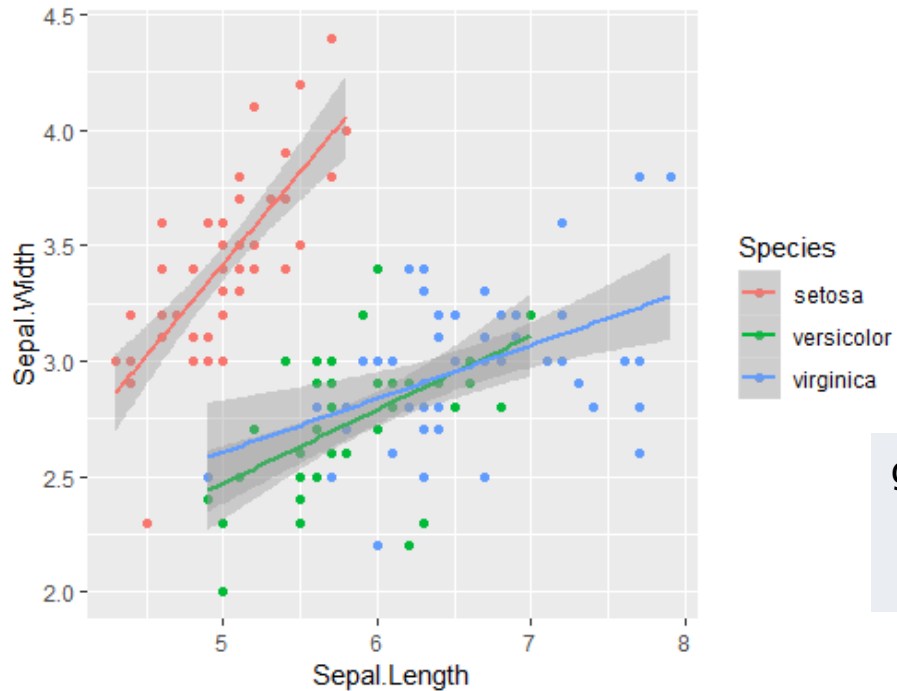
```
ggplot(iris, aes(x=Petal.Length, y=Petal.Width, col=Species)) +
  geom_point() +
  geom_point(data=iris.summary, aes(x=Petal.Length_mean, y=Petal.Width_mean), size=7,
    shape=21, fill="grey60") +
  geom_errorbar(data=iris.summary, aes(x=Petal.Length_mean, y=Petal.Width_mean,
    ymin=Petal.Width_mean-Petal.Width_sd, ymax=Petal.Width_mean+Petal.Width_sd), width=0.2) +
  geom_errorbarh(data=iris.summary, aes(x=Petal.Length_mean, y=Petal.Width_mean,
    xmin=Petal.Length_mean-Petal.Length_sd, xmax=Petal.Length_mean+Petal.Length_sd), height=0.1)
```

geom_point and geom_smooth / stat_smooth

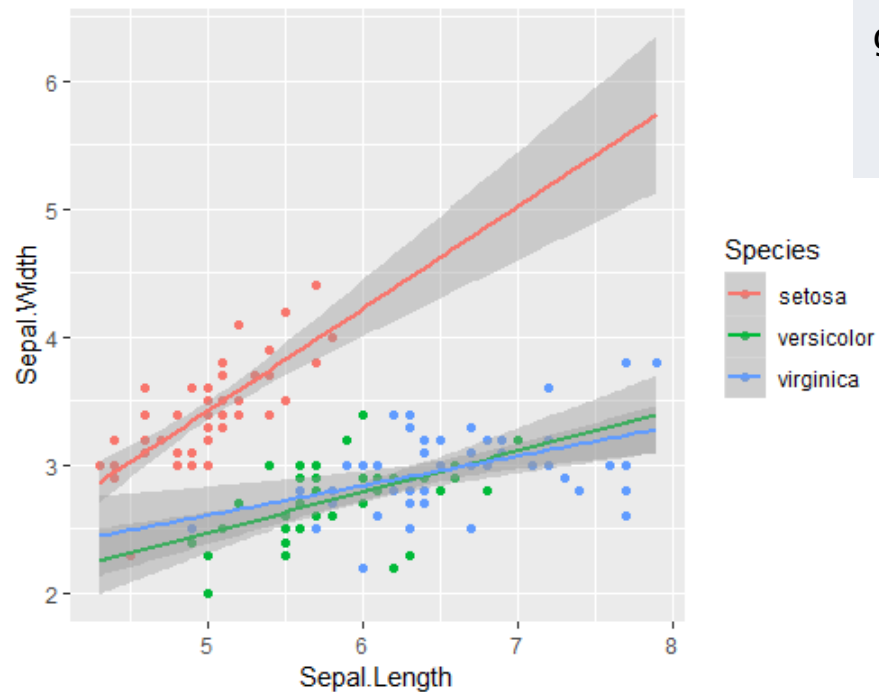


```
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width, col=Species)) +  
  geom_point() +  
  geom_smooth() #use "span" to control wiggleness,  
               #se=FALSE to remove standard error ribbon
```

``geom_smooth()`` using `method = 'loess'` and formula `'y ~ x'`



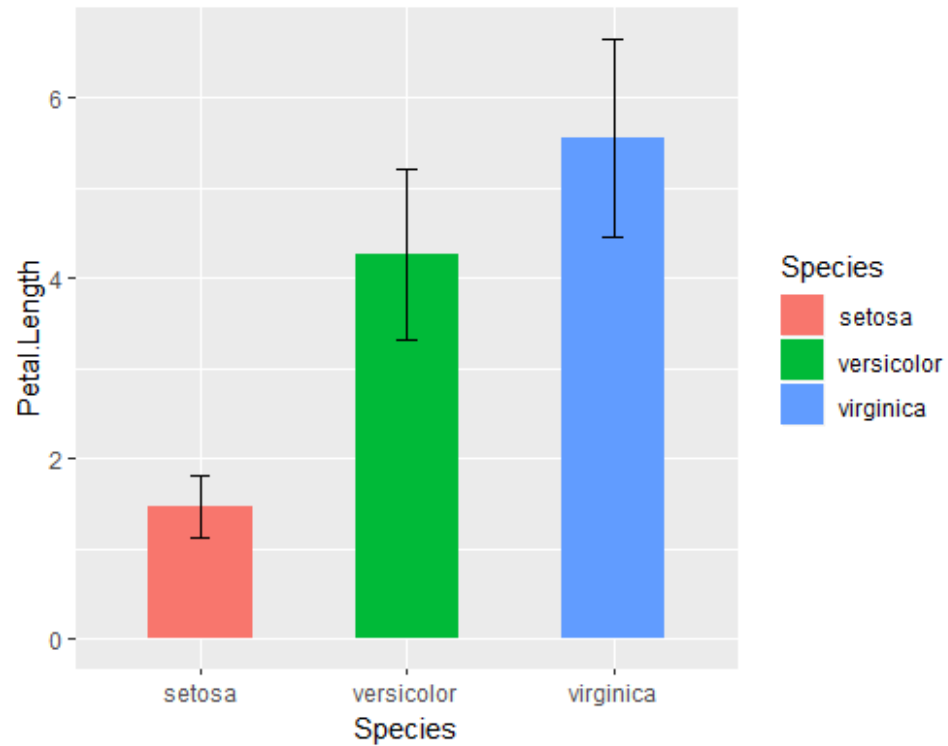
```
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width, col=Species)) +
  geom_point() +
  stat_smooth(method="lm") #also loess, glm, gam, MASS::rlm
```



```
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width, col=Species)) +
  geom_point() +
  stat_smooth(method="lm", fullrange=TRUE) #predict beyond data
```



geom vs stat

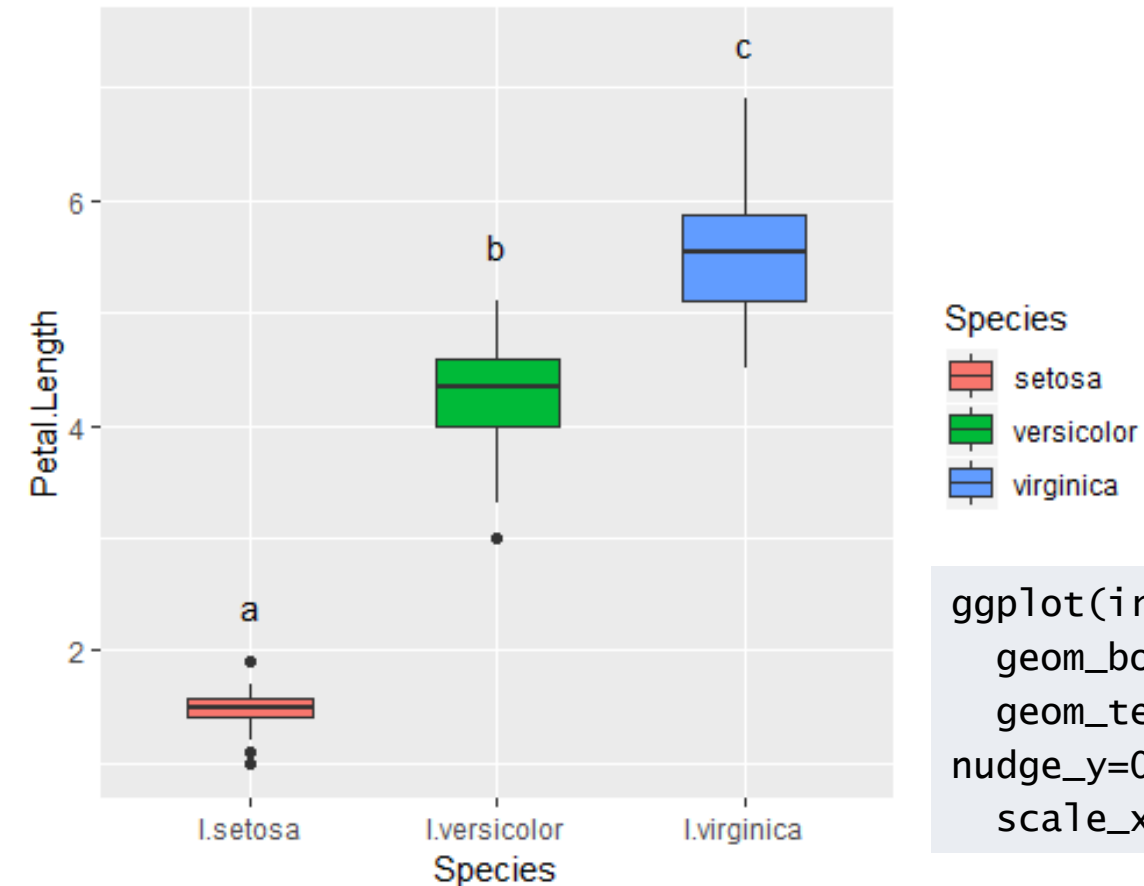


```
ggplot(iris, aes(x=Species, y=Petal.Length, fill=Species)) +  
  geom_bar(stat="summary", fun.y=mean, width=0.5) +  
  geom_errorbar(stat="summary", fun.data=mean_sd1, width=0.1)
```

```
ggplot(iris, aes(x=Species, y=Petal.Length, fill=Species)) +  
  stat_summary(geom="bar", fun.y=mean, width=0.5) +  
  stat_summary(geom="errorbar", fun.data=mean_sd1, width=0.1)  
#mean_sd1 = 2 standard deviations  
#for 1 standard deviation use: fun.args = list(mult = 1)
```

geom_boxplot

- Bar graphs / dynamite plots = BAD!
- Boxplots show 5-number summary and give indication of data distribution



```
Tukey <- iris %>% group_by(Species) %>%  
  summarise(value = max(Petal.Length)) %>%  
  mutate(letters = c("a", "b", "c"))
```

```
ggplot(iris, aes(x=Species, y=Petal.Length, fill=Species)) +  
  geom_boxplot(width=0.5) +  
  geom_text(data=Tukey, aes(x=Species, y=value, label=letters),  
    nudge_y=0.5) + #also see "annotate"  
  scale_x_discrete(labels=c("I.setosa", "I.versicolor", "I.virginica"))
```

geom_histogram

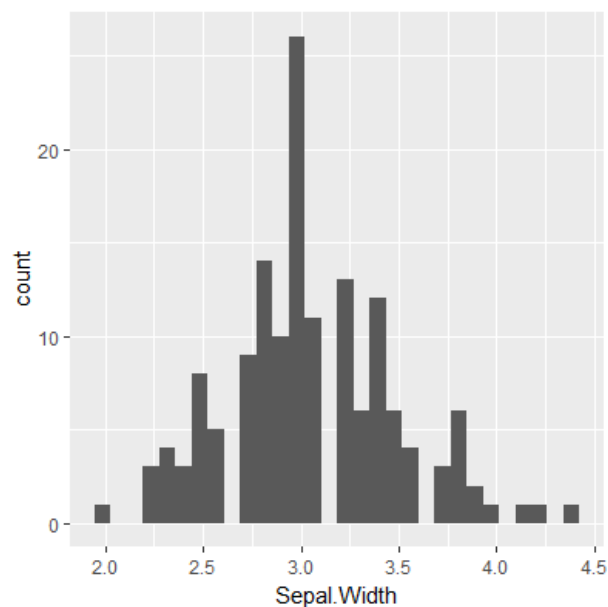
```
ggplot(iris, aes(x=Sepal.Width)) +  
  geom_histogram() #default binwidth = range/30
```

```
ggplot(iris, aes(x=Sepal.Width)) +  
  geom_histogram(binwidth=0.1) #choose own binwidth
```

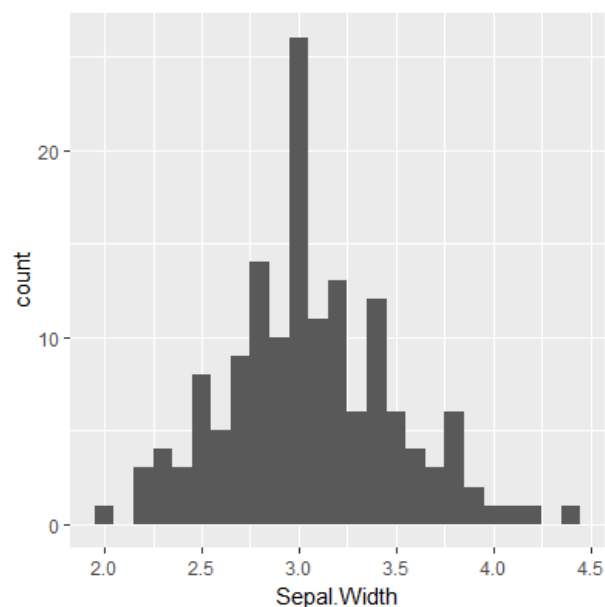
```
ggplot(iris, aes(x = Sepal.Width)) +  
  geom_histogram(aes(y=..density..), binwidth = 0.1)  
#plot density instead of counts
```



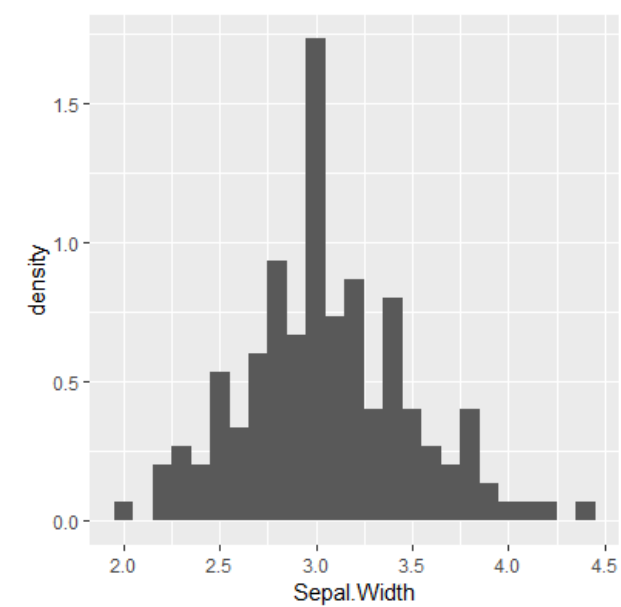
default binwidth



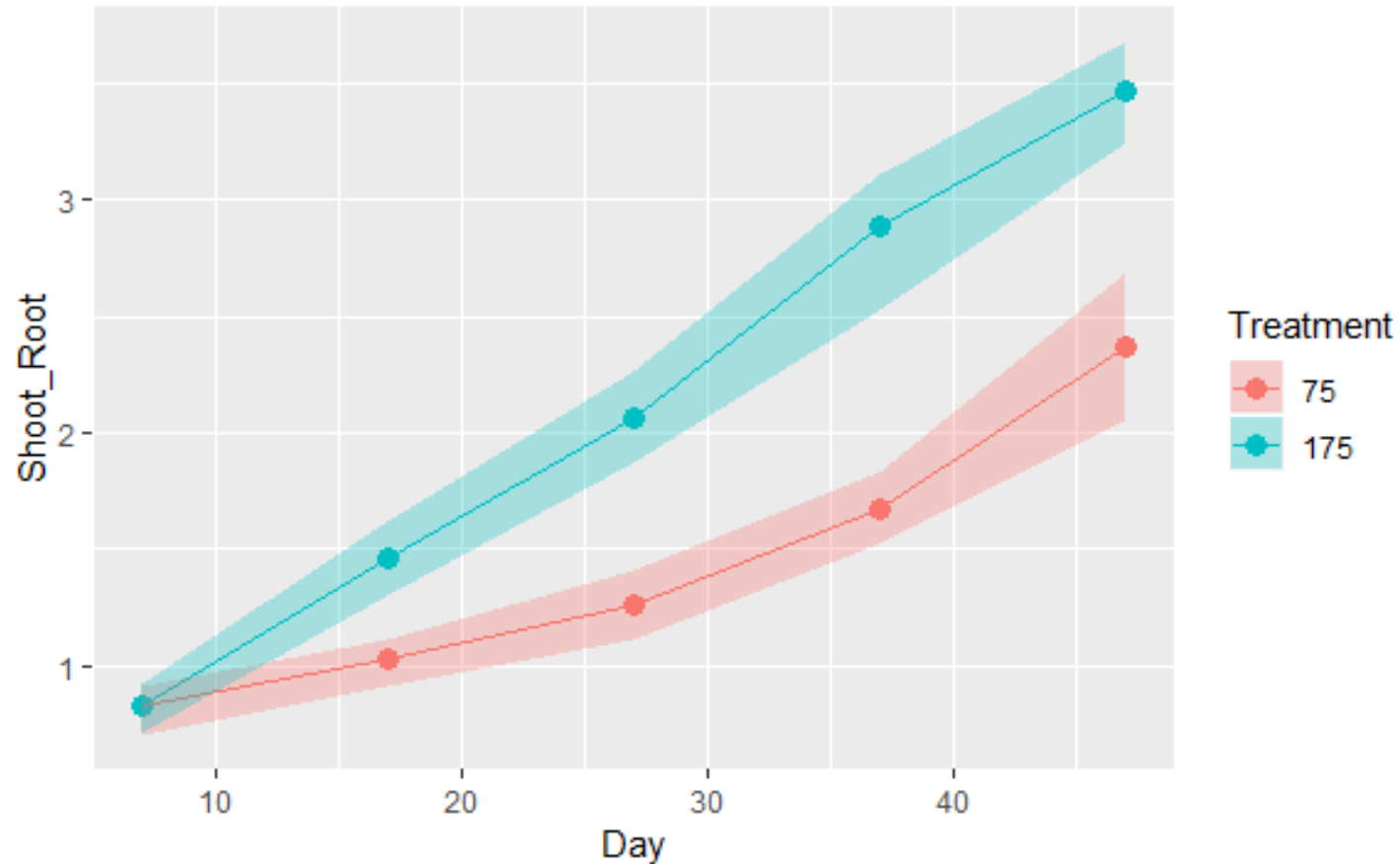
choose binwidth



density

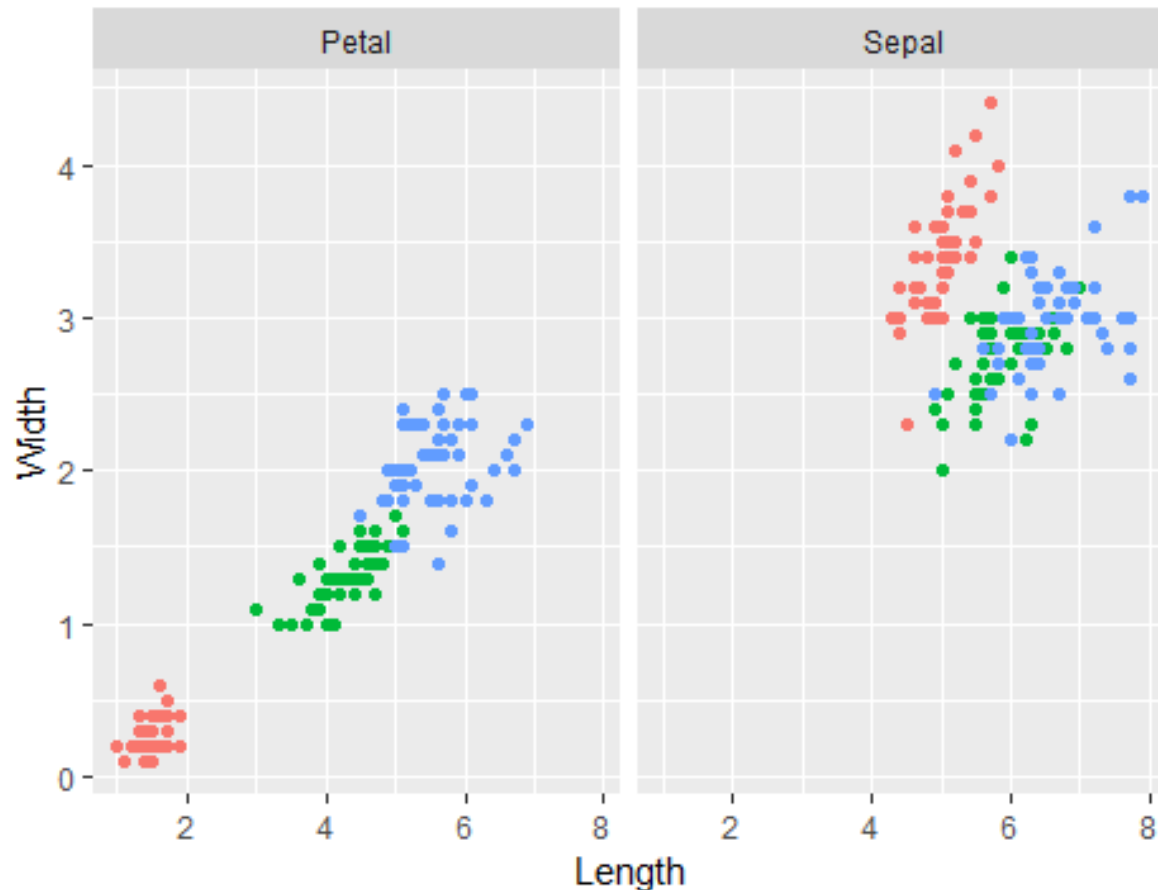


geom_line + geom_ribbon



```
ggplot(shoot.root, aes(x=Day, y=Shoot_Root, col=factor(Treatment), fill=factor(Treatment))) +  
  geom_point(stat="summary", fun.y=mean, size=3) +  
  geom_line(stat="summary", fun.y=mean) +  
  geom_ribbon(stat="summary", fun.data=mean_cl_boot, alpha=0.3, col=NA) + #note alpha and col  
  labs(colour="Treatment") + #little glitchy thing  
  labs(fill="Treatment")
```

- Split data according to levels of a factor, therefore adding another variable
- Common coordinate system = aids decoding
- `facet_grid(row ~ column)`



Species

- setosa
- versicolor
- virginica

```
ggplot(iris.wide, aes(x=Length, y=width, col=Species)) +  
  geom_point() +  
  facet_grid(.~Part)
```

scale*_continuous vs coord_cartesian

```
iris.lm <- ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width, col=Species)) +  
  geom_point() +  
  stat_smooth(method="lm")
```

```
iris.lm + scale_x_continuous(limits = c(4.5, 5.5))
```

Warning messages:

- 1: Removed 95 rows containing non-finite values (stat_smooth).
- 2: Removed 95 rows containing missing values (geom_point).

Coordinates

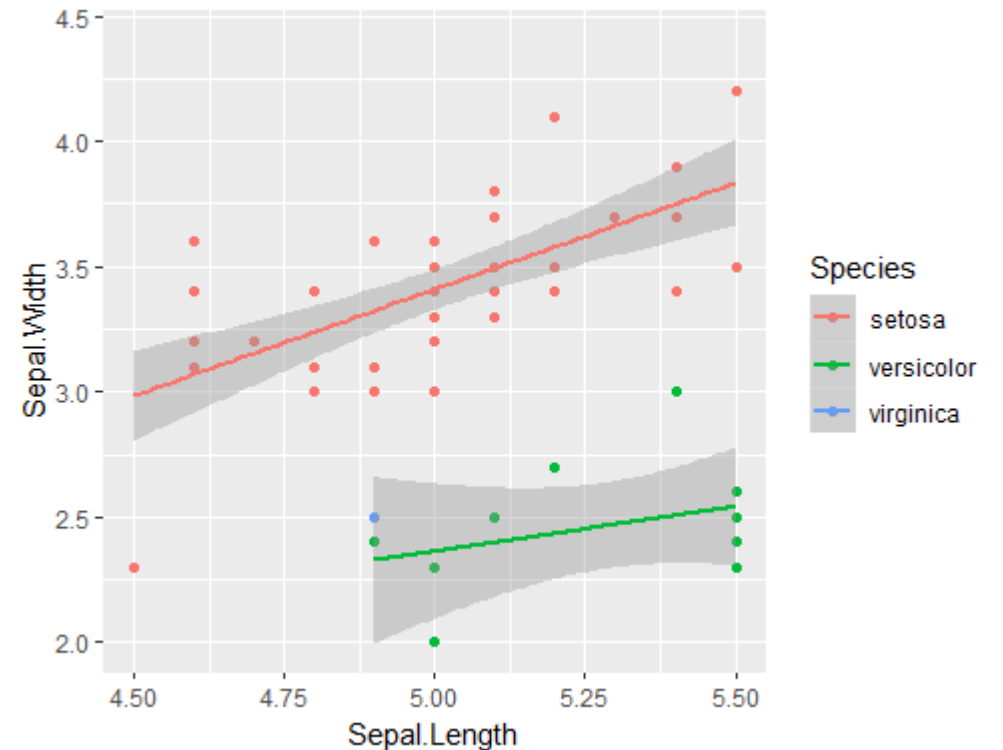
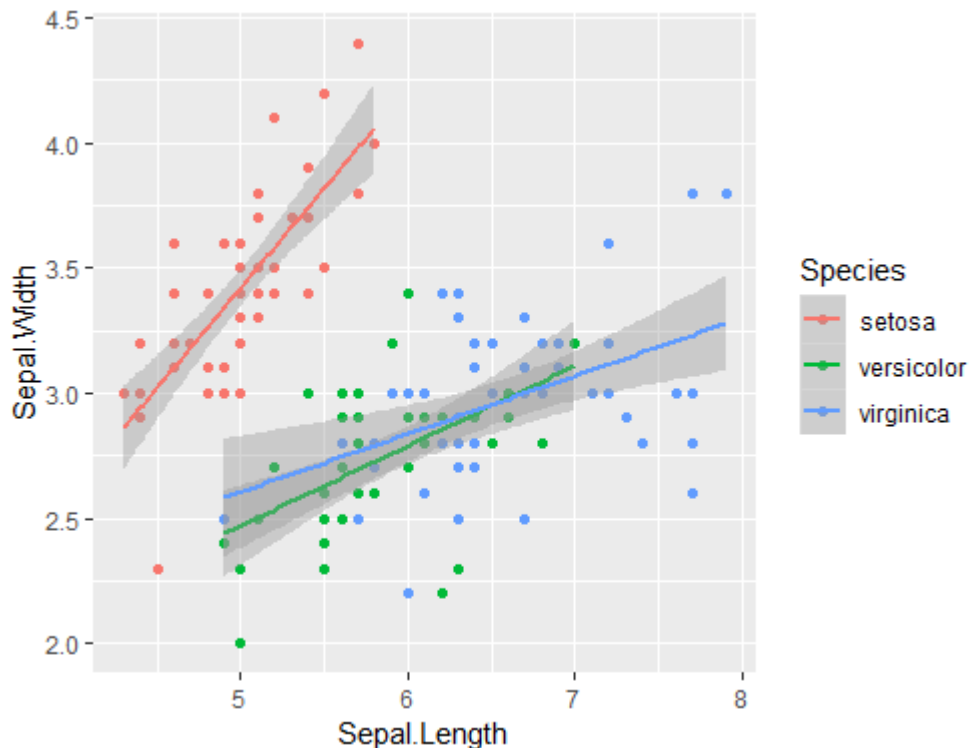
Statistics

Facets

Geometries

Aesthetics

Data



scale_*_continuous vs coord_cartesian

```
iris.lm + coord_cartesian(xlim = c(4.5, 5.5))
```

```
#also see coord_equal(), coord_fixed() and coord_flip()
```

Coordinates

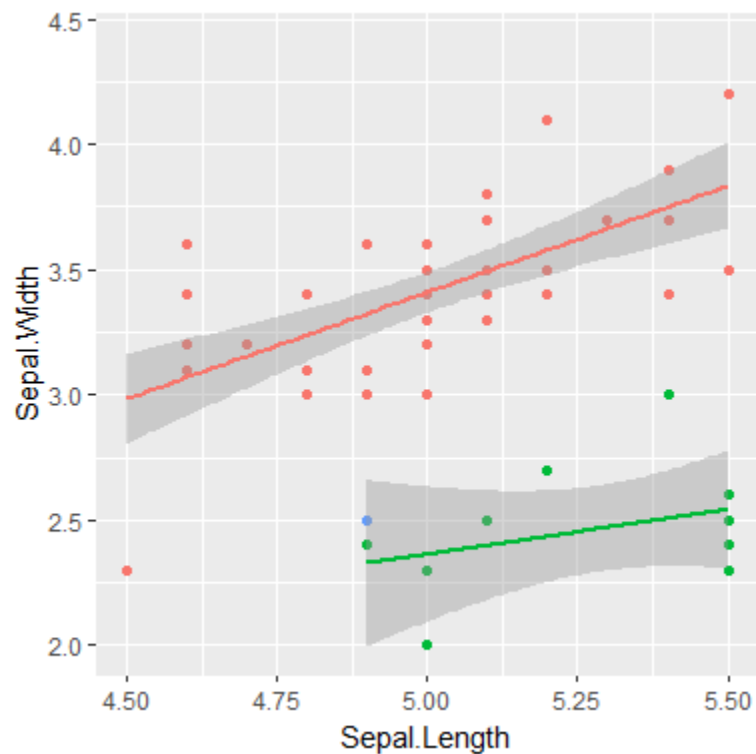
Statistics

Facets

Geometries

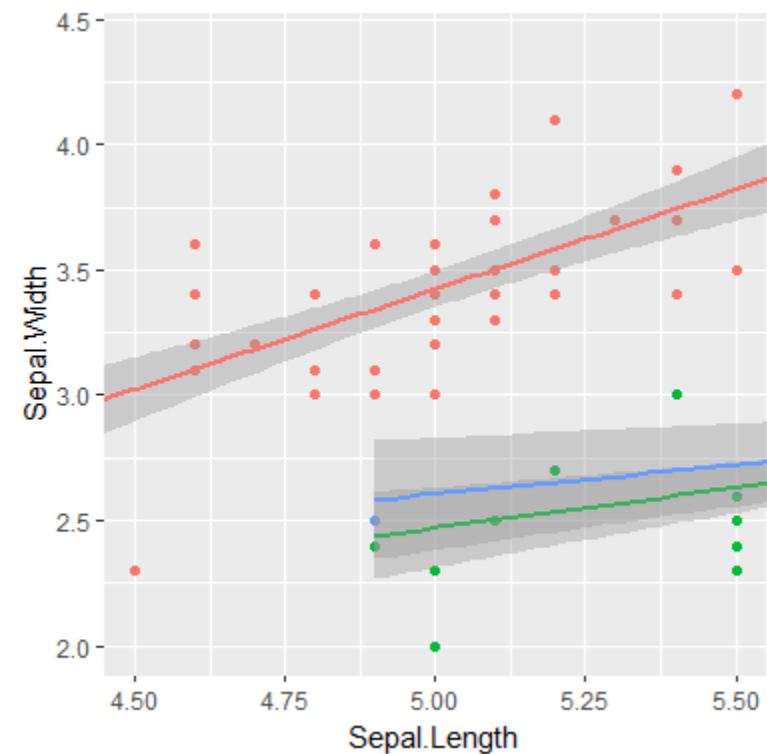
Aesthetics

Data



Species

- setosa
- versicolor
- virginica



Species

- setosa
- versicolor
- virginica

- All non-data ink
- Theme also an object – can set for series of graphs
- Inheritance

`element_text()`

text

title
 plot.title
 legend.title
 axis.title
 axis.title.x
 axis.title.y
 legend.text
 axis.text
 axis.text.x
 axis.text.y
 strip.text
 strip.text.x
 strip.text.y

`element_line()`

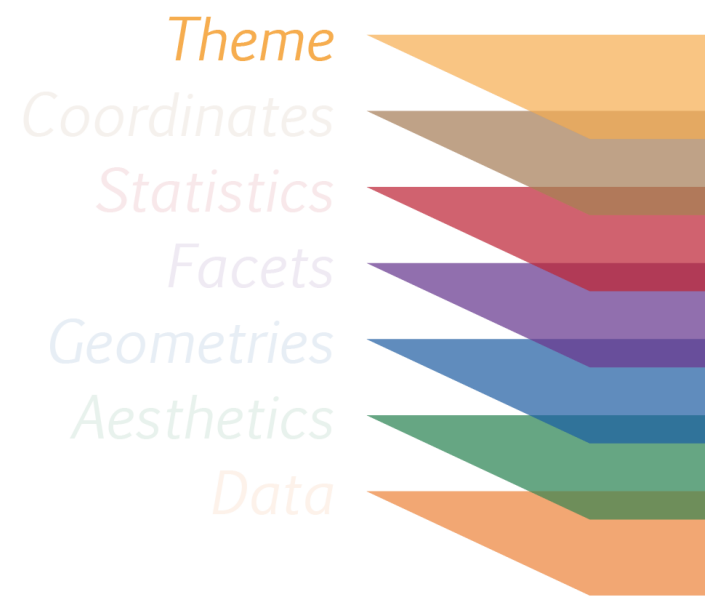
line

axis.ticks
 axis.ticks.x
 axis.ticks.y
 axis.line
 axis.line.x
 axis.line.y
 panel.grid
 panel.grid.major
 panel.grid.major.x
 panel.grid.major.y
 panel.grid.minor
 panel.grid.minor.x
 panel.grid.minor.y

`element_rect()`

rect

legend.background
 legend.key
 panel.background
 panel.border
 plot.background
 strip.background

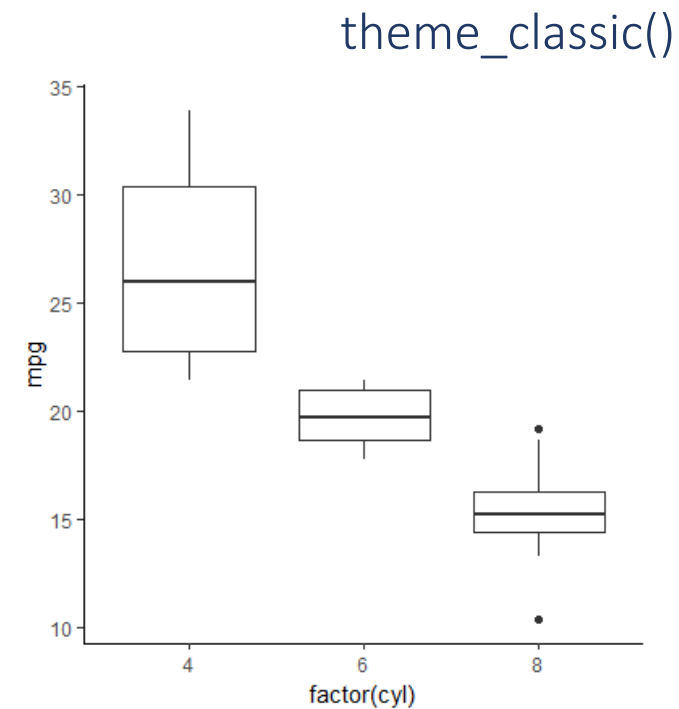
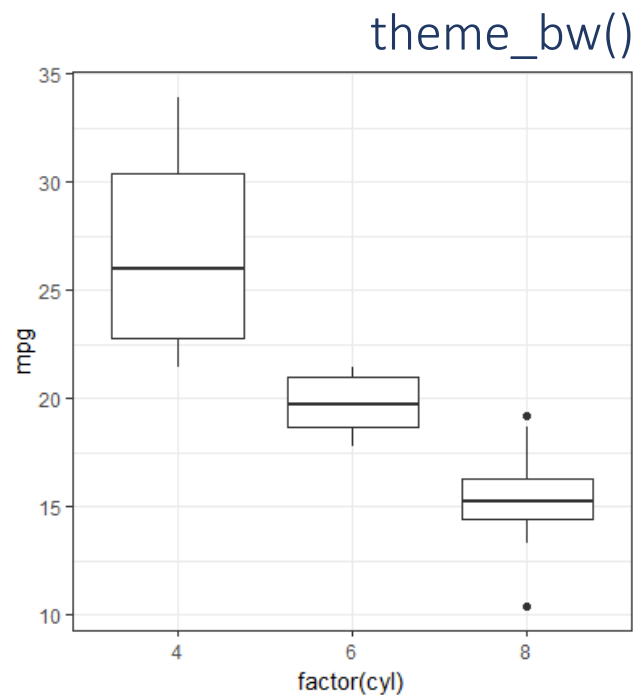
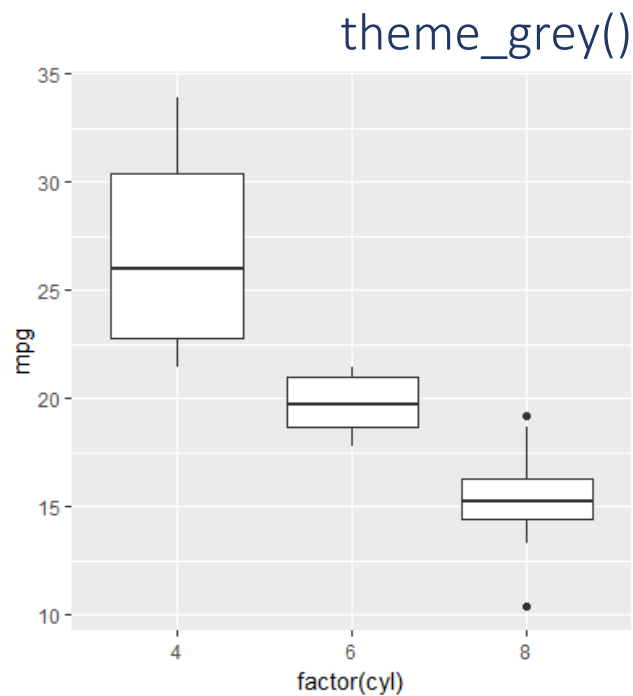
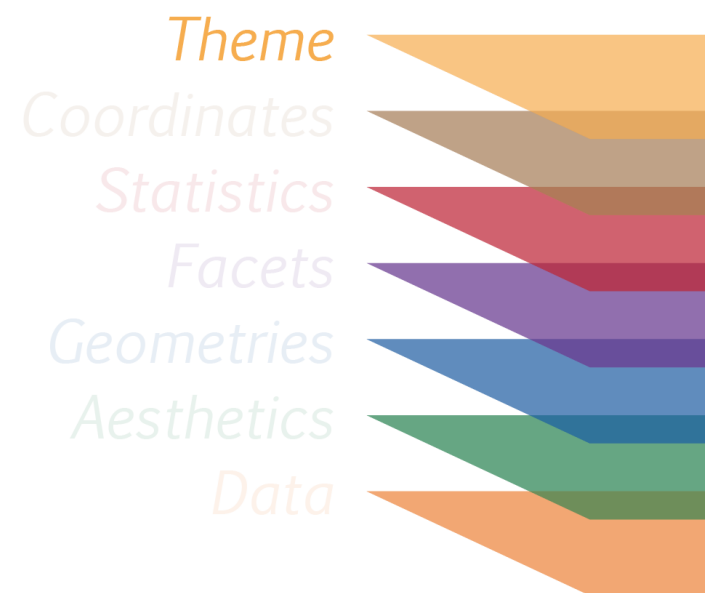


`element_blank()`

```
cars <- ggplot(mtcars, aes(x=factor(cyl), y=mpg)) +  
  geom_boxplot() +  
  theme_grey() #default
```

```
cars + theme_bw()
```

```
cars + theme_classic()
```



```

theme_bw <- function(base_size = 12) {
  structure(list(
    axis.line =      theme_blank(),
    axis.text.x =    theme_text(size = base_size * 0.8 , lineheight = 0.9, vjust = 1),
    axis.text.y =    theme_text(size = base_size * 0.8, lineheight = 0.9, hjust = 1),
    axis.ticks =     theme_segment(colour = "black", size = 0.2),
    axis.title.x =   theme_text(size = base_size, vjust = 1),
    axis.title.y =   theme_text(size = base_size, angle = 90, vjust = 0.5),
    axis.ticks.length = unit(0.3, "lines"),
    axis.ticks.margin = unit(0.5, "lines"),

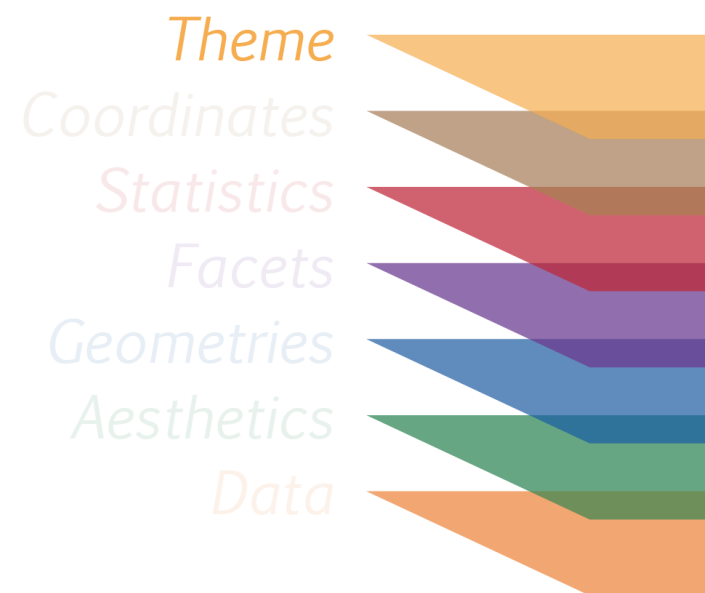
    legend.background = theme_rect(colour=NA),
    legend.key =       theme_rect(colour = "grey80"),
    legend.key.size =  unit(1.2, "lines"),
    legend.text =      theme_text(size = base_size * 0.8),
    legend.title =     theme_text(size = base_size * 0.8, face = "bold", hjust = 0),
    legend.position =  "right",

    panel.background = theme_rect(fill = "white", colour = NA),
    panel.border =     theme_rect(fill = NA, colour="grey50"),
    panel.grid.major = theme_line(colour = "grey90", size = 0.2),
    panel.grid.minor = theme_line(colour = "grey98", size = 0.5),
    panel.margin =     unit(0.25, "lines"),

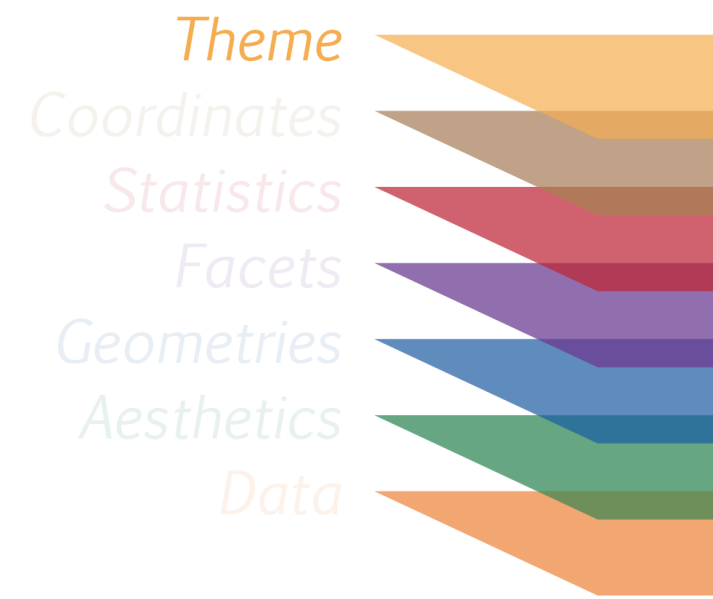
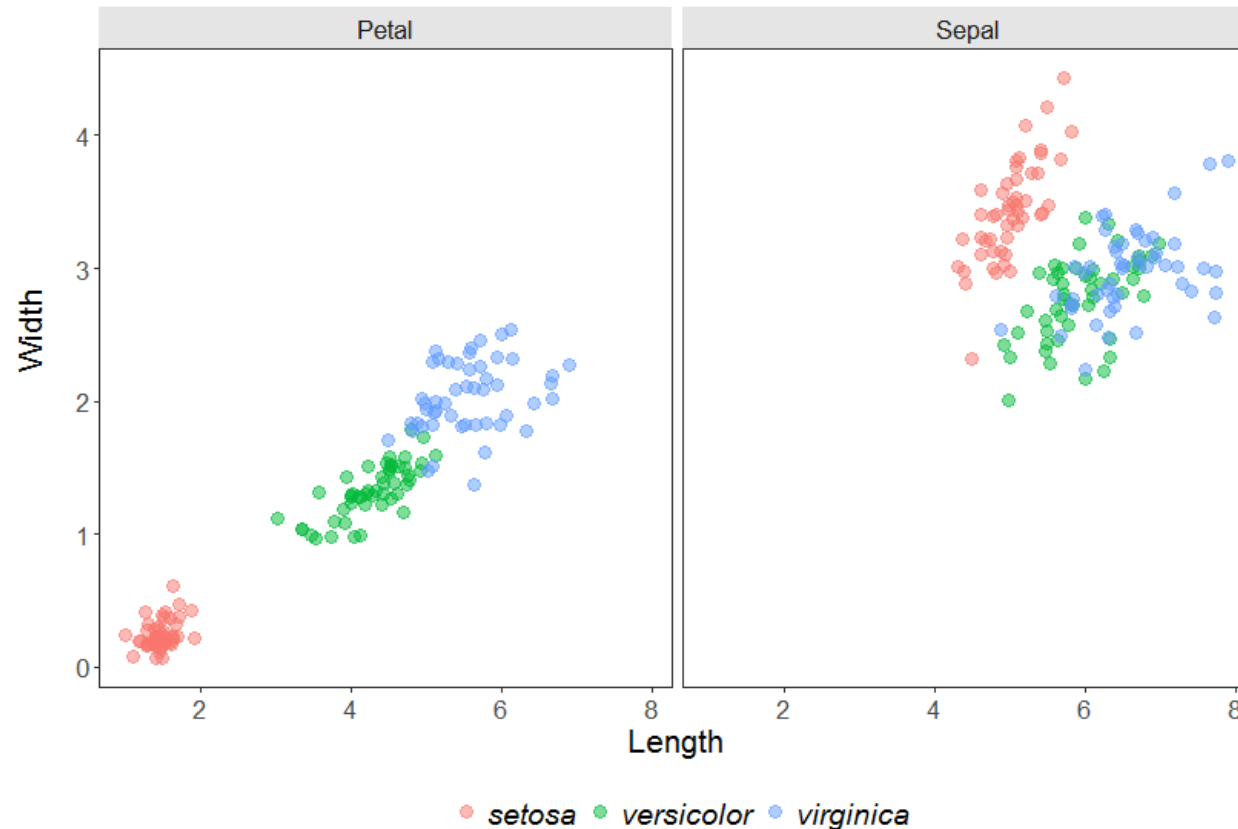
    strip.background = theme_rect(fill = "grey80", colour = "grey50"),
    strip.text.x =     theme_text(size = base_size * 0.8),
    strip.text.y =     theme_text(size = base_size * 0.8, angle = -90),

    plot.background =  theme_rect(colour = NA),
    plot.title =       theme_text(size = base_size * 1.2),
    plot.margin =      unit(c(1, 1, 0.5, 0.5), "lines")
  ), class = "options")
}

```



```
ggplot(iris.wide, aes(x=Length, y=width, col=Species)) +
  geom_jitter(size = 3, alpha = 0.5) +
  facet_grid(.~Part) +
  theme_bw() +
  theme(panel.grid = element_blank(),
        axis.line = element_blank(),
        text = element_text(size = 16),
        legend.title = element_blank(),
        legend.position = "bottom",
        legend.key.size = unit(1.5, "lines"),
        legend.text = element_text(face = "italic", size = 14),
        strip.background = element_rect(fill = "grey90", colour = NA),
        axis.title.y = element_text(hjust=0.57, margin = margin(r = 15)))
```



- <https://ggplot2.tidyverse.org/index.html> (cheat sheet and full listing)
- <https://r4ds.had.co.nz/data-visualisation.html> (R for Data Science book)
- <http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf> (ggplot2 colours)

Data Visualization with ggplot2 :: CHEAT SHEET



Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data set**, a **coordinate system**, and **geoms**—visual marks that represent data points.



To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.



Complete the template below to build a graph.

```
ggplot(data = <DATA>) +  
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),  
  stat = <STAT>, position = <POSITION>) +  
  <COORDINATE_FUNCTION> +  
  <FACET_FUNCTION> +  
  <SCALE_FUNCTION> +  
  <THEME_FUNCTION>
```

ggplot(data = mpg, aes(x = cty, y = hwy)) Begins a plot that you finish by adding layers to. Add one geom function per layer.

qplot(x = cty, y = hwy, data = mpg, geom = "point") Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

last_plot() Returns the last plot

ggsave("plot.png", width = 5, height = 5) Saves last plot as 5" x 5" file named "plot.png" in working directory. Matches file type to file extension.

Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

a <- ggplot(economics, aes(data, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))

a + geom_blank()
(Useful for expanding limits)

b + geom_curve(aes(yend = lat + 1,
xend = long + 1, curvature = z)) ~ x, xend, y, yend,
alpha, angle, color, curvature, linetype, size

a + geom_path(linetype = "butt", linejoin = "round",
linemitre = 1)
x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(group = group))
x, y, alpha, color, fill, group, linetype, size

b + geom_rect(aes(xmin = long, ymin = lat, xmax =
long + 1, ymax = lat + 1)) ~ x, xmax, xmin, ymax,
ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin = unemploy - 900,
ymax = unemploy + 900)) ~ x, xmax, xmin,
alpha, color, fill, group, linetype, size

LINE SEGMENTS

Common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept = 0, slope = 1))
b + geom_hline(aes(yintercept = lat))
b + geom_vline(aes(xintercept = long))

b + geom_segment(aes(yend = lat + 1, xend = long + 1))
b + geom_spoke(aes(angle = 1:155, radius = 1))

ONE VARIABLE continuous

c <- ggplot(mpg, aes(hwy))
c2 <- ggplot(mpg)

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian")
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly() x, y, alpha, color, group,
linetype, size

c + geom_histogram(binwidth = 5) x, y, alpha,
color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy)) x, y, alpha,
color, fill, linetype, size, weight

discrete

d <- ggplot(mpg, aes(R))

d + geom_bar()
x, alpha, color, fill, linetype, size, weight

TWO VARIABLES

continuous x, continuous y
e <- ggplot(mpg, aes(cty, hwy))

e + geom_label(aes(label = cty, nudge_x = 1,
nudge_y = 1, check_overlap = TRUE)) x, y, label,
alpha, angle, color, family, fontface, hjust,
lineheight, size, vjust

e + geom_jitter(height = 2, width = 2)
x, y, alpha, color, fill, shape, size

e + geom_point() x, y, alpha, color, fill, shape,
size, stroke

e + geom_quantile() x, y, alpha, color, group,
linetype, size, weight

e + geom_rug(sides = "bl") x, y, alpha, color,
linetype, size

e + geom_smooth(method = lm) x, y, alpha,
color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty, nudge_x = 1,
nudge_y = 1, check_overlap = TRUE)) x, y, label,
alpha, angle, color, family, fontface, hjust,
lineheight, size, vjust

discrete x, continuous y
f <- ggplot(mpg, aes(class, hwy))

f + geom_col() x, y, alpha, color, fill, group,
linetype, size

f + geom_boxplot() x, y, lower, middle, upper,
ymax, ymin, alpha, color, fill, group, linetype,
shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir =
"center") x, y, alpha, color, fill, group

f + geom_violin(scale = "area") x, y, alpha, color,
fill, group, linetype, size, weight

discrete x, discrete y

g <- ggplot(diamonds, aes(cut, color))

g + geom_count() x, y, alpha, color, fill, shape,
size, stroke

THREE VARIABLES

sealsSz <- with(seals, sqrt(delta_long^2 + delta_lat^2))
l <- ggplot(seals, aes(long, lat))

l + geom_contour(aes(z = z))
x, y, z, alpha, color, group, linetype,
size, weight

continuous bivariate distribution
h <- ggplot(diamonds, aes(carat, price))

h + geom_bin2d(binwidth = c(0.25, 500))
x, y, alpha, color, fill, linetype, size, weight

h + geom_density2d()
x, y, alpha, color, group, linetype, size

h + geom_hex()
x, y, alpha, color, fill, size

continuous function

i <- ggplot(economics, aes(data, unemploy))

i + geom_area()
x, y, alpha, color, fill, linetype, size

i + geom_line()
x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv")
x, y, alpha, color, group, linetype, size

visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4.5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se))

j + geom_crossbar(fatten = 2)
x, y, ymax, ymin, alpha, color, fill, group, linetype,
size

j + geom_errorbar() x, ymax, ymin, alpha, color,
group, linetype, size, width [also
geom_errorbarh()]

j + geom_linerange() x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange() x, y, ymin, ymax, alpha, color, fill, group, linetype,
shape, size

maps

data <- data.frame(murder = USArrests\$Murder,
state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))

k + geom_map(aes(map_id = state), map = map)
+ expand_limits(x = map\$long, y = map\$lat),
map_id, alpha, color, fill, linetype, size

l + geom_raster(aes(fill = z), hjust = 0.5, vjust = 0.5,
interpolate = FALSE)
x, y, alpha, fill

l + geom_tile(aes(fill = z), x, y, alpha, color, fill,
linetype, size, width



R ggplot2 ...

and don't forget about



SEEC Stats Toolbox Schedule 2019



Date	Topic	Speaker
28 February	ggplot2 – the grammar of graphics	Kirsten Packer
28 March	Population status assessment tools	Henning Winker
25 April	Meta-analysis	Vernon Visser
30 May	Time series analyses	Birgit Erni
25 July	Cloud computing and R with Amazon Web Services (AWS)	Ian Durbach
29 August	Spatial interpolation	Mzabalazo Ngwenya
26 September	R Markdown and Leaflet	Dominic Henry
31 October	Multidimensional Scaling (MDS)	Natasha Karenzi
28 November	Dynamic Occupancy Models	Res Altwegg