R Graphs

Pilsung Kang

School of Industrial Management Engineering

Korea University

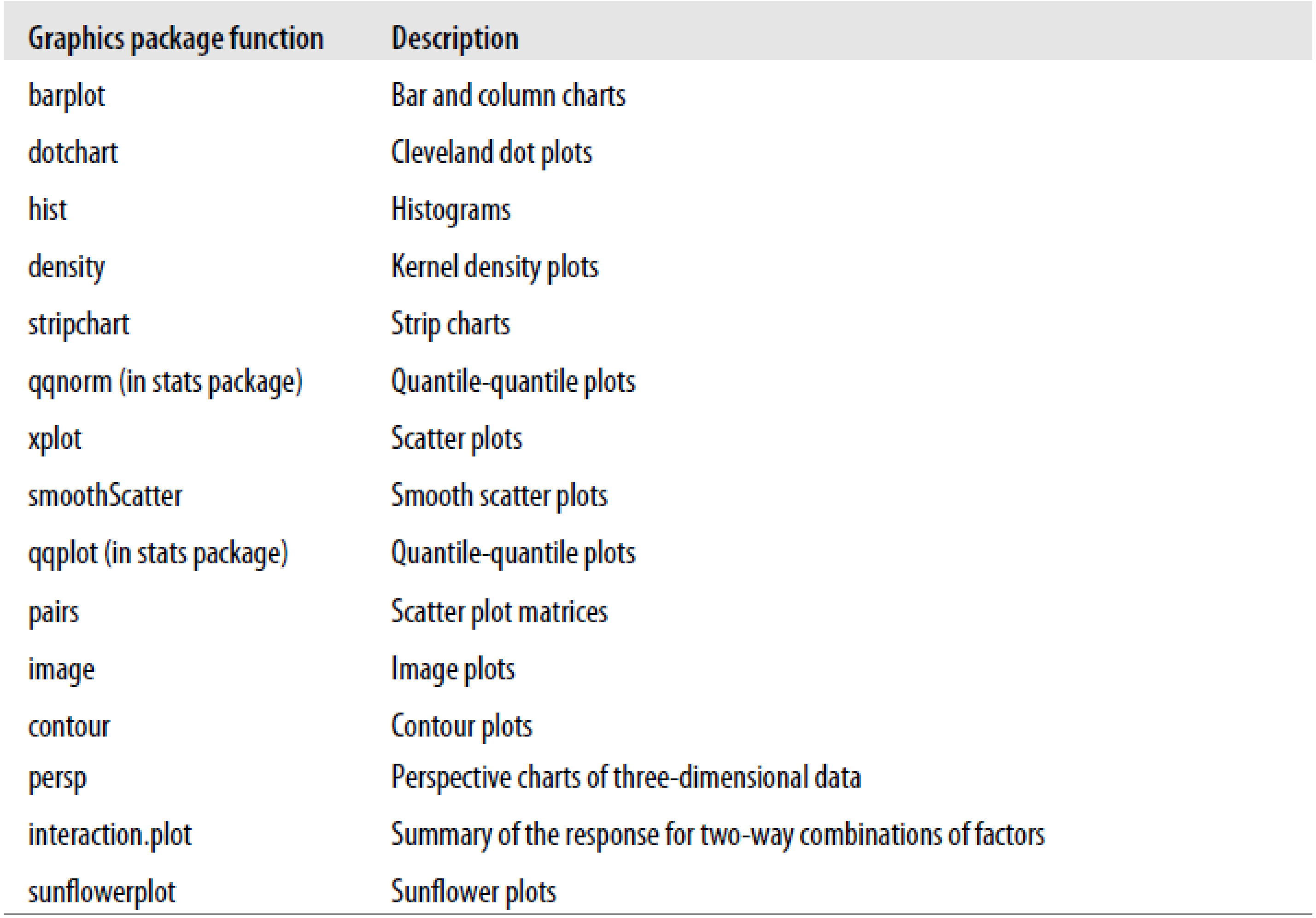
AGENDA

01 R Graph Basics

02 R Graphs with ggplot2 package

R Graphics

- Basic functions that are provided by "graphics" package



Graphics package function	Description
barplot	Bar and column charts
dotchart	Cleveland dot plots
hist	Histograms
density	Kernel density plots
stripchart	Strip charts
qqnorm (in stats package)	Quantile-quantile plots
xplot	Scatter plots
smoothScatter	Smooth scatter plots
qqplot (in stats package)	Quantile-quantile plots
pairs	Scatter plot matrices
image	Image plots
contour	Contour plots
persp	Perspective charts of three-dimensional data
interaction.plot	Summary of the response for two-way combinations of factors
sunflowerplot	Sunflower plots

R Graphics

- Fisher’s Iris dataset (default dataset provided by R)
  - ✓ Five variables
    - sepal length in cm, sepal width in cm, petal length in cm, petal width in cm, and
    - Species : Iris Setosa, Iris Versicolour, and Iris Virginica.

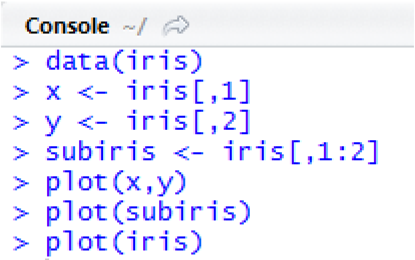
Setosa

Versicolor

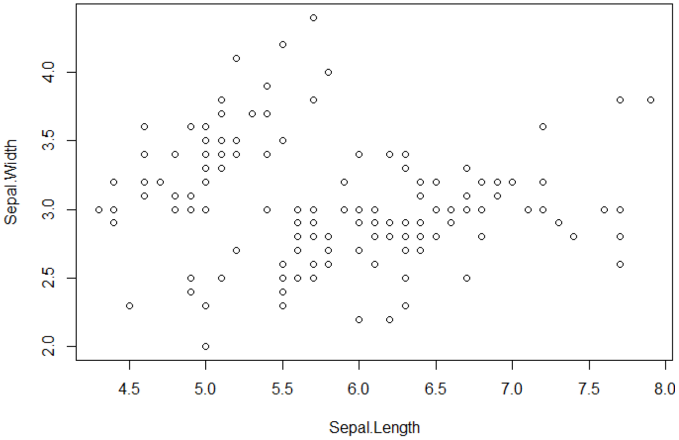
Virginica

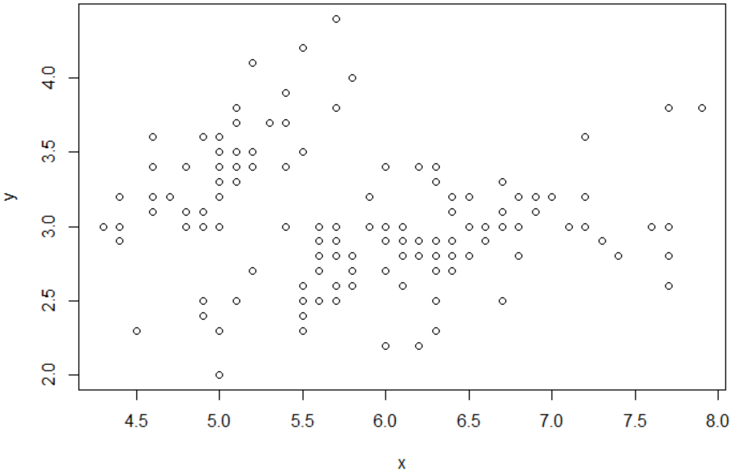
R Graphics

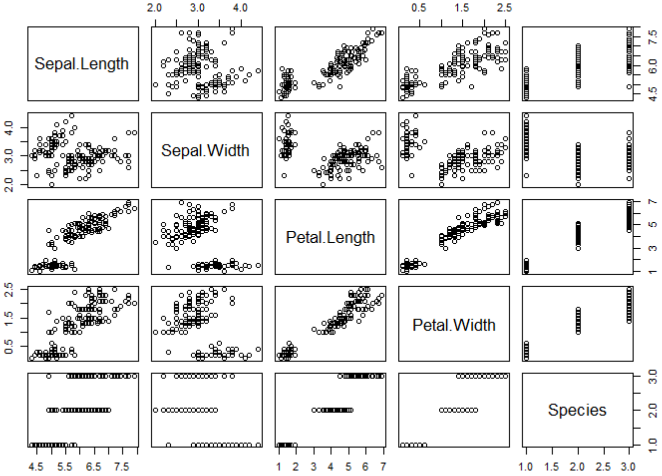
- Polymorphism of R graph functions
  - ✓ polymorphic function: has different operations for different arguments
  - ✓ ex: plot( )



```
> data(iris)
> x <- iris[,1]
> y <- iris[,2]
> subiris <- iris[,1:2]
> plot(x,y)
> plot(subiris)
> plot(iris)
```



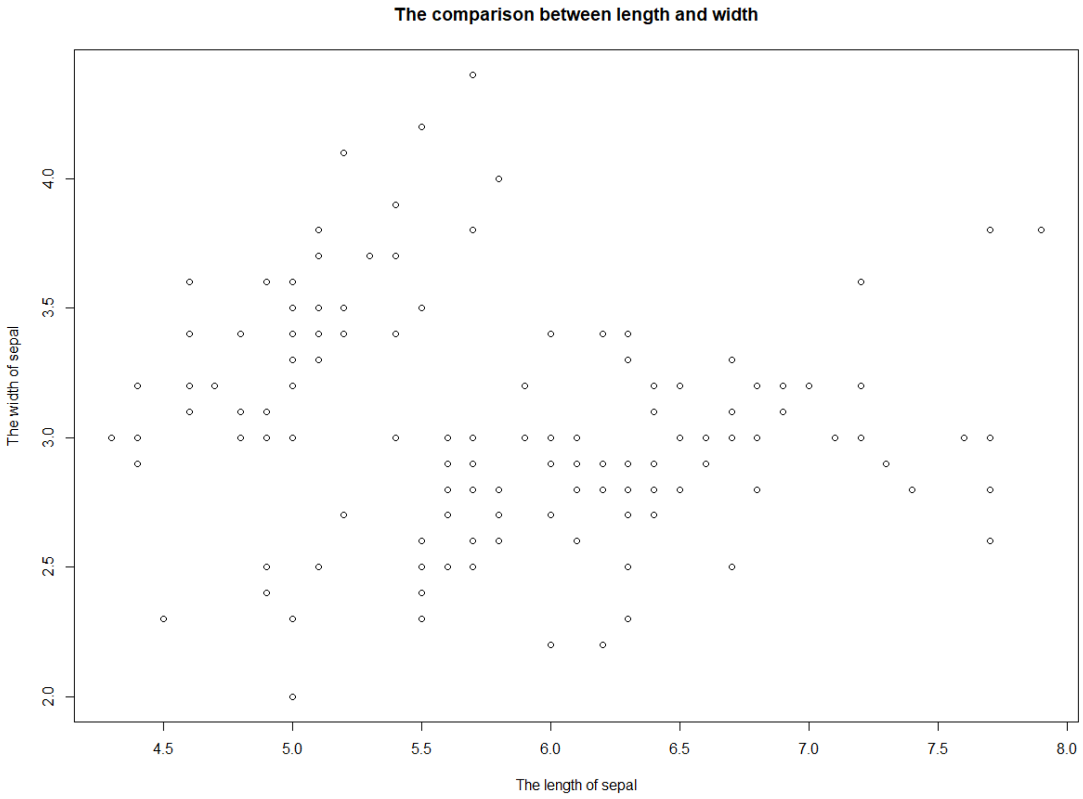




R Graphics

- Titles and labels in a graph
  - ✓ title: main, x-axis label: xlab, y-axis label: ylab

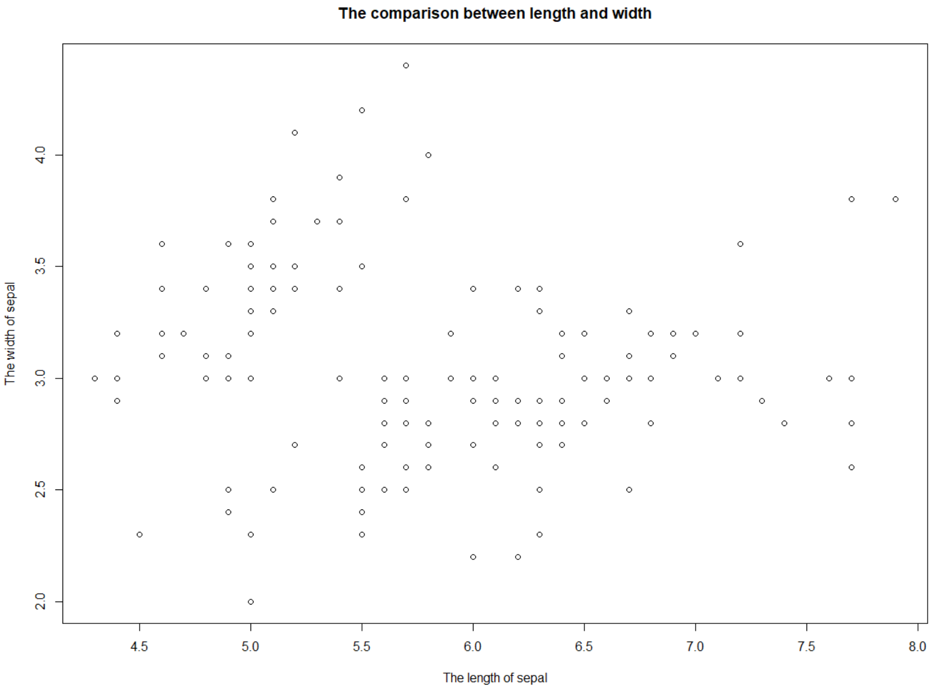
```
# Add title and x,y labels
plot(subiris, main="The comparison between length and width",
         xlab = "The length of sepal", ylab = "The width of sepal")
```

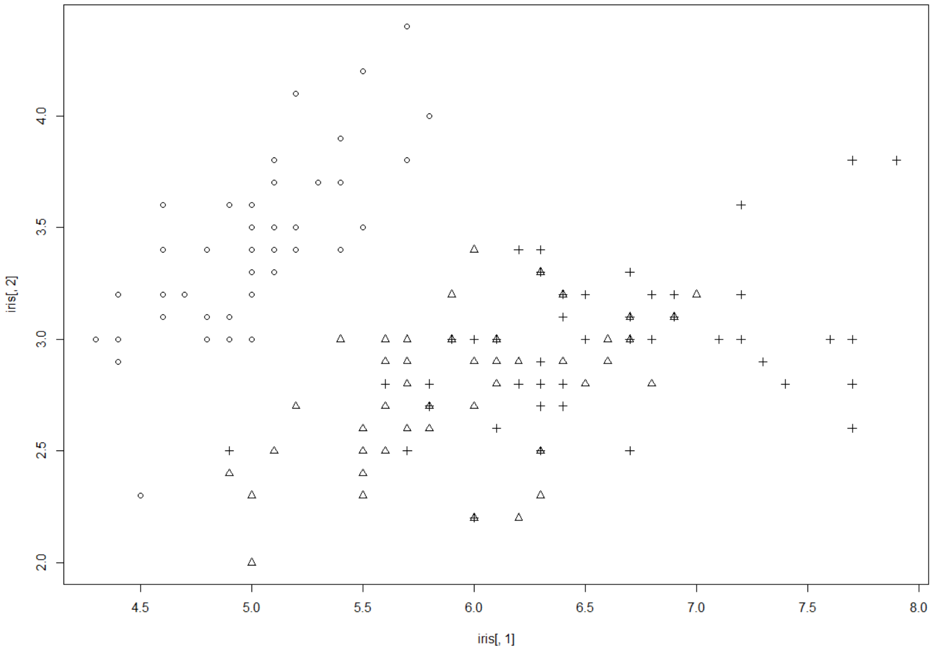


R Graphics

- Some options for basic R graphs
  - ✓ pch: shape, cex: size, col: color

```
# Scatter plot with different shapes for different classes
plot(iris[,1],iris[,2],pch=as.integer(iris[,5]))
```

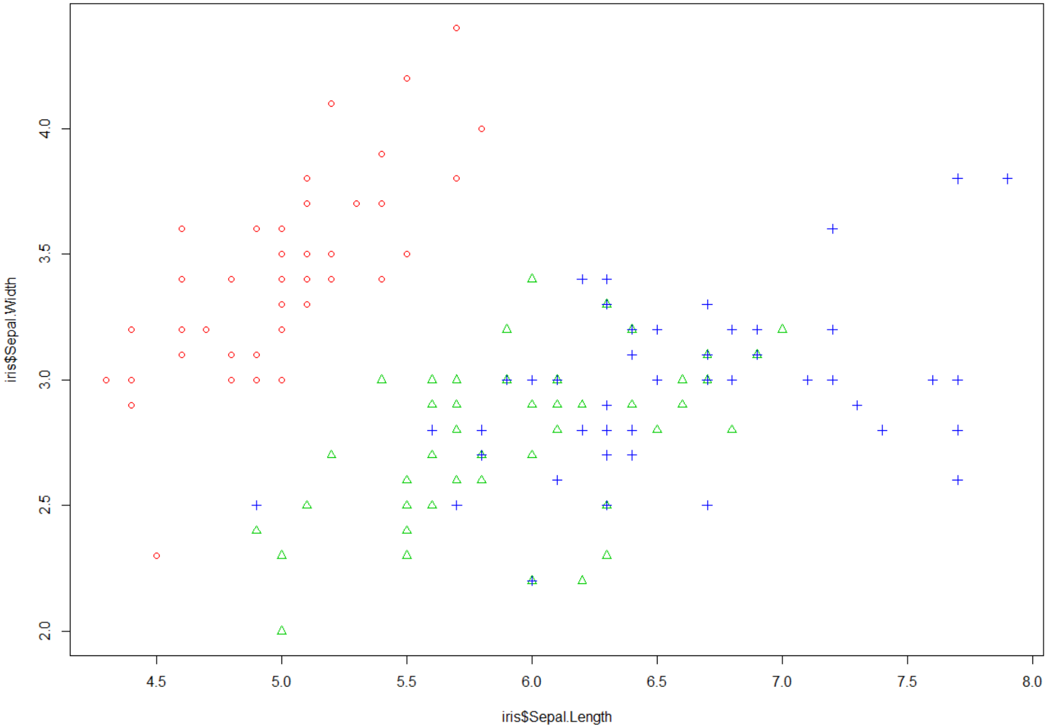




R Graphics

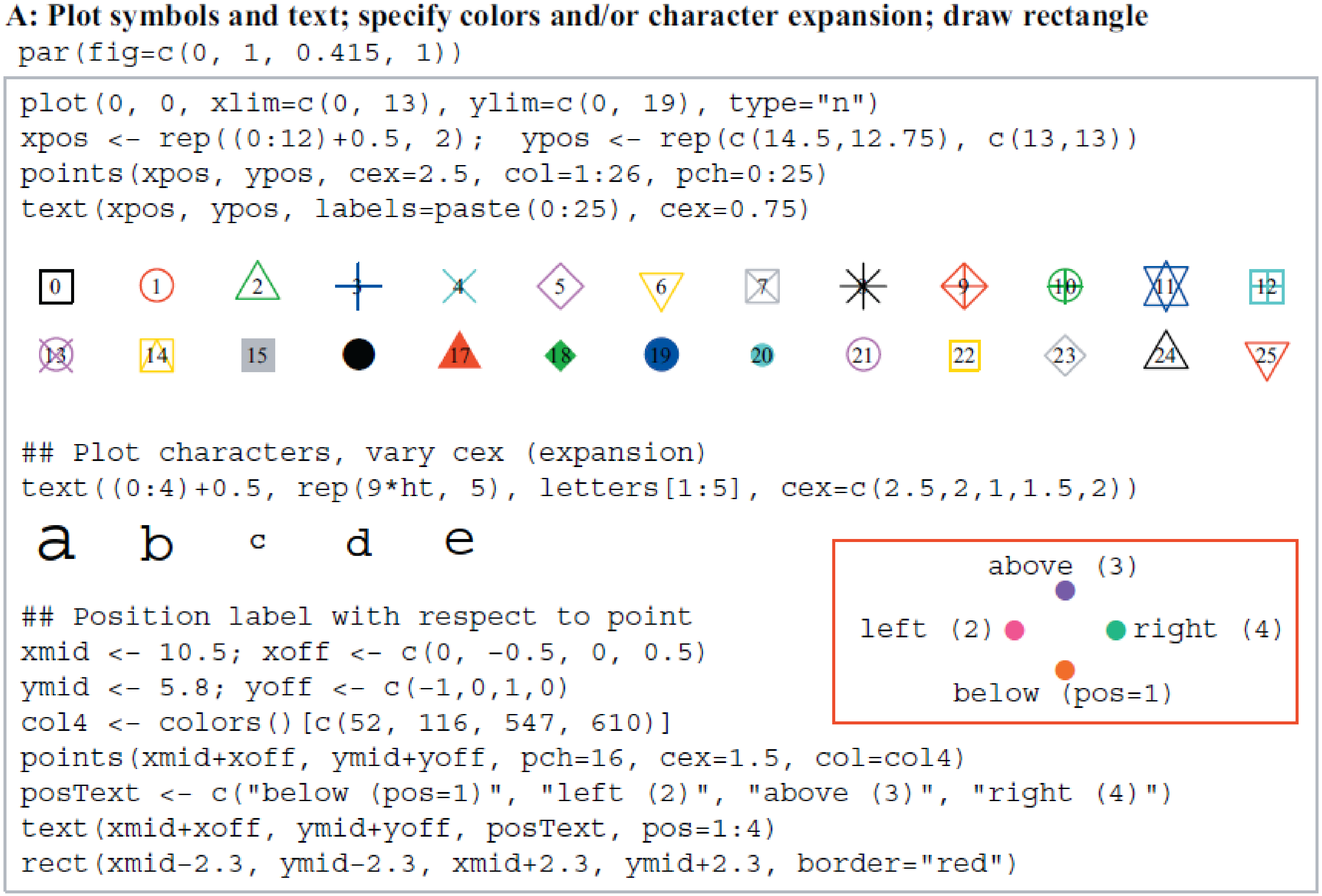
- Symbols in graphs
  - ✓ pch: shape, cex: size, col: color

```
# Scatter plot with different shapes & colors for different classes
plot(iris$Sepal.Length,iris$Sepal.Width,
     pch=as.integer(iris$Species),col=as.integer(iris$Species)+1)
```



R Graphics

- Options for better readability



A: Plot symbols and text; specify colors and/or character expansion; draw rectangle

```
par(fig=c(0, 1, 0.415, 1))
```

```
plot(0, 0, xlim=c(0, 13), ylim=c(0, 19), type="n")
xpos <- rep((0:12)+0.5, 2);  ypos <- rep(c(14.5,12.75), c(13,13))
points(xpos, ypos, cex=2.5, col=1:26, pch=0:25)
text(xpos, ypos, labels=paste(0:25), cex=0.75)
```

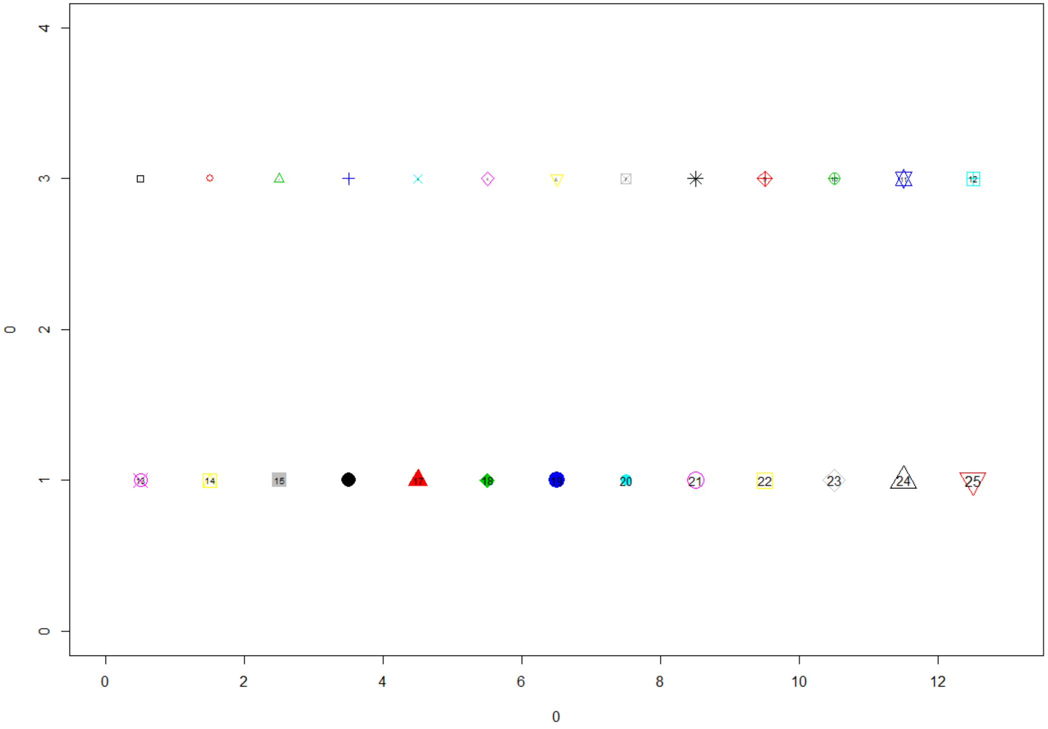
```
## Plot characters, vary cex (expansion)
text((0:4)+0.5, rep(9*ht, 5), letters[1:5], cex=c(2.5,2,1,1.5,2))
```

```
## Position label with respect to point
xmid <- 10.5; xoff <- c(0, -0.5, 0, 0.5)
ymid <- 5.8; yoff <- c(-1,0,1,0)
col4 <- colors()[c(52, 116, 547, 610)]
points(xmid+xoff, ymid+yoff, pch=16, cex=1.5, col=col4)
posText <- c("below (pos=1)", "left (2)", "above (3)", "right (4)")
text(xmid+xoff, ymid+yoff, posText, pos=1:4)
rect(xmid-2.3, ymid-2.3, xmid+2.3, ymid+2.3, border="red")
```

R Graphics

- Options for better readability

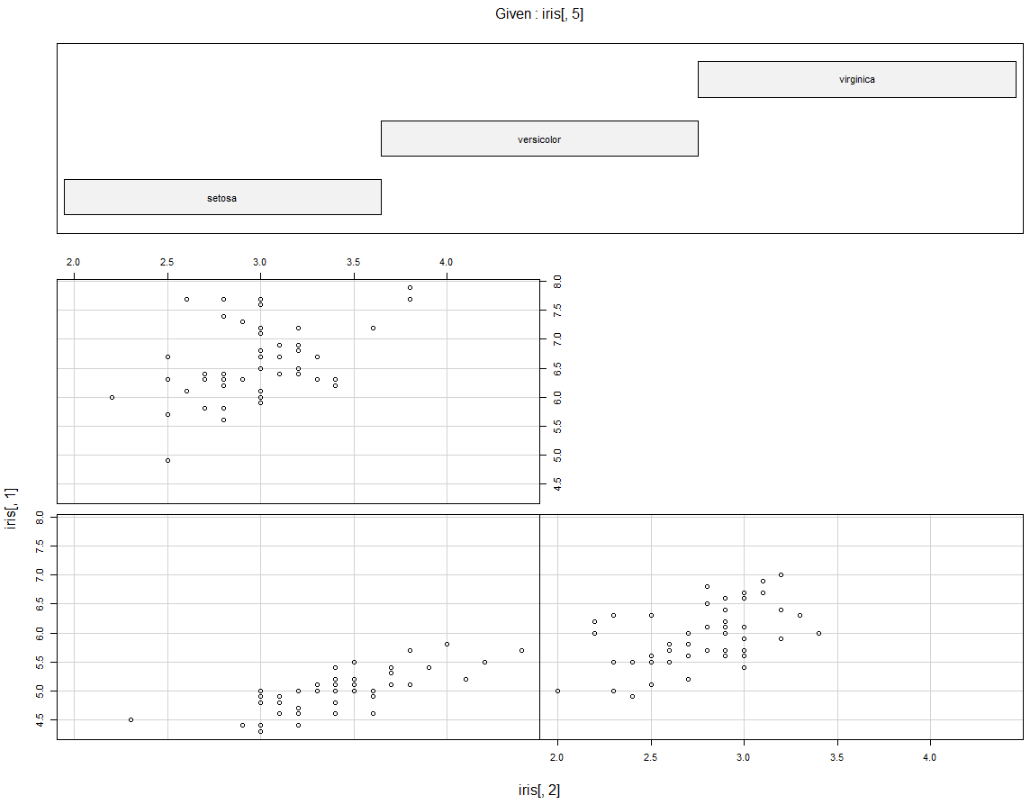
```
# Predefined shapes and colors
plot(0,0, xlim=c(0,13), ylim=c(0,4), type="n")
xpos <- rep((0:12)+0.5,2)
ypos <- rep(c(3,1), c(13,13))
points(xpos, ypos, cex=seq(from=1,to=3,length=26), col=1:26, pch=0:25)
text(xpos, ypos, labels = paste(0:25), cex=seq(from=0.1,to=1,length=26))
```



R Graphics

- Conditional plot
  - ✓ coplot(y ~ x | f)
  - ✓ For every f values, draw scatter plot for x and y

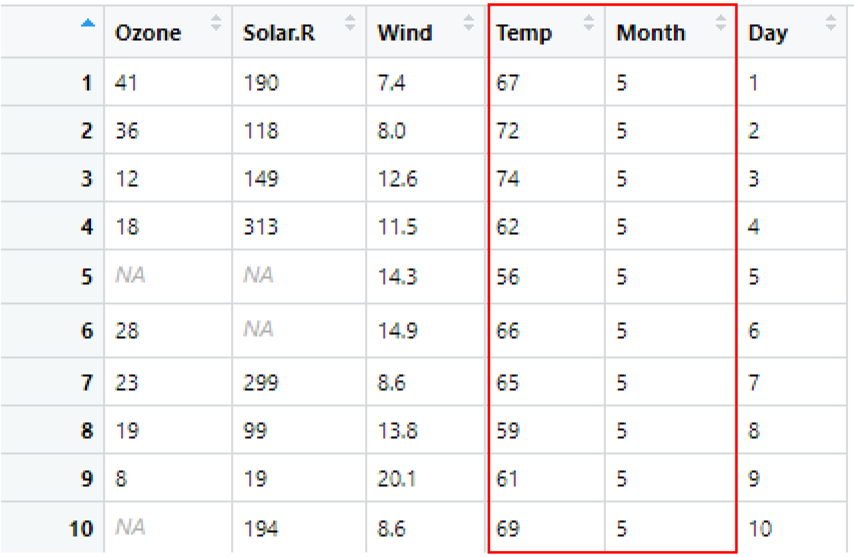
```
# Conditional plot
coplot(iris[,1]~iris[,2] | iris[,5])
```



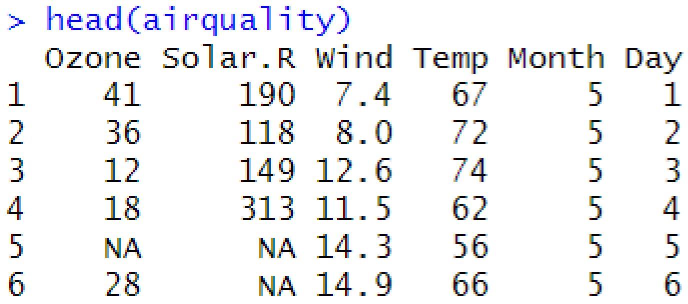
R Graphics

- Histogram
  - ✓ Example dataset: air quality embedded in R base

```
# Histogram
data(airquality)
head(airquality)
```



	Ozone	Solar.R	Wind	Temp	Month	Day
1	41	190	7.4	67	5	1
2	36	118	8.0	72	5	2
3	12	149	12.6	74	5	3
4	18	313	11.5	62	5	4
5	NA	NA	14.3	56	5	5
6	28	NA	14.9	66	5	6
7	23	299	8.6	65	5	7
8	19	99	13.8	59	5	8
9	8	19	20.1	61	5	9
10	NA	194	8.6	69	5	10

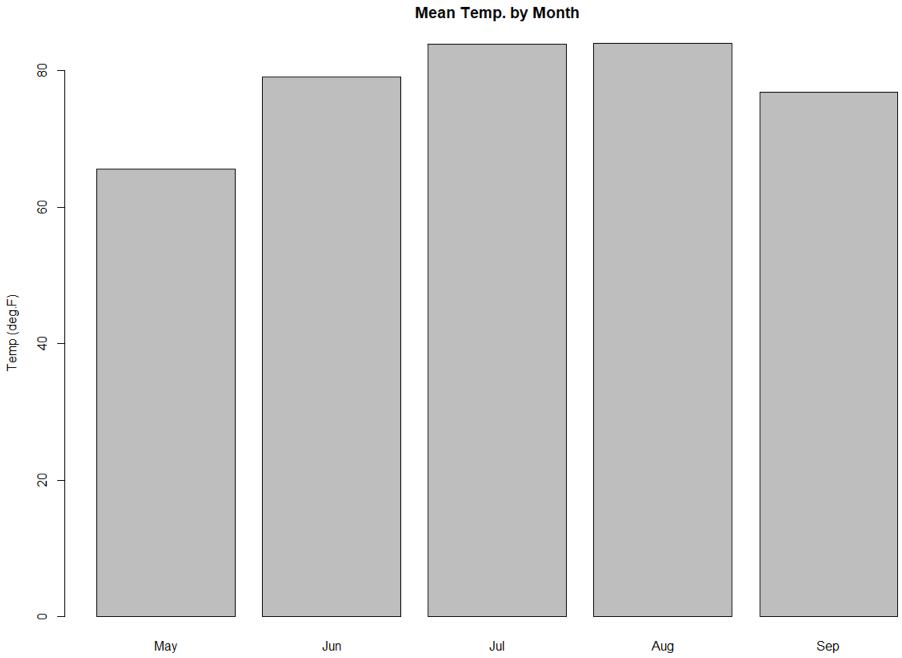


```
> head(airquality)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
```

R Graphics

- Histogram
  - ✓ Average temperature for each month

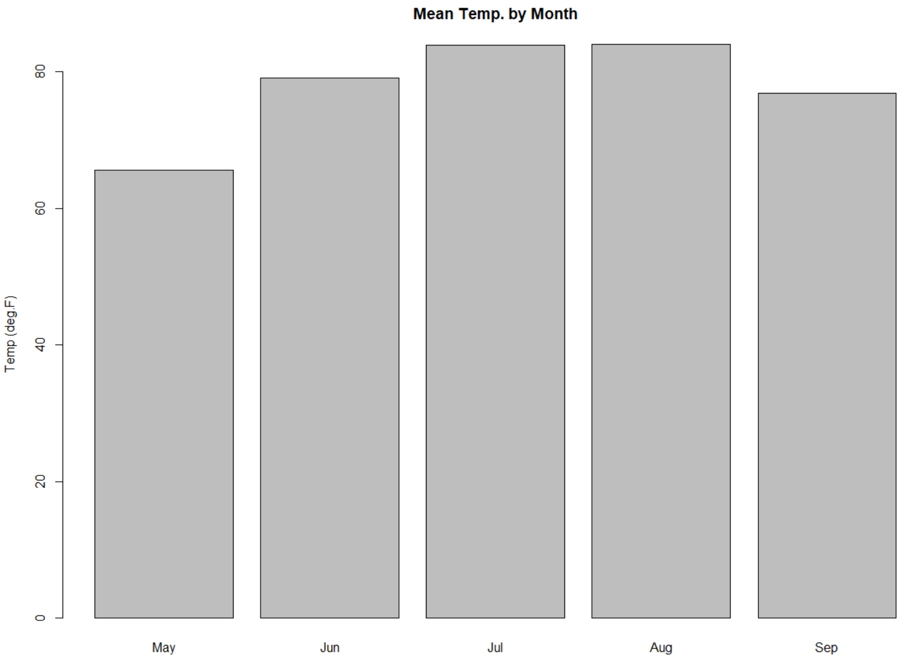
```
heights <- tapply(airquality$Temp, airquality$Month, mean)
barplot(heights)
barplot(heights, main="Mean Temp. by Month",
        names.arg = c("May", "Jun", "Jul", "Aug", "Sep"),
        ylab = "Temp (deg.F)")
```



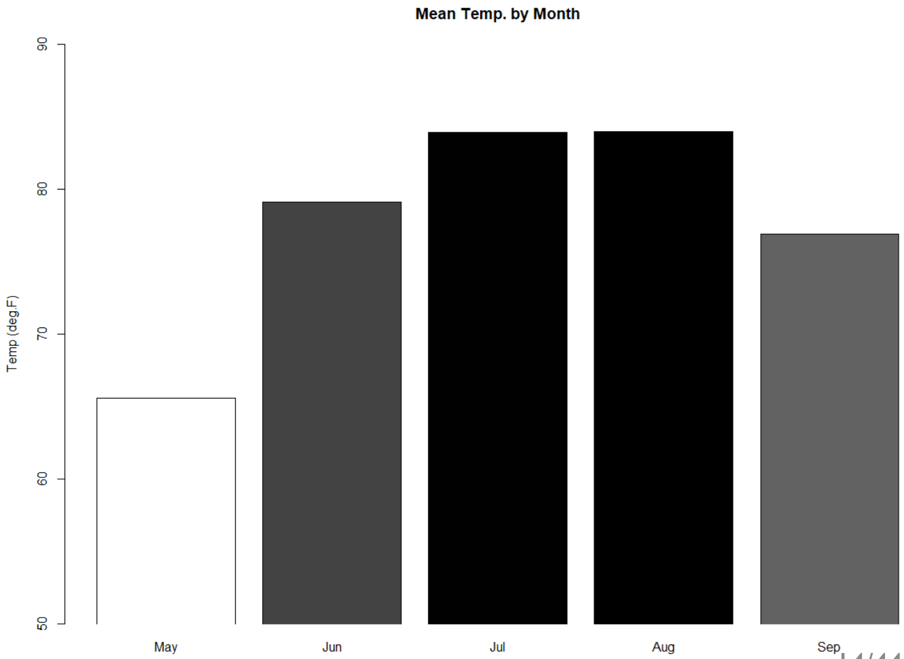
R Graphics

- Histogram
  - ✓ Make the histogram look better

```
# Histogram with more advanced options
rel.hts <-  (heights-min(heights))/(max(heights)-min(heights))
grays <- gray(1-rel.hts)
barplot(heights, col=grays, ylim=c(50,90), xpd=FALSE,
     main="Mean Temp. by Month",
     names.arg = c("May", "Jun", "Jul", "Aug", "Sep"), ylab = "Temp (deg.F)")
```



Mean Temp. by Month

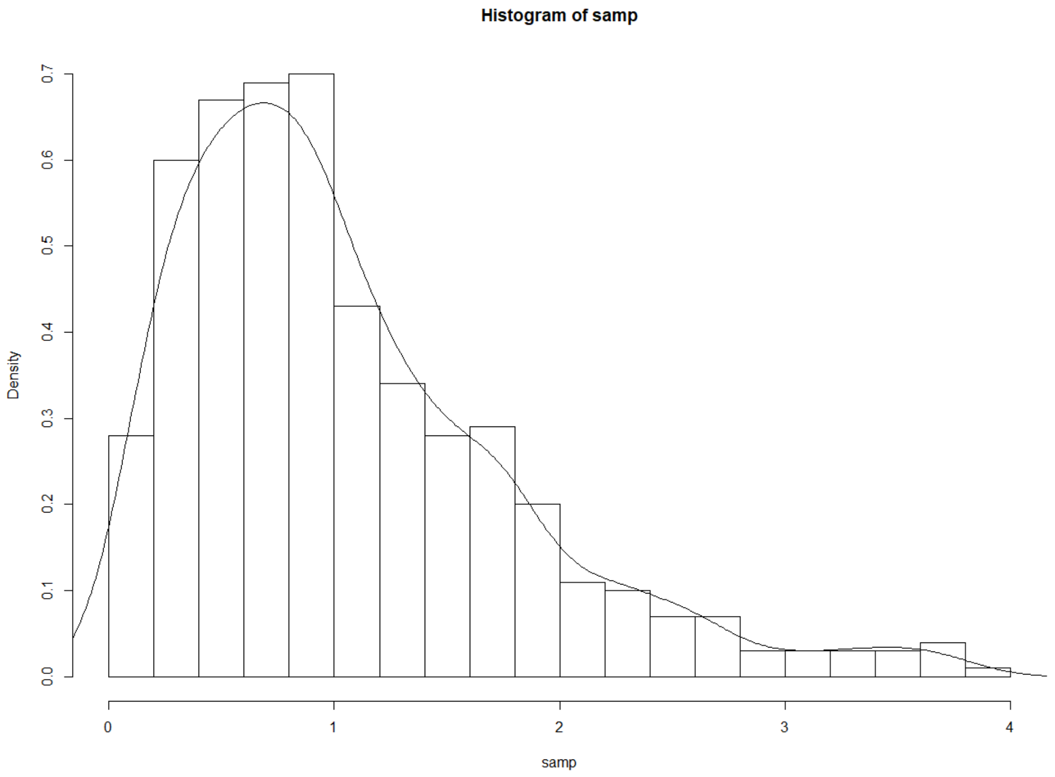


Mean Temp. by Month

R Graphics

- Histogram
    - ✓ Histogram with the estimated distribution

```
# Histogram with the estimated distribution
samp <- rgamma(500,2,2)
hist(samp, 20, prob=T)
lines(density(samp))
```



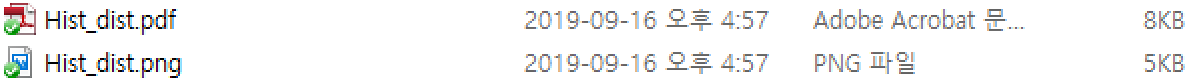
R Graphics

- Save the graph object

```r
# Save the plot as a png format
png("Hist_dist.png")
hist(samp, 20, prob=T)
lines(density(samp))
dev.off()

# Save the plot as a pdf format
pdf("Hist_dist.pdf")
hist(samp, 20, prob=T)
lines(density(samp))
dev.off()
```

✓ Look at the working directory and you will find the following two files



Hist_dist.pdf	2019-09-16 오후 4:57	Adobe Acrobat 문...	8KB
Hist_dist.png	2019-09-16 오후 4:57	PNG 파일	5KB

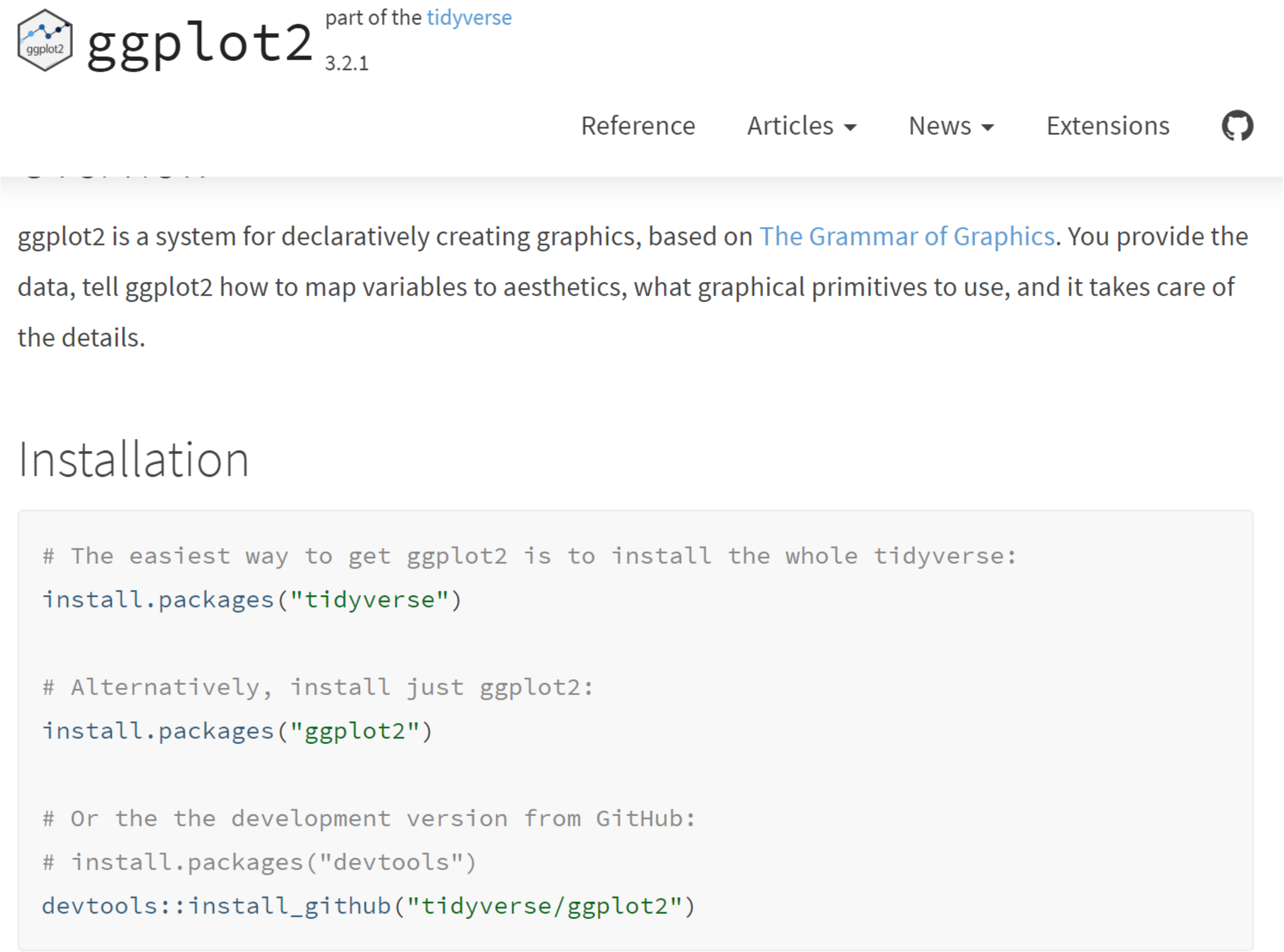
AGENDA

01 R Graph Basics

02 R Graphs with ggplot2 package

R Graphs with ggplot2 package

- ggplot2 package
  - ✓ https://ggplot2.tidyverse.org/index.html



ggplot2 is a system for declaratively creating graphics, based on The Grammar of Graphics. You provide the data, tell ggplot2 how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details.

Installation

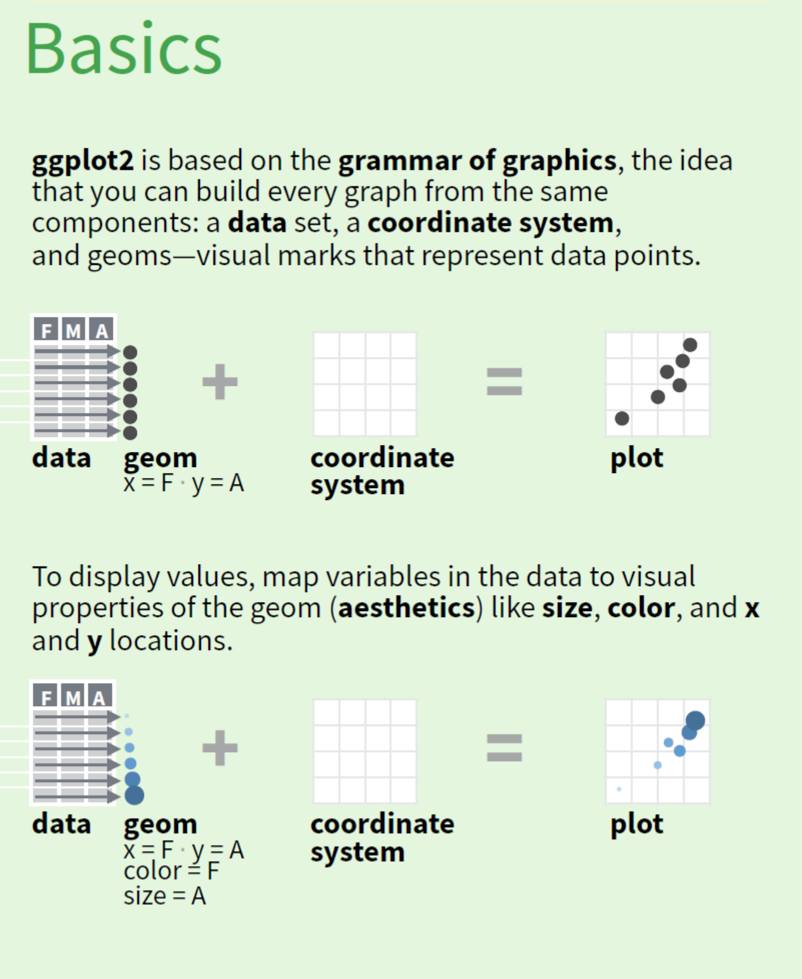
```
# The easiest way to get ggplot2 is to install the whole tidyverse:
install.packages("tidyverse")

# Alternatively, install just ggplot2:
install.packages("ggplot2")

# Or the the development version from GitHub:
# install.packages("devtools")
devtools::install_github("tidyverse/ggplot2")
```

R Graphs with ggplot2 package

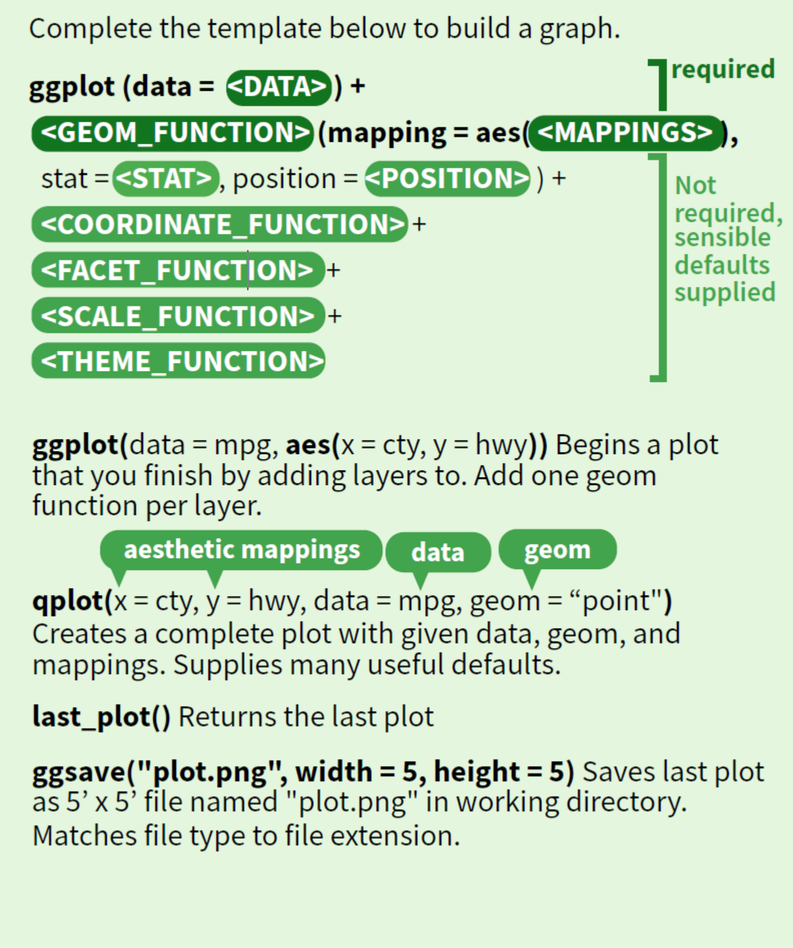
- ggplot2 Basics



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>
```

required
Not required, sensible defaults supplied

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

aesthetic mappings
data
geom

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.

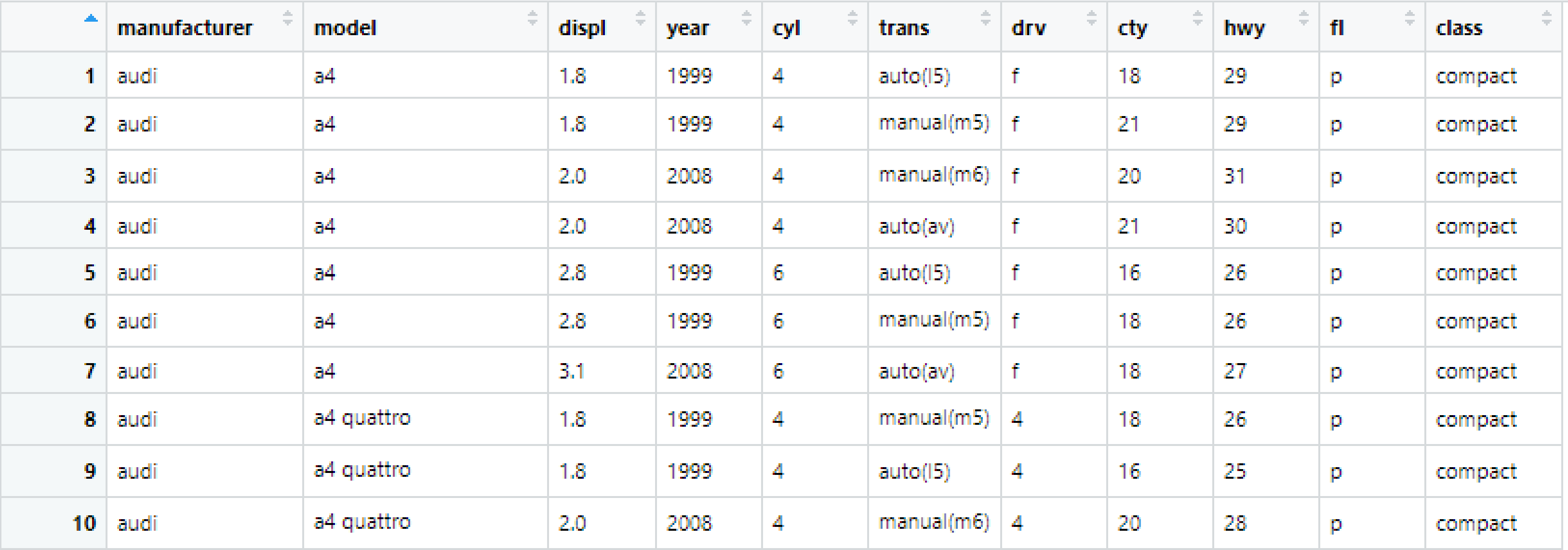
R Graphics

- Install and activate the package

```
# ggplot2: make r graphs more beautiful and informative
install.packages("ggplot2")
library(ggplot2)
```

- Load a sample dataset

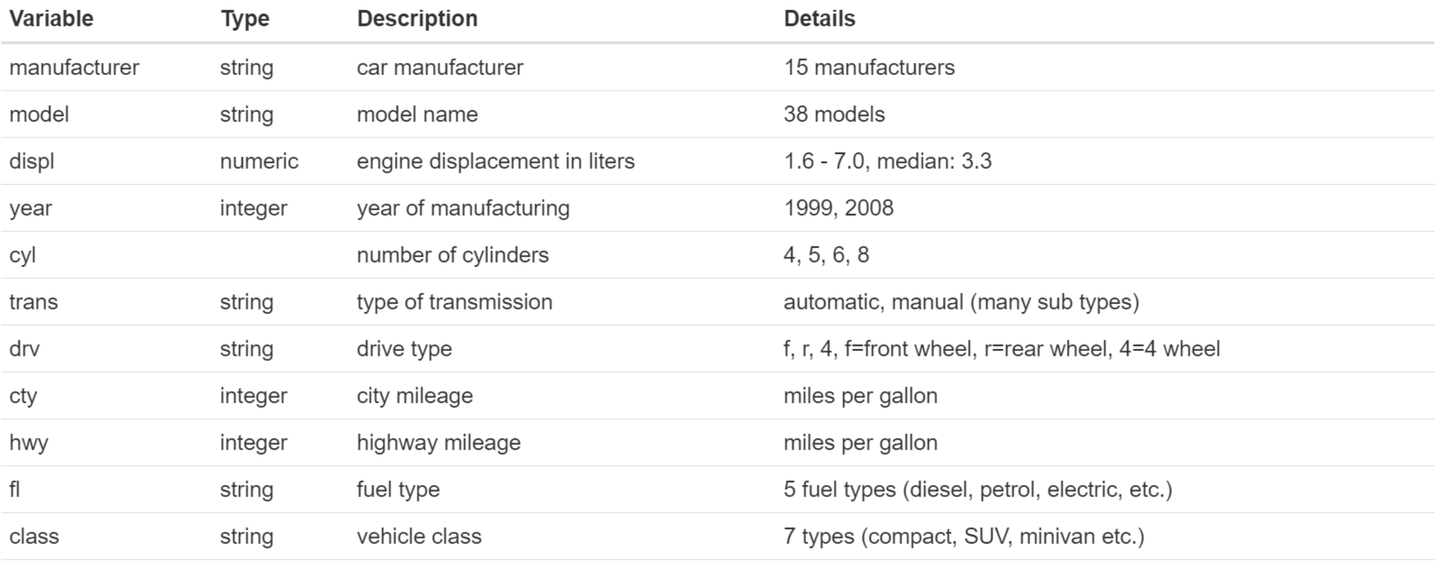
```
# Load a sample dataset
data("mpg")
head(mpg)
str(mpg)
```



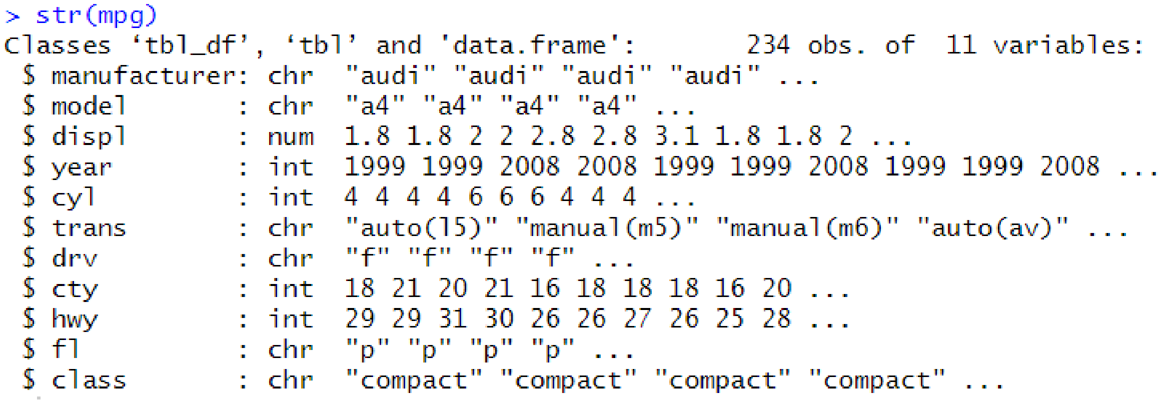
	manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	class
1	audi	a4	1.8	1999	4	auto(l5)	f	18	29	p	compact
2	audi	a4	1.8	1999	4	manual(m5)	f	21	29	p	compact
3	audi	a4	2.0	2008	4	manual(m6)	f	20	31	p	compact
4	audi	a4	2.0	2008	4	auto(av)	f	21	30	p	compact
5	audi	a4	2.8	1999	6	auto(l5)	f	16	26	p	compact
6	audi	a4	2.8	1999	6	manual(m5)	f	18	26	p	compact
7	audi	a4	3.1	2008	6	auto(av)	f	18	27	p	compact
8	audi	a4 quattro	1.8	1999	4	manual(m5)	4	18	26	p	compact
9	audi	a4 quattro	1.8	1999	4	auto(l5)	4	16	25	p	compact
10	audi	a4 quattro	2.0	2008	4	manual(m6)	4	20	28	p	compact

R Graphics

- Dataset description



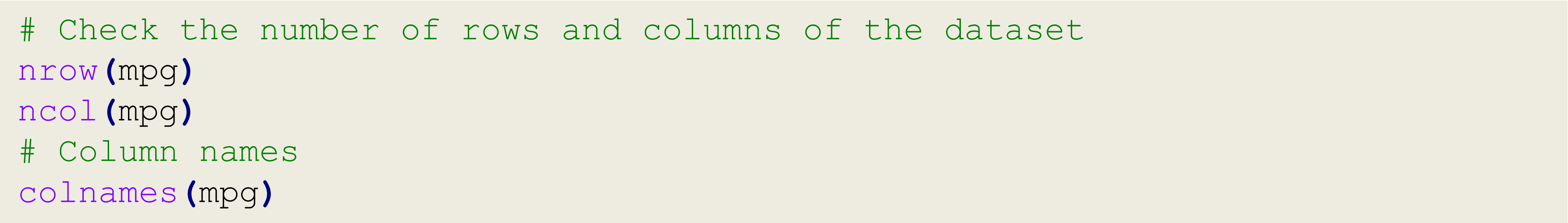
Variable	Type	Description	Details
manufacturer	string	car manufacturer	15 manufacturers
model	string	model name	38 models
displ	numeric	engine displacement in liters	1.6 - 7.0, median: 3.3
year	integer	year of manufacturing	1999, 2008
cyl		number of cylinders	4, 5, 6, 8
trans	string	type of transmission	automatic, manual (many sub types)
drv	string	drive type	f, r, 4, f=front wheel, r=rear wheel, 4=4 wheel
cty	integer	city mileage	miles per gallon
hwy	integer	highway mileage	miles per gallon
fl	string	fuel type	5 fuel types (diesel, petrol, electric, etc.)
class	string	vehicle class	7 types (compact, SUV, minivan etc.)



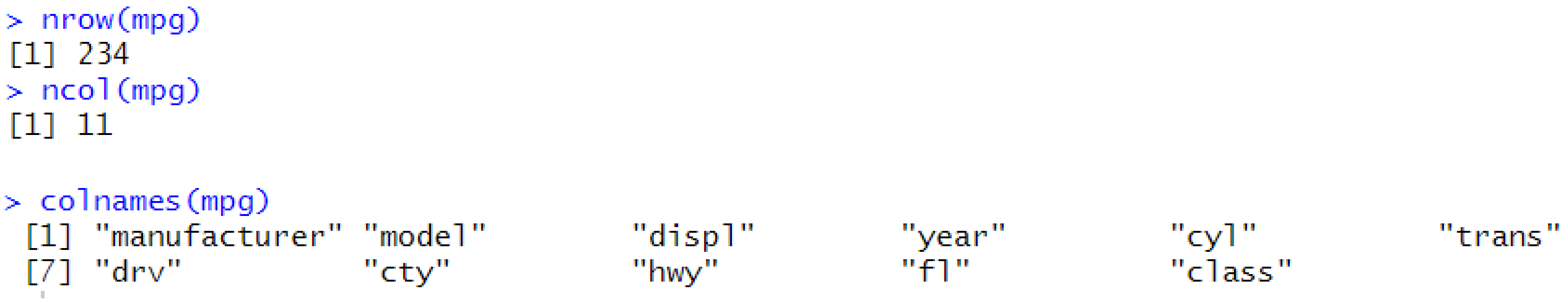
```
> str(mpg)
Classes ‘tbl_df’, ‘tbl’ and 'data.frame':	234 obs. of  11 variables:
 $ manufacturer: chr  "audi" "audi" "audi" "audi" ...
 $ model       : chr  "a4" "a4" "a4" "a4" ...
 $ displ       : num  1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
 $ year        : int  1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...
 $ cyl         : int  4 4 4 4 6 6 6 4 4 4 ...
 $ trans       : chr  "auto(l5)" "manual(m5)" "manual(m6)" "auto(av)" ...
 $ drv         : chr  "f" "f" "f" "f" ...
 $ cty         : int  18 21 20 21 16 18 18 18 16 20 ...
 $ hwy         : int  29 29 31 30 26 26 27 26 25 28 ...
 $ fl          : chr  "p" "p" "p" "p" ...
 $ class       : chr  "compact" "compact" "compact" "compact" ...
```

R Graphics

- Note: All of the following sample codes are from Rpubs website
  - ✓ https://rpubs.com/shailesh/mpg-exploration
- Check the number of rows and columns of the dataset



```
# Check the number of rows and columns of the dataset
nrow(mpg)
ncol(mpg)
# Column names
colnames(mpg)
```



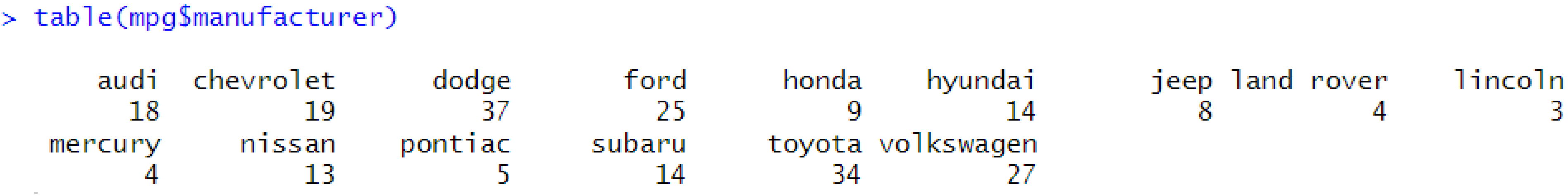
```
> nrow(mpg)
[1] 234
> ncol(mpg)
[1] 11

> colnames(mpg)
 [1] "manufacturer" "model"        "displ"        "year"         "cyl"          "trans"
 [7] "drv"          "cty"          "hwy"          "fl"           "class"
```

R Graphics

- qplot: function for quick and simple plot: Histogram for manufacturer

```
# qplot: function for quick and simple plot: Histogram for manufacturer
table(mpg$manufacturer)
qplot(manufacturer, data=mpg, geom="bar", fill=manufacturer)
```



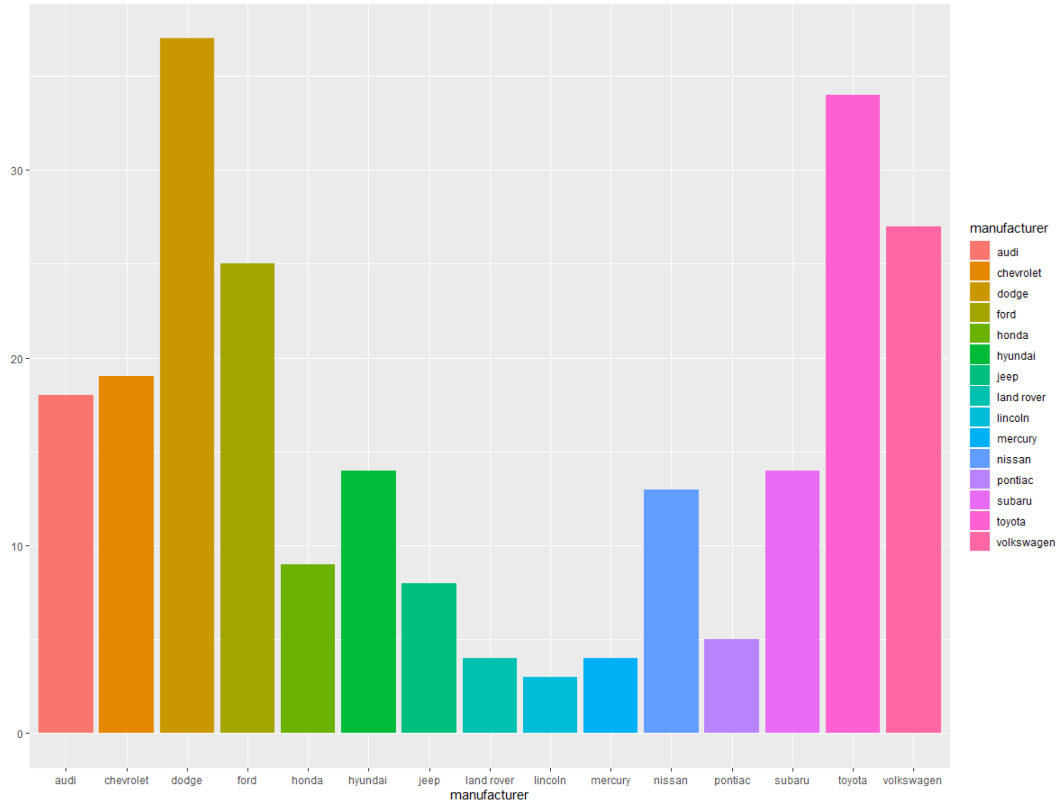
```
> table(mpg$manufacturer)

      audi  chevrolet      dodge       ford      honda    hyundai       jeep land rover    lincoln 
        18         19         37         25          9         14          8          4          3 
   mercury     nissan    pontiac     subaru     toyota volkswagen 
         4         13          5         14         34         27 
```

R Graphics

- qplot: function for quick and simple plot: Bar plot for manufacturer

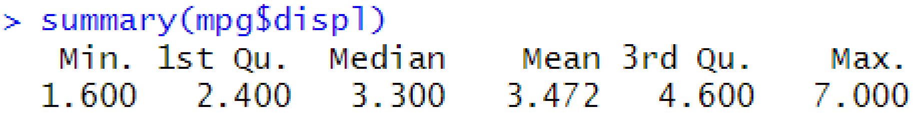
```
# qplot: function for quick and simple plot: Bar plot for manufacturer
table(mpg$manufacturer)
qplot(manufacturer, data=mpg, geom="bar", fill=manufacturer)
```



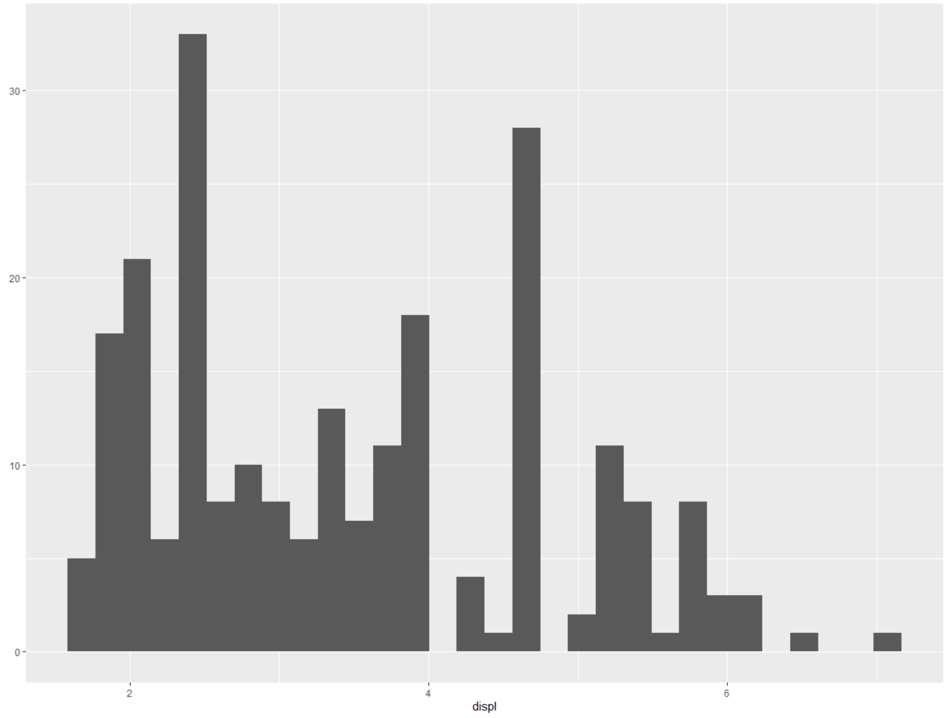
R Graphics

- qplot: function for quick and simple plot: Histogram for manufacturer

```
# qplot: function for quick and simple plot: Histogram for displacement
summary(mpg$displ)
qplot(displ, data=mpg, geom="histogram", bin=20)
```



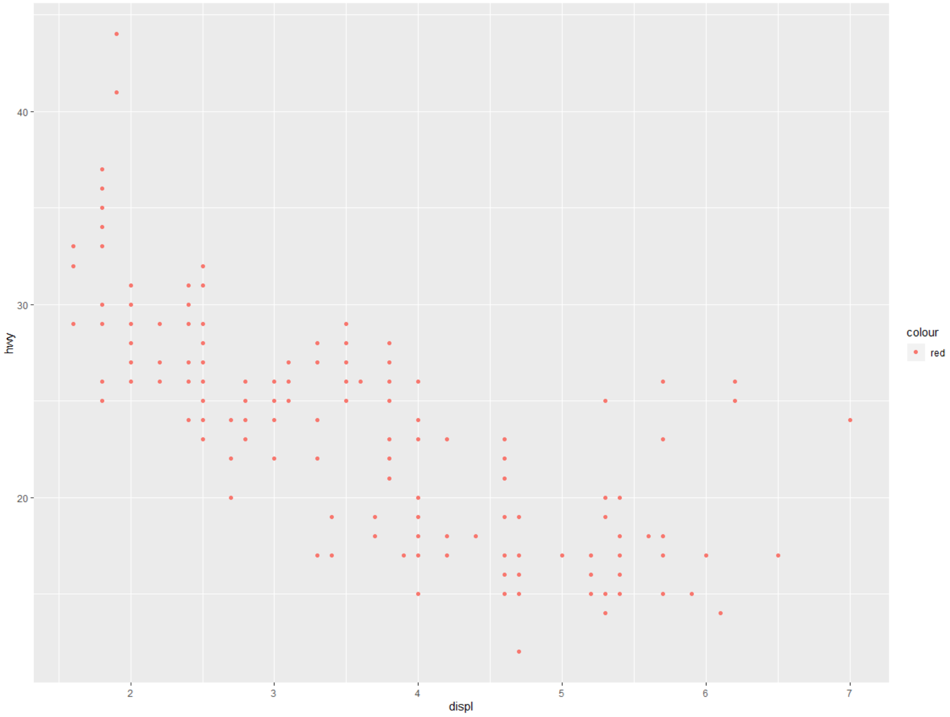
```
> summary(mpg$displ)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  1.600   2.400   3.300   3.472   4.600   7.000 
```



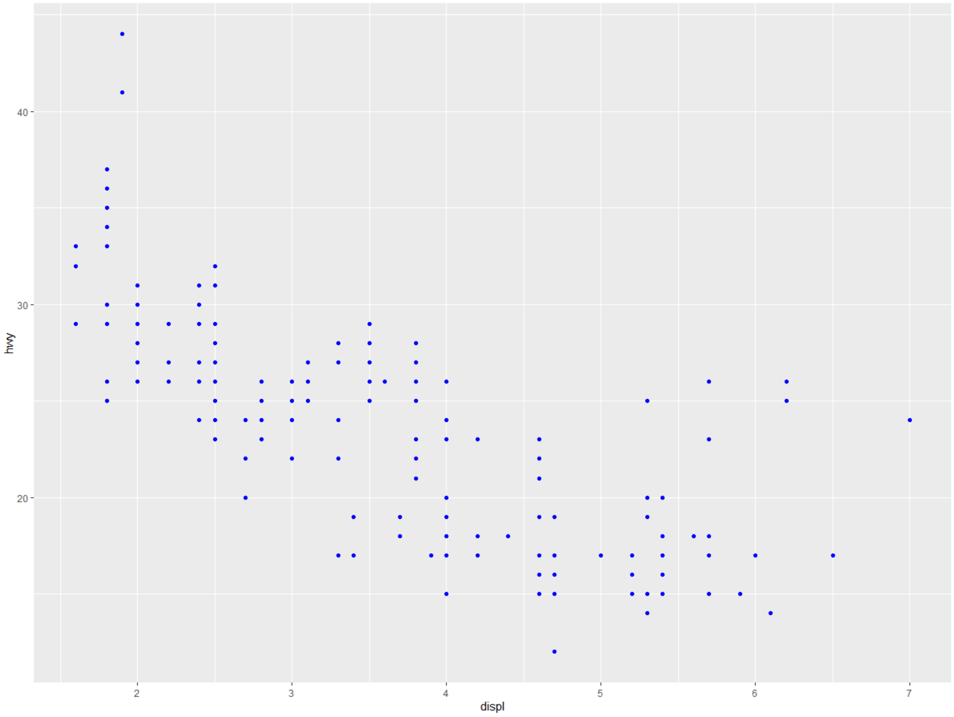
R Graphics

- Same graph with qplot( ) and ggplot( )

```
# Same graph with qplot( ) and ggplot( )
qplot(displ, hwy, data=mpg, geom="point", color='red')
ggplot(mpg, aes(x = displ, y = hwy)) + geom_point(color='blue')
```



qplot( )

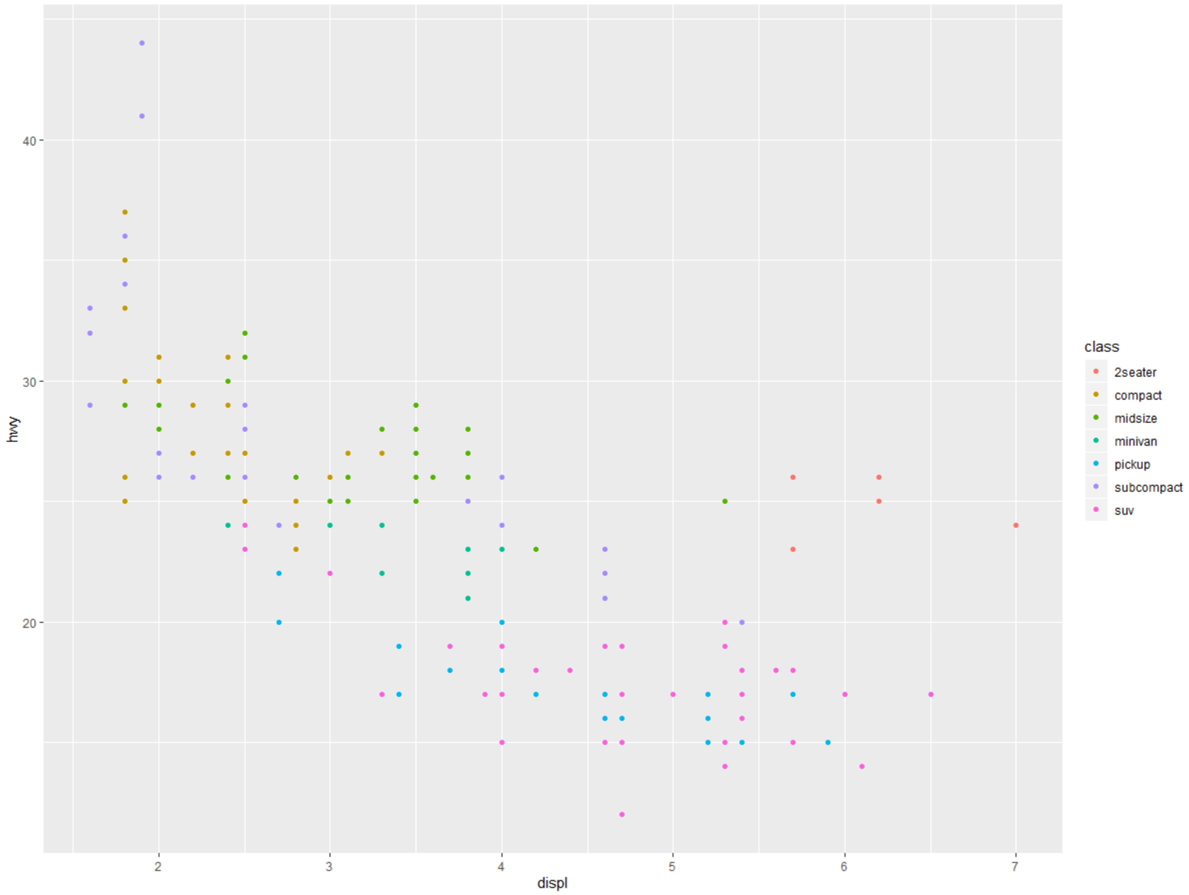


ggplot( )

R Graphics

- Looking at the data separately for each class

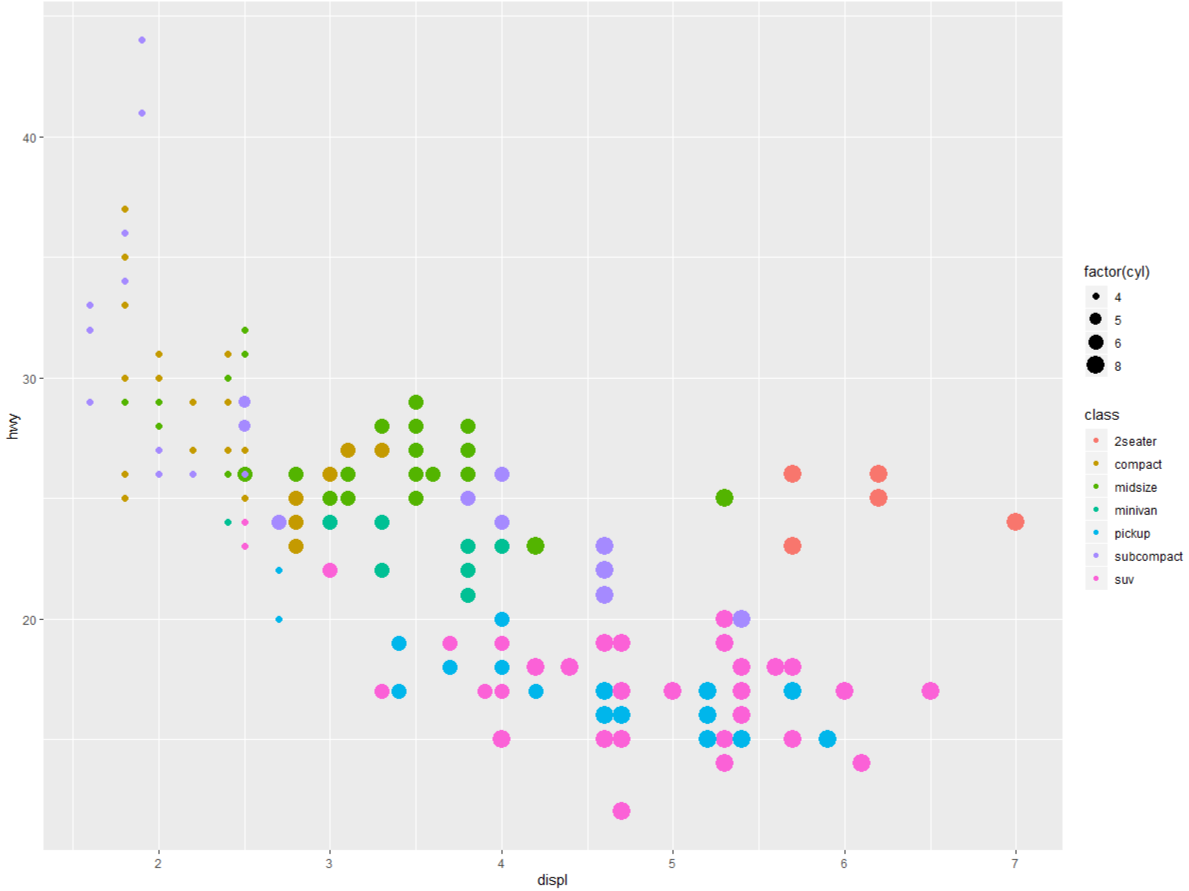
```
# Looking at the data separately for each class
ggplot(mpg, aes(x = displ, y = hwy, color=class)) + geom_point()
```



R Graphics

- Looking at the data separately for each class

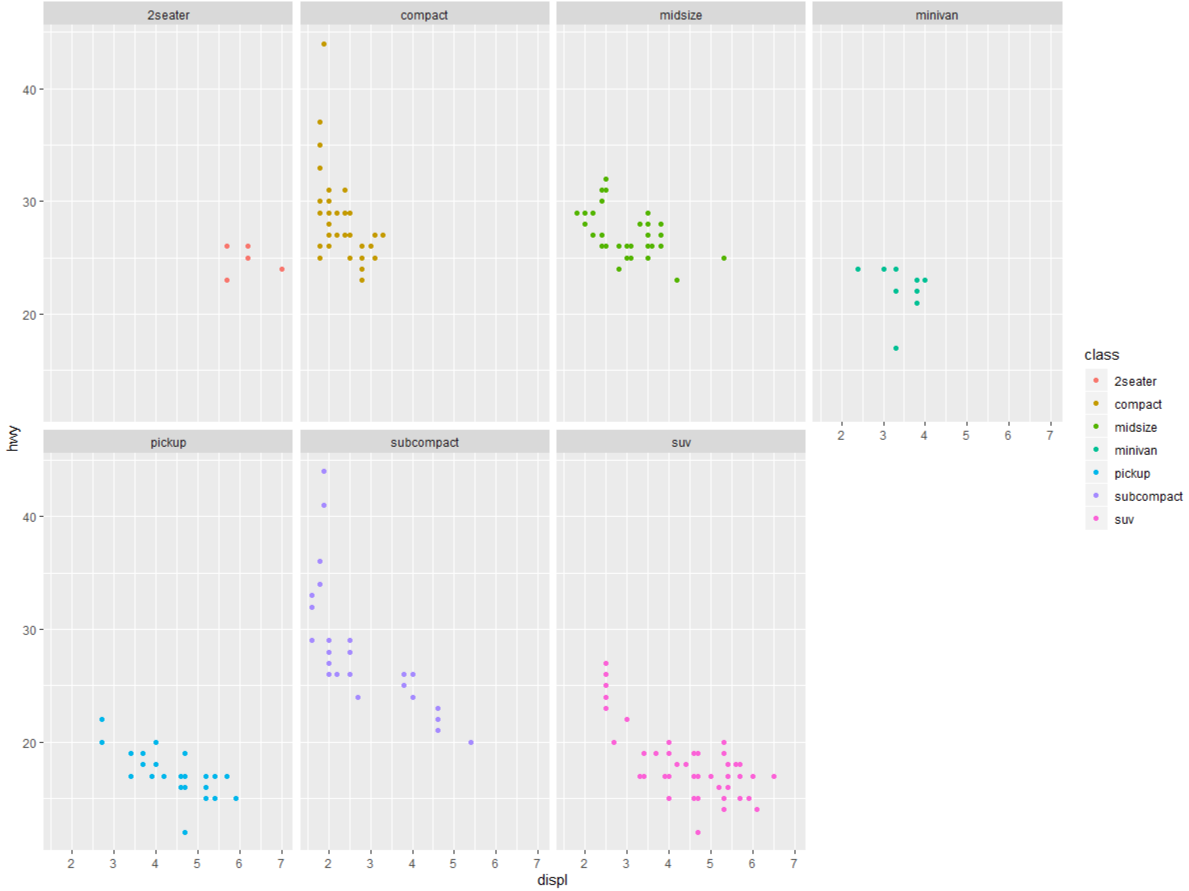
```
# Add another information using the size of points
ggplot(mpg, aes(x = displ, y = hwy, colour = class)) +
         geom_point(aes(size = factor(cyl)))
```



R Graphics

- Separate graphs for each vehicle class

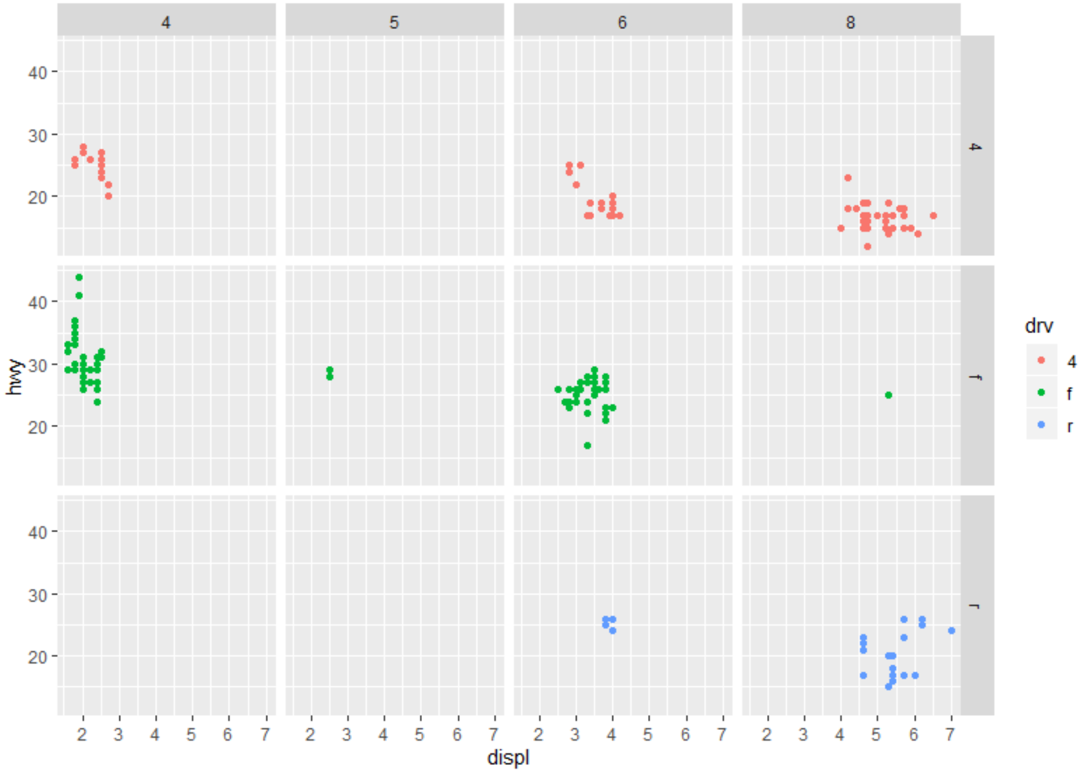
```
# Separate graphs for each vehicle class
ggplot(data = mpg) +
    geom_point(mapping = aes(x = displ, y = hwy, color=class)) +
    facet_wrap(~ class, nrow = 2)
```



R Graphics

- Creating facets on the basis of two variables: number of cylinders and type of drive

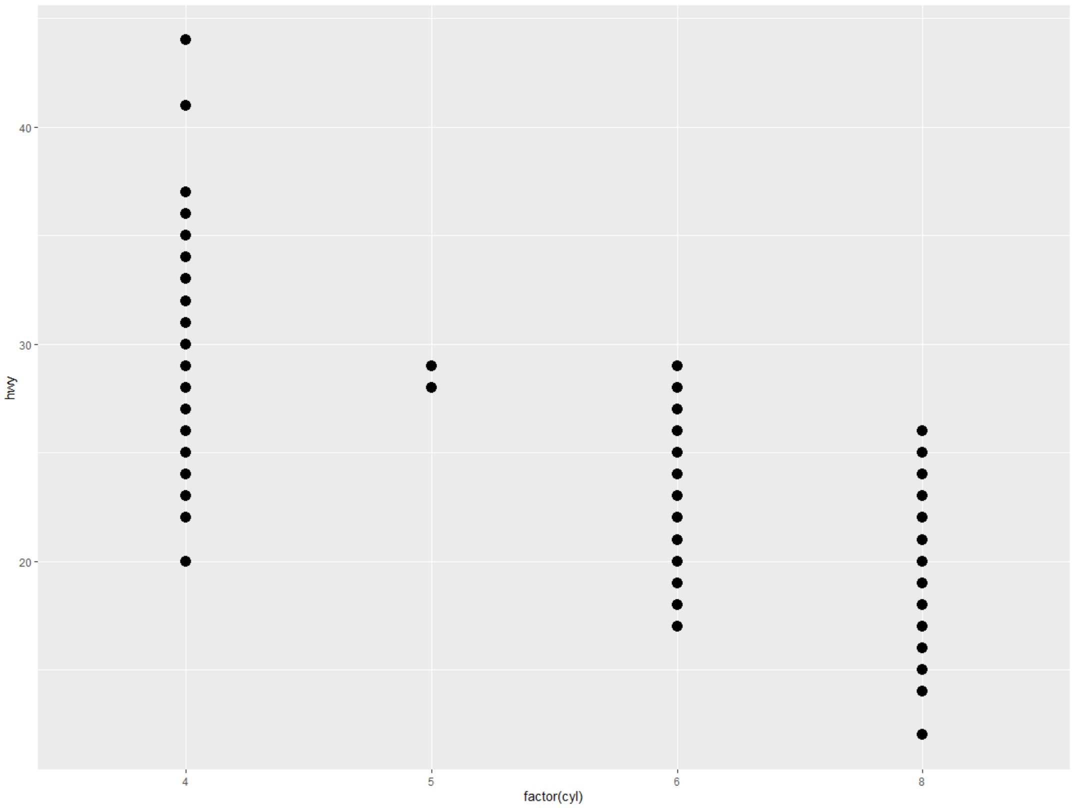
```
# Creating facets on the basis of two variables : number of cylinders and
type of drive
ggplot(data = mpg) +
     geom_point(mapping = aes(x = displ, y = hwy, color=drv)) +
     facet_grid(drv ~ cyl)
```



R Graphics

- Plot for a continuous variable w.r.t. a categorical variable

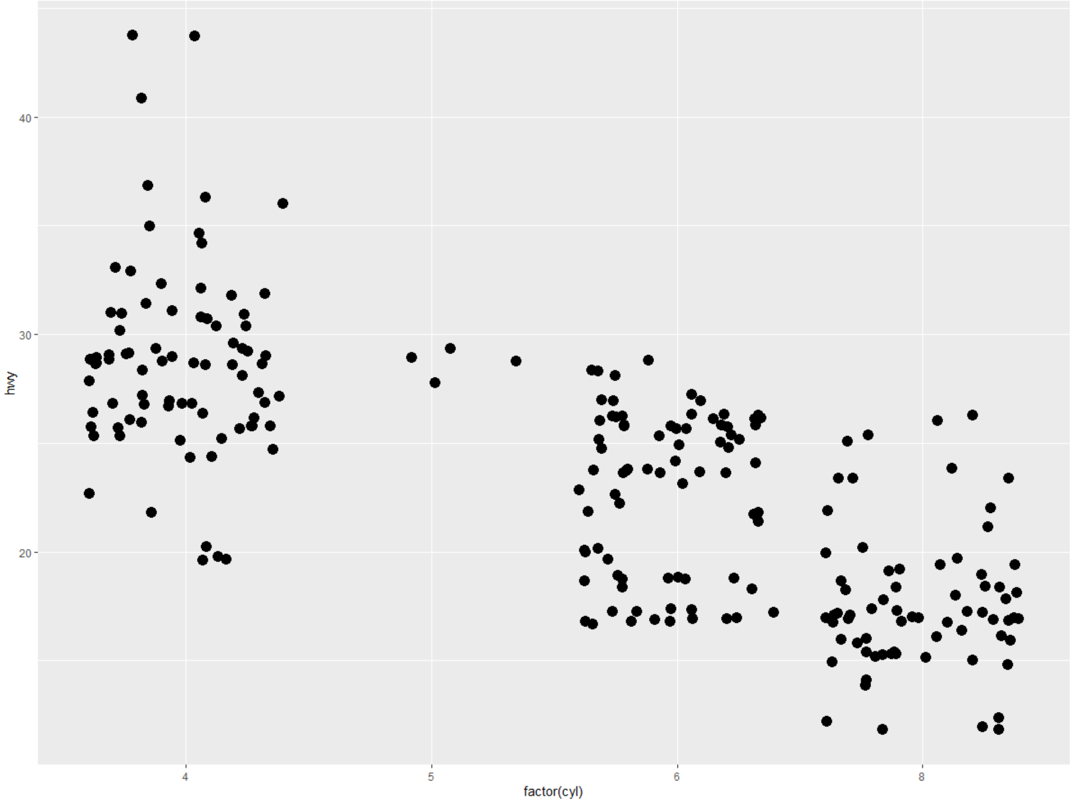
```
# Continuous + categorical
p <- ggplot(mpg, aes(factor(cyl), hwy))
p + geom_point(size=4) # Overlaid dots
p + geom_point(size=4, position="jitter") # Jittered dots
p + geom_point(size=4, position="jitter", alpha=.2) # Transparent dots
```



R Graphics

- Plot for a continuous variable w.r.t. a categorical variable

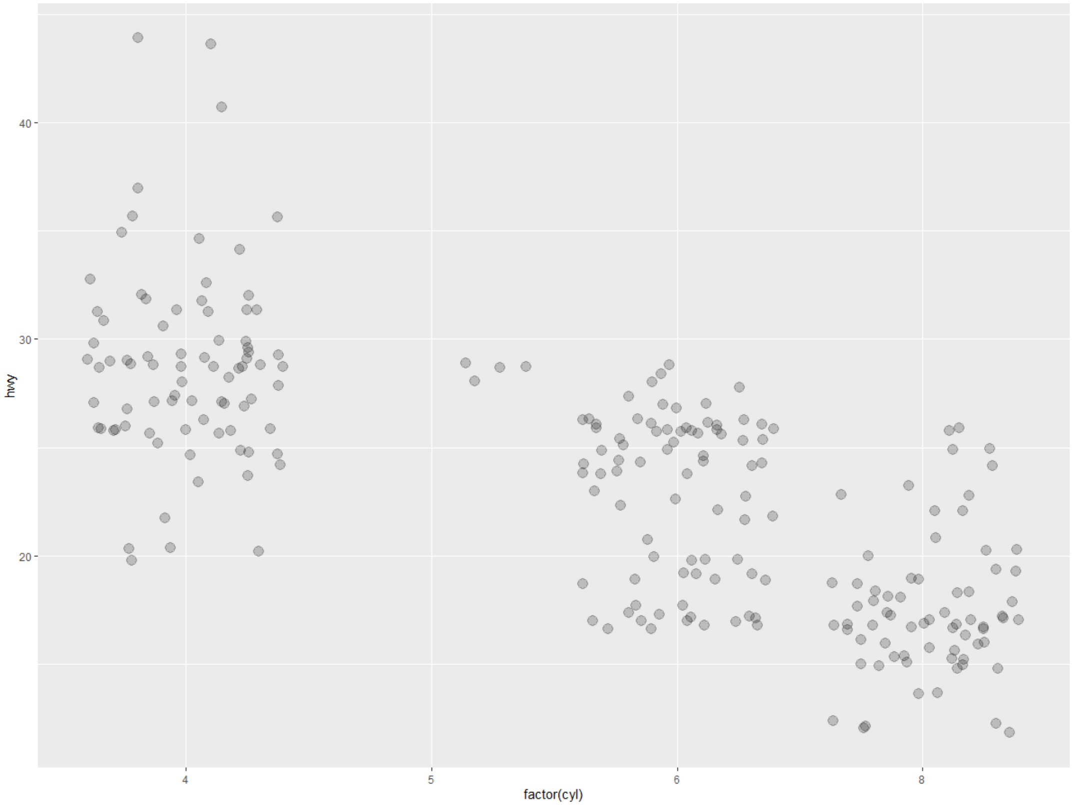
```
# Continuous + categorical
p <- ggplot(mpg, aes(factor(cyl), hwy))
p + geom_point(size=4) # Overlaid dots
p + geom_point(size=4, position="jitter") # Jittered dots
p + geom_point(size=4, position="jitter", alpha=.2) # Transparent dots
```



R Graphics

- Plot for a continuous variable w.r.t. a categorical variable

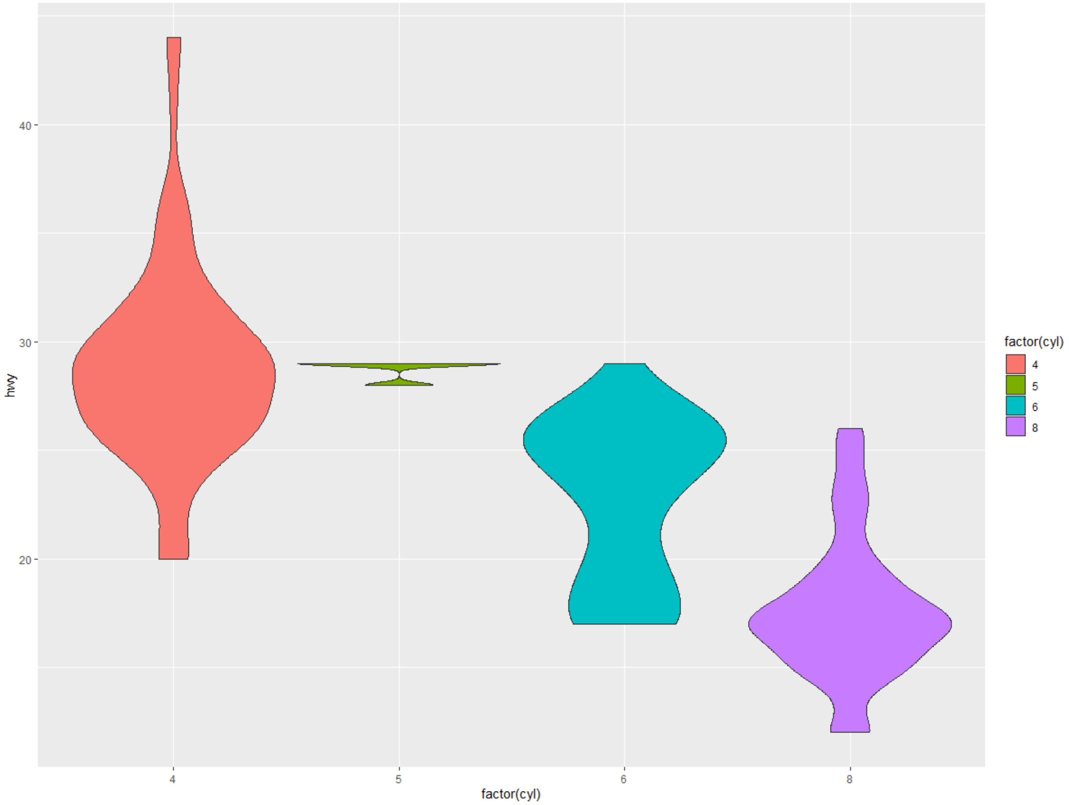
```
# Continuous + categorical
p <- ggplot(mpg, aes(factor(cyl), hwy))
p + geom_point(size=4) # Overlaid dots
p + geom_point(size=4, position="jitter") # Jittered dots
p + geom_point(size=4, position="jitter", alpha=.2) # Transparent dots
```



R Graphics

- Other plots:Violin plot

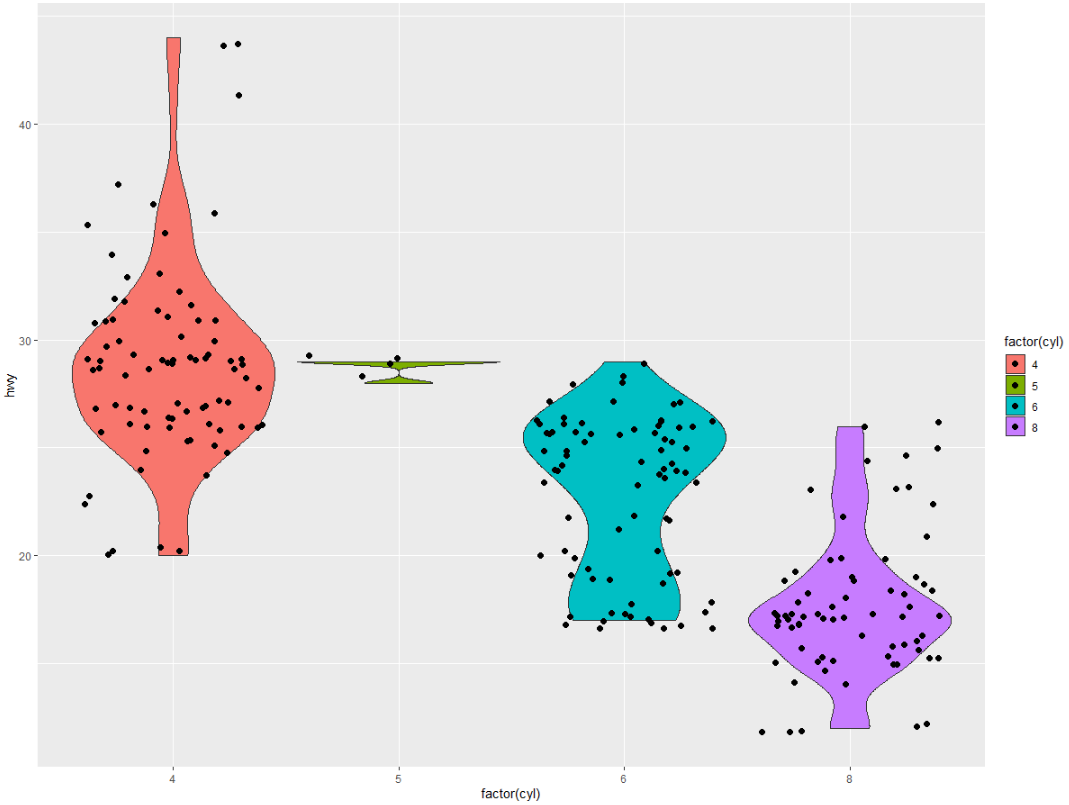
```
# Violin plots
p <- ggplot(mpg, aes(x=factor(cyl), y=hwy, fill=factor(cyl)))
p + geom_violin(scale = "width")
# Add jittered dots for fun
p + geom_violin(scale = "width") + geom_point(size=2, position="jitter")
```



R Graphics

- Other plots:Violin plot

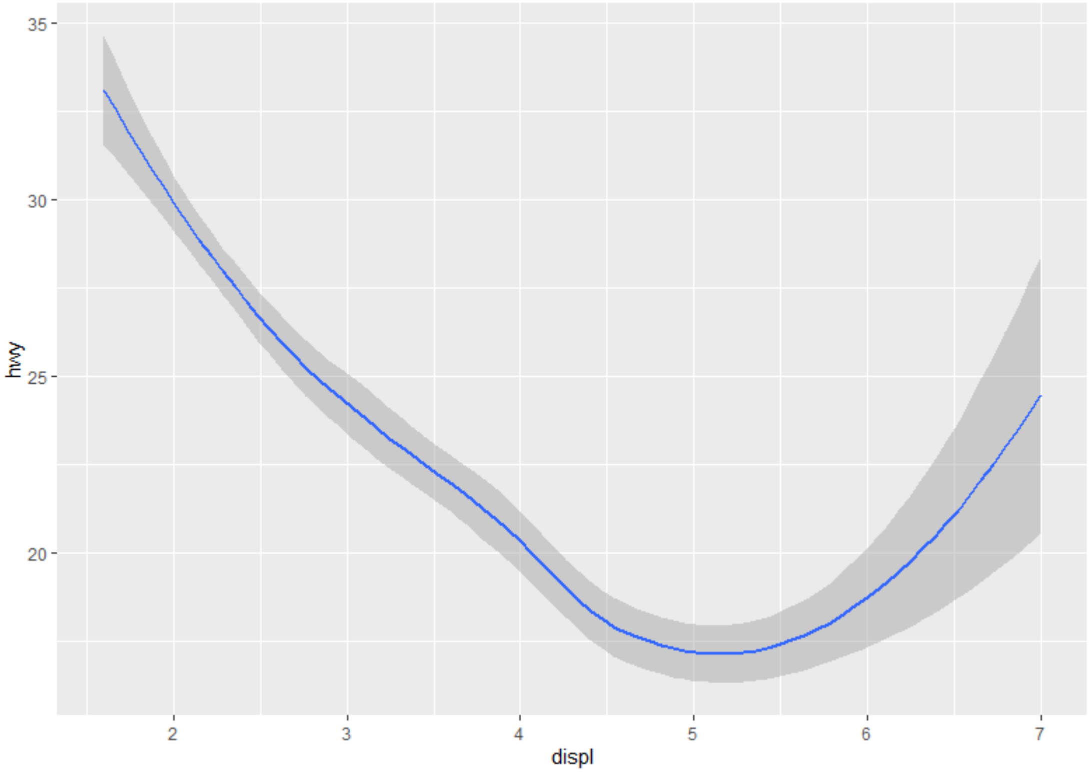
```
# Violin plots
p <- ggplot(mpg, aes(x=factor(cyl), y=hwy, fill=factor(cyl)))
p + geom_violin(scale = "width")
# Add jittered dots for fun
p + geom_violin(scale = "width") + geom_point(size=2, position="jitter")
```



R Graphics

- Estimating a smooth curve for the relationship between displacement and highway mileage:

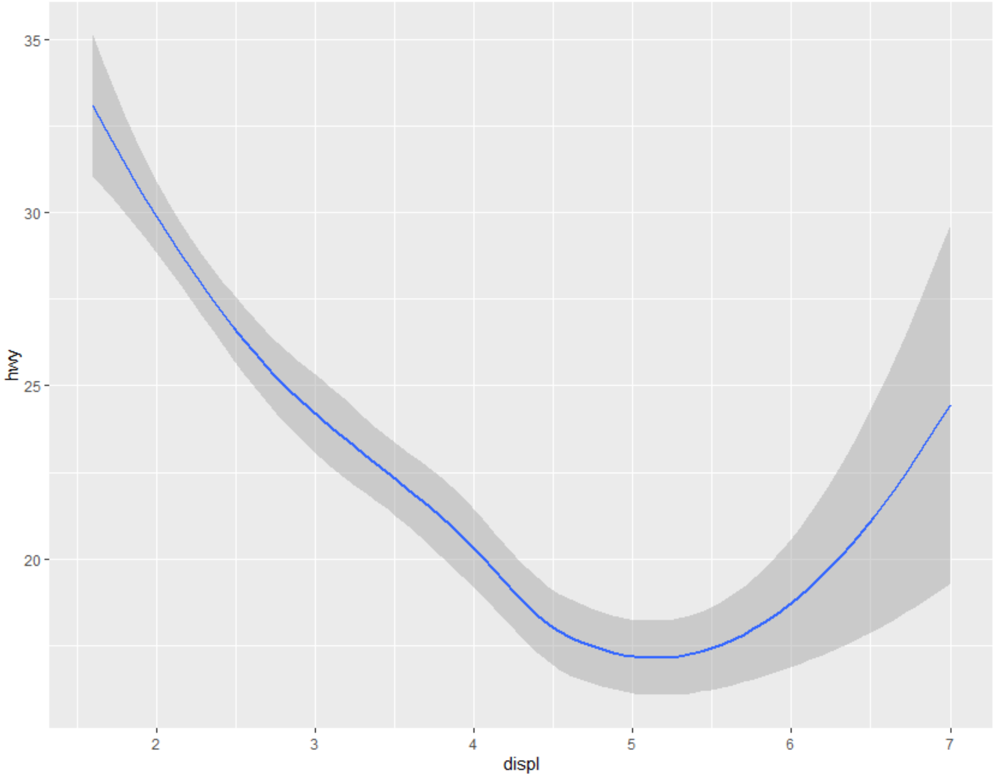
```
# Estimating a smooth curve for the relationship between displacement and
highway mileage:
ggplot(data = mpg) + geom_smooth(mapping = aes(x = displ, y = hwy))
```



R Graphics

- Estimating a smooth curve for the relationship between displacement and highway mileage: change the significance level

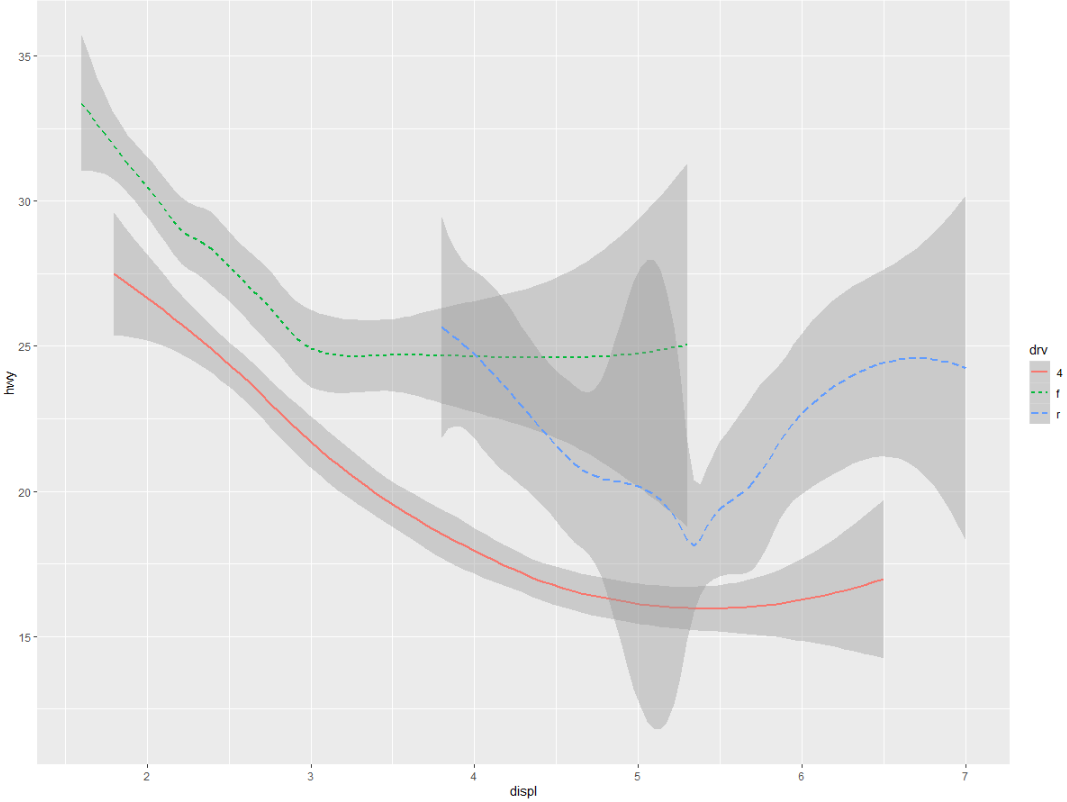
```
# Estimating a smooth curve for the relationship between displacement and
highway mileage:
ggplot(data = mpg) + geom_smooth(mapping = aes(x = displ, y = hwy))
```



R Graphics

- Estimating a smooth curve for the relationship between displacement and highway mileage: change the significance level

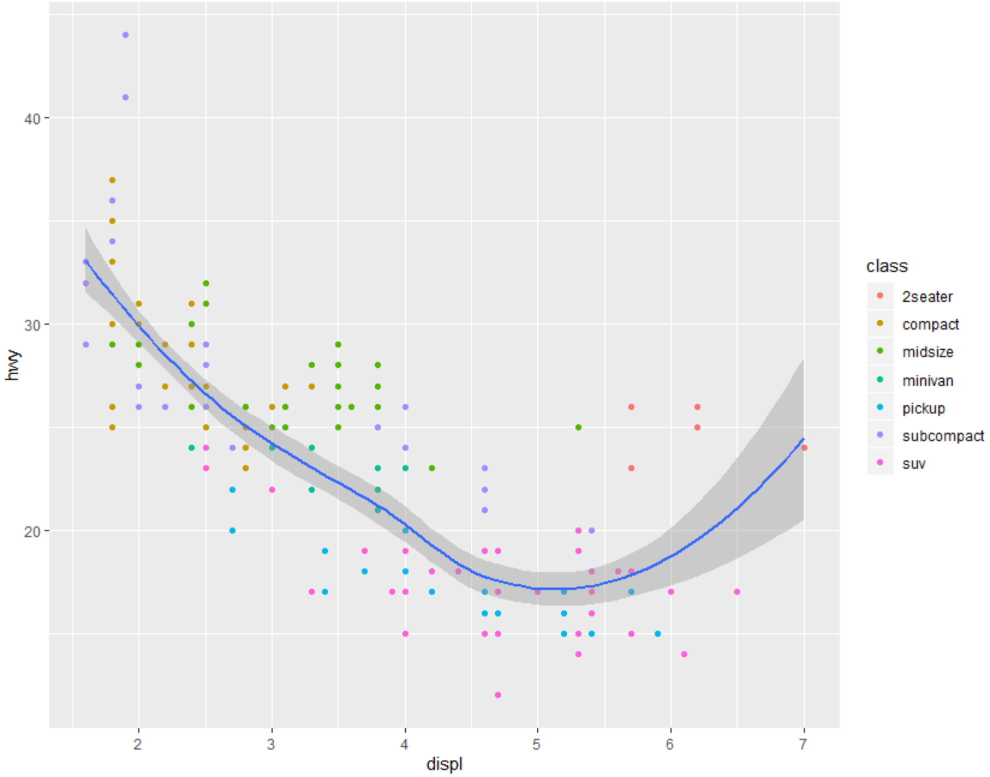
```
# Separate curve for each type of drive:
ggplot(data = mpg) +
geom_smooth(mapping = aes(x = displ, y = hwy, linetype = drv, color=drv))
```



R Graphics

- Overlaying a smooth curve on top of scatter plot:

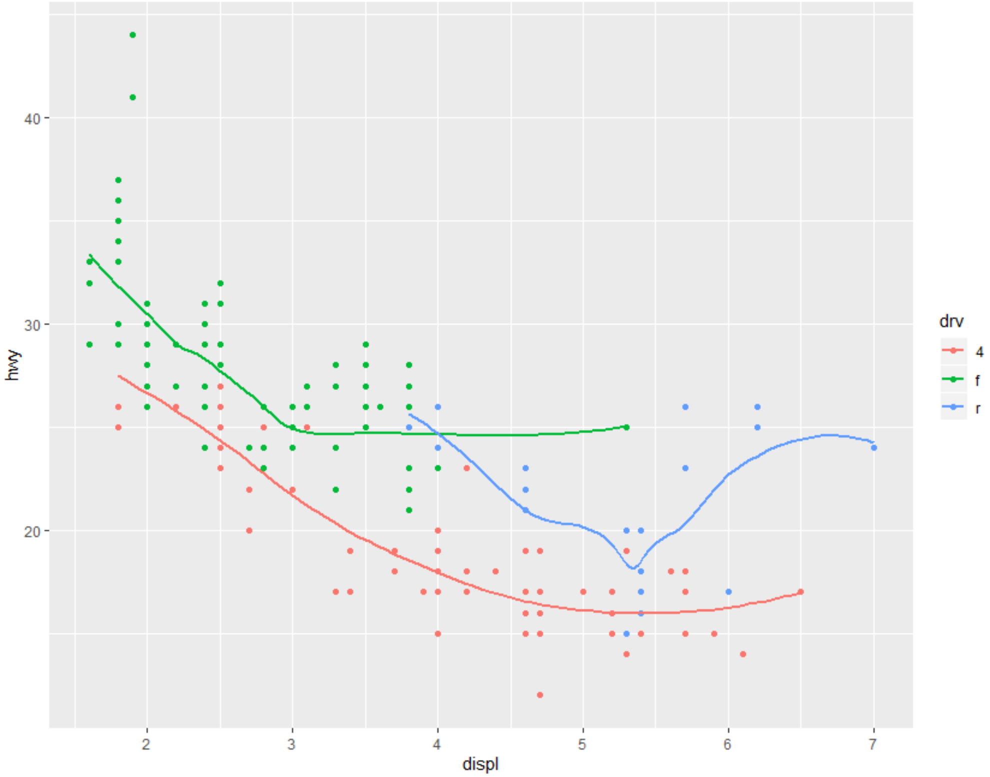
```
# Overlaying a smooth curve on top of scatter plot:
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
         geom_point(mapping=aes(color=class)) +
         geom_smooth()
```



R Graphics

- Grouping data by drive and then drawing scatter plot with estimated curve for each group:

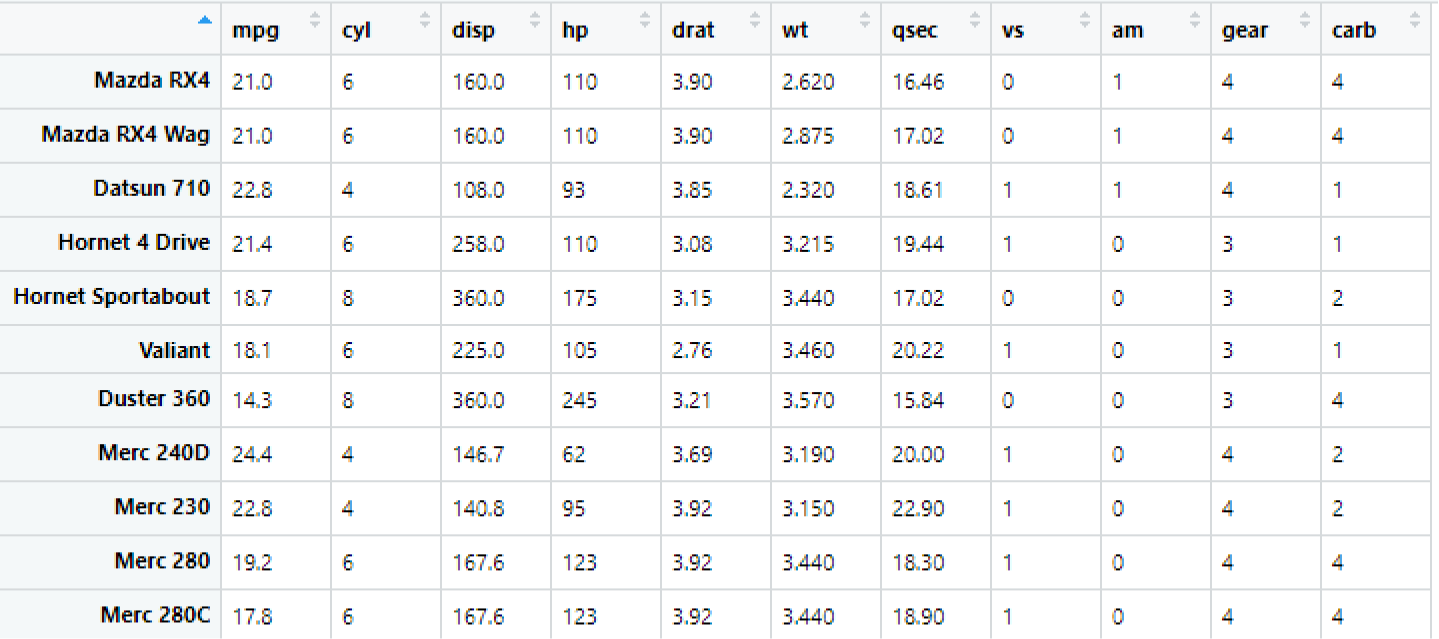
```r
# Grouping data by drive and then drawing scatter plot with estimated curve for each group:
ggplot(data = mpg, mapping = aes(x = displ, y = hwy, color = drv)) +
geom_point() +
geom_smooth(se = FALSE)
```



R Graphics

- Load another dataset: mtcars

```r
# Load another dataset
data("mtcars")
head(mtcars)
```

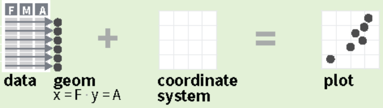


	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4

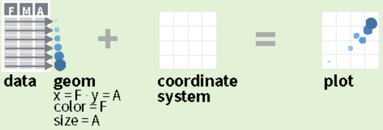
Data Visualization with ggplot2 : : CHEAT SHEET

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>) +                                      required
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),
 stat = <STAT>, position = <POSITION>) +                      Not
<COORDINATE_FUNCTION> +                                       required,
<FACET_FUNCTION> +                                            sensible
<SCALE_FUNCTION> +                                            defaults
<THEME_FUNCTION>                                              supplied
```

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

aesthetic mappings · data · geom

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(date, 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(lineend="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)) - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900)) - x, ymax, ymin, 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 = 1:1155, 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(fl))

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

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, colour, group, linetype, size

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

continuous function

i <- ggplot(economics, aes(date, 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

THREE VARIABLES

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

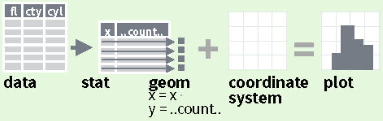
l + geom_contour(aes(z = z))
x, y, z, alpha, colour, group, linetype, size, weight

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

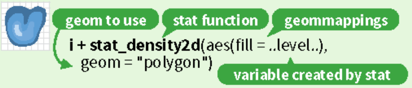
l + geom_tile(aes(fill = z)), x, y, alpha, color, fill, linetype, size, width

Stats An alternative way to build a layer

A stat builds new variables to plot (e.g., count, prop).



Visualize a stat by changing the default stat of a geom function, **geom_bar(stat="count")** or by using a stat function, **stat_count(geom="bar")**, which calls a default geom to make a layer (equivalent to a geom function). Use **..name..** syntax to map stat variables to aesthetics.



i + stat_density2d(aes(fill = ..level..),
geom = "polygon")

c + stat_bin(binwidth = 1, origin = 10)
x, y | ..count.., ..ncount.., ..density.., ..ndensity..

c + stat_count(width = 1) **x, y,** | ..count.., ..prop..

c + stat_density(adjust = 1, kernel = "gaussian")
x, y, | ..count.., ..density.., ..scaled..

e + stat_bin_2d(bins = 30, drop = T)
x, y, fill | ..count.., ..density..

e + stat_bin_hex(bins=30) **x, y, fill** | ..count.., ..density..

e + stat_density_2d(contour = TRUE, n = 100)
x, y, color, size | ..level..

e + stat_ellipse(level = 0.95, segments = 51, type = "t")

l + stat_contour(aes(z = z)) **x, y, z, order** | ..level..

l + stat_summary_hex(aes(z = z), bins = 30, fun = max)
x, y, z, fill | ..value..

l + stat_summary_2d(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..

f + stat_boxplot(coef = 1.5) **x, y** | ..lower.., ..middle.., ..upper.., ..width.. , ..ymin.., ..ymax..

f + stat_ydensity(kernel = "gaussian", scale = "area") **x, y** | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

e + stat_ecdf(n = 40) **x, y** | ..x.., ..y..

e + stat_quantile(quantiles = c(0.1, 0.9), formula = y ~ log(x), method = "rq") **x, y** | ..quantile..

e + stat_smooth(method = "lm", formula = y ~ x, se=T, level=0.95) **x, y** | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

ggplot() + stat_function(aes(x = -3:3), n = 99, fun = dnorm, args = list(sd=0.5)) **x** | ..x.., ..y..

e + stat_identity(na.rm = TRUE)

ggplot() + stat_qq(aes(sample=1:100), dist = qt, dparam=list(df=5)) **sample, x, y** | ..sample.., ..theoretical..

e + stat_sum() **x, y, size** | ..n.., ..prop..

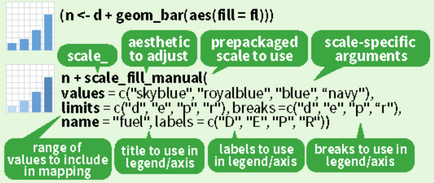
e + stat_summary(fun.data = "mean_cl_boot")

h + stat_summary_bin(fun.y = "mean", geom = "bar")

e + stat_unique()

Scales

Scales map data values to the visual values of an aesthetic. To change a mapping, add a new scale.


scale_
aesthetic to adjust
prepackaged scale to use
scale-specific arguments
range of values to include in mapping
title to use in legend/axis
labels to use in legend/axis
breaks to use in legend/axis

(n <- d + geom_bar(aes(fill = fl)))

n + scale_fill_manual(
values = c("skyblue", "royalblue", "blue", "navy"),
limits = c("d", "e", "p", "r"), breaks =c("d", "e", "p", "r"),
name = "fuel", labels = c("D", "E", "P", "R"))

GENERAL PURPOSE SCALES

Use with most aesthetics

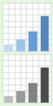
scale_*_continuous() - map cont' values to visual ones
scale_*_discrete() - map discrete values to visual ones
scale_*_identity() - use data values **as** visual ones
scale_*_manual(values = c()) - map discrete values to manually chosen visual ones
scale_*_date(date_labels = "%m/%d"), date_breaks = "2 weeks") - treat data values as dates.
scale_*_datetime() - treat data x values as date times. Use same arguments as scale_x_date(). See ?strptime for label formats.

X & Y LOCATION SCALES

Use with x or y aesthetics (x shown here)

scale_x_log10() - Plot x on log10 scale
scale_x_reverse() - Reverse direction of x axis
scale_x_sqrt() - Plot x on square root scale

COLOR AND FILL SCALES (DISCRETE)



n <- d + geom_bar(aes(fill = fl))

n + scale_fill_brewer(palette = "Blues")
For palette choices:
RColorBrewer::display.brewer.all()

n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red")

COLOR AND FILL SCALES (CONTINUOUS)

o <- c + geom_dotplot(aes(fill = ..x..))

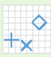
o + scale_fill_distiller(palette = "Blues")

o + scale_fill_gradient(low="red", high="yellow")

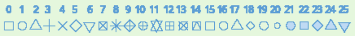
o + scale_fill_gradient2(low="red", high="blue", mid = "white", midpoint = 25)

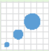
o + scale_fill_gradientn(colours=topo.colors(6))
Also: rainbow(), heat.colors(), terrain.colors(), cm.colors(), RColorBrewer::brewer.pal()

SHAPE AND SIZE SCALES



p <- e + geom_point(aes(shape = fl, size = cyl))
p + scale_shape() **+ scale_size**()
p + scale_shape_manual(values = c(3:7))

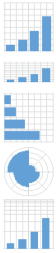




p + scale_radius(range = c(1,6))
p + scale_size_area(max_size = 6)

Coordinate Systems

r <- d + geom_bar()



r + coord_cartesian(xlim = c(0, 5))
xlim, ylim
The default cartesian coordinate system

r + coord_fixed(ratio = 1/2)
ratio, xlim, ylim
Cartesian coordinates with fixed aspect ratio between x and y units

r + coord_flip()
xlim, ylim
Flipped Cartesian coordinates

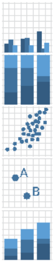
r + coord_polar(theta = "x", direction=1)
theta, start, direction
Polar coordinates

r + coord_trans(ytrans = "sqrt")
xtrans, ytrans, limx, limy
Transformed cartesian coordinates. Set xtrans and ytrans to the name of a window function.

π + coord_quickmap()
π + coord_map(projection = "ortho", orientation=c(41, -74, 0))projection, orienztation, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)

Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.



s <- ggplot(mpg, aes(fl, fill = drv))

s + geom_bar(position = "dodge")
Arrange elements side by side

s + geom_bar(position = "fill")
Stack elements on top of one another, normalize height

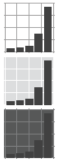
e + geom_point(position = "jitter")
Add random noise to X and Y position of each element to avoid overplotting

e + geom_label(position = "nudge")
Nudge labels away from points

s + geom_bar(position = "stack")
Stack elements on top of one another

Each position adjustment can be recast as a function with manual **width** and **height** arguments
s + geom_bar(position = position_dodge(width = 1))

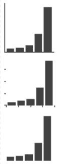
Themes



r + theme_bw()
White background with grid lines

r + theme_gray()
Grey background (default theme)

r + theme_dark()
dark for contrast



r + theme_classic()

r + theme_light()

r + theme_linedraw()

r + theme_minimal()
Minimal themes

r + theme_void()
Empty theme

Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

t <- ggplot(mpg, aes(cty, hwy)) + geom_point()

t + facet_grid(cols = vars(fl))
facet into columns based on fl

t + facet_grid(rows = vars(year))
facet into rows based on year

t + facet_grid(rows = vars(year), cols = vars(fl))
facet into both rows and columns

t + facet_wrap(vars(fl))
wrap facets into a rectangular layout

Set **scales** to let axis limits vary across facets

t + facet_grid(rows = vars(drv), cols = vars(fl),
scales = "free")
x and y axis limits adjust to individual facets
"free_x" - x axis limits adjust
"free_y" - y axis limits adjust

Set **labeller** to adjust facet labels

t + facet_grid(cols = vars(fl), labeller = label_both)

fl: c	fl: d	fl: e	fl: p	fl: r

t + facet_grid(rows = vars(fl),
labeller = label_bquote(alpha ^ .(fl)))

α^c	α^d	α^e	α^p	α^r

Labels

t + labs(**x** = "New x axis label", **y** = "New y axis label",
title ="Add a title above the plot",
subtitle = "Add a subtitle below title",
caption = "Add a caption below plot",
<AES> = "New <AES> legend title")

Use scale functions to update legend labels

t + annotate(geom = "text", x = 8, y = 9, label = "A")

geom to place

manual values for geom's aesthetics

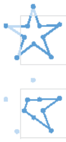
Legends

n + theme(legend.position = "bottom")
Place legend at "bottom", "top", "left", or "right"

n + guides(fill = "none")
Set legend type for each aesthetic: colorbar, legend, or none (no legend)

n + scale_fill_discrete(name = "Title", labels = c("A", "B", "C", "D", "E"))
Set legend title and labels with a scale function.

Zooming



Without clipping (preferred)

t + coord_cartesian(
xlim = c(0, 100), ylim = c(10, 20))

With clipping (removes unseen data points)

t + xlim(0, 100) **+ ylim**(10, 20)

t + scale_x_continuous(limits = c(0, 100)) +
scale_y_continuous(limits = c(0, 100))

ANY
questions?