CT5102: Programming for Data Analytics

5. Visualisation with ggplot2

Prof. Jim Duggan,
School of Computer Science
University of Galway.

https://github.com/JimDuggan/explore_or

> Data graphics provide one of the most accessible, compelling, and expressive modes to investigate and depict patterns in data.
>
> — Benjamin S. Baumer, Daniel T. Kaplan, and Nicholas J. Horton (Baumer et al., 2021)

Overview

- A core part of any data analysis and modelling process is to visualize data and explore relationships between variables
- Within R's tidyverse we are fortunate to have access to a visualization library known as ggplot2.
- There are three important benefits of using this library:
  1. plots can be designed in a layered manner, where additional plotting details can be added using the + operator;
  2. a wide range of plots can be generated to support decision analysis, including scatterplots, histograms, and time series charts,
  3. once the analyst is familiar with the structure and syntax of ggplot2, charts can be developed rapidly, and this supports an iterative process of decision support.

(5.1) Datasets and tidy data

- For this lecture, we will base our plots on two (tidy) datasets that are already part of the ggplot2 package. These are the tibbles mpg and diamonds.
- With tidy data, where every column is a variable, and every row is an observation.
- These are defined as (Wickham, 2016):
  - A variable is a quantity, quality, or property that you can measure, and will have a value at the time it is measured.
  - An observation is a set of measurements made under similar conditions (often at the same time)

ggplot2::mpg, N = 234

Variable	Description
manufacturer	Manufacturer name
model	Model name
displ	Engine displacement (liters)
year	Year of manufacture
cyl	Number of cylinders
trans	Type of transmission
drv	Type of drive train (e.g. front wheel)
cty	City miles per gallon
hwy	Highway miles per gallon
fl	Fuel type
class	"type" of car (e.g. "compact")

ggplot2::diamonds, N = 53,940

Variable	Description
carat	Weight of the diamond
cut	Quality of the cut (categorical)
color	Diamond color (categorical)
clarity	Diamond clarity (categorical)
depth	Total depth percentage
table	Measure related to width of diamond top
price	Price in dollars
x	Length in mm
y	Width in mm
z	Depth in mm

(5.2) Exploring relationships with a scatterplot

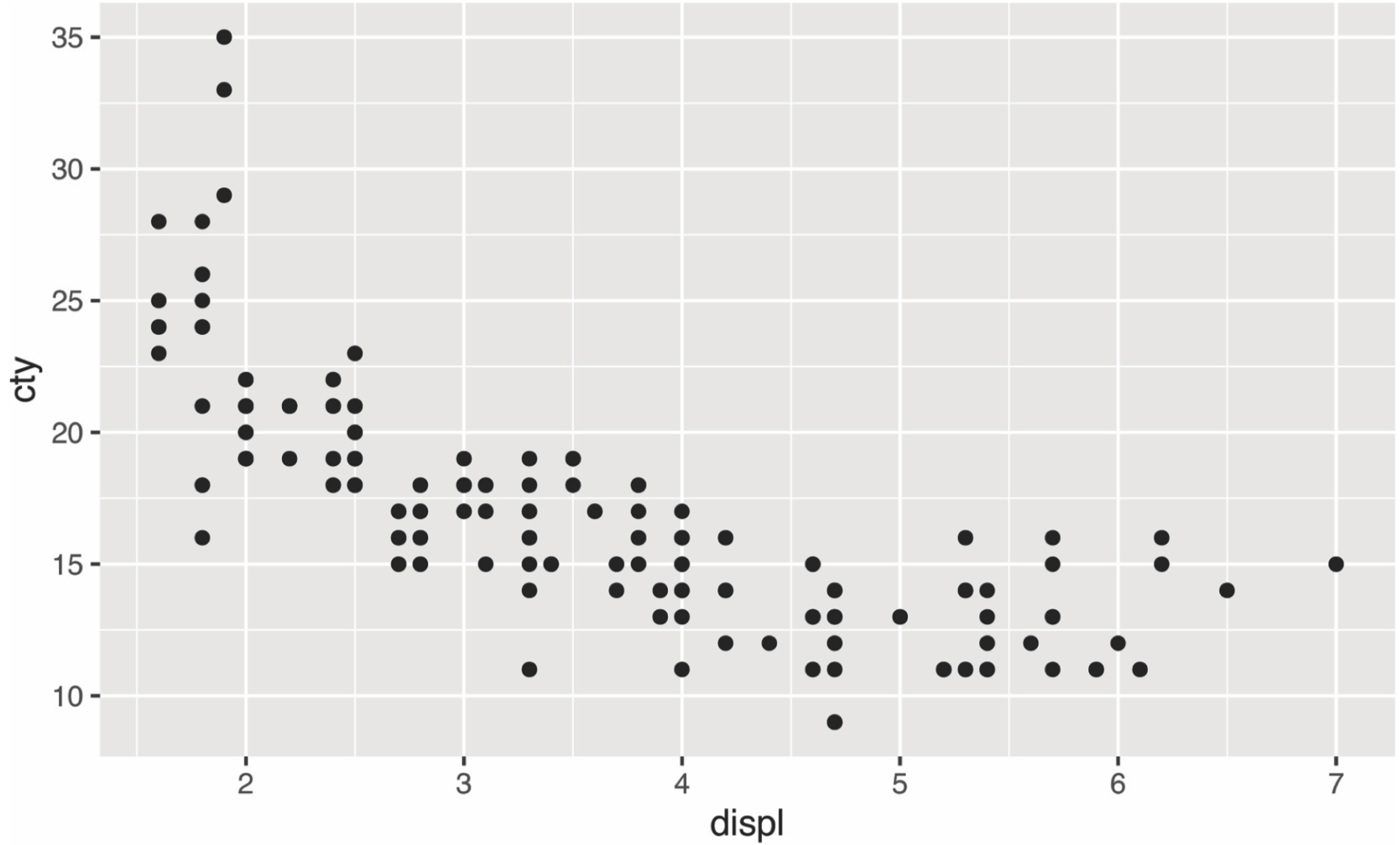
- Creating a scatterplot to explore the relationship between two numeric variables is a convenient way to start our exploration of ggplot2.
- A scatterplot requires two sets of paired values: those that are represented on the x-axis, and the corresponding values mapped to the y-axis.

```
library(ggplot2)
library(tibble)
mpg |> subset(select=c("displ","cty")) |> summary()
#>      displ            cty
#>  Min.   :1.60   Min.   : 9.0
#>  1st Qu.:2.40   1st Qu.:14.0
#>  Median :3.30   Median :17.0
#>  Mean   :3.47   Mean   :16.9
#>  3rd Qu.:4.60   3rd Qu.:19.0
#>  Max.   :7.00   Max.   :35.0
```

Creating our first graph.

1. We call ggplot(data=mpg) which initializes a ggplot object, and this call also allows us to specify the tibble that contains that data.
2. We extend this call to include the x-axis and y-axis variables by including an addition argument (mapping) and the function aes() which 7.4 Aesthetic mappings describe how variables in data are mapped to the plot's visual properties.
3. To visualize the set of points on the graph, and we do this by calling the relevant geometric object, which is one that is designed to draw points, namely the function geom_point()

```
ggplot(data=mpg, mapping=aes(x=displ,y=cty)) +
  geom_point()
```

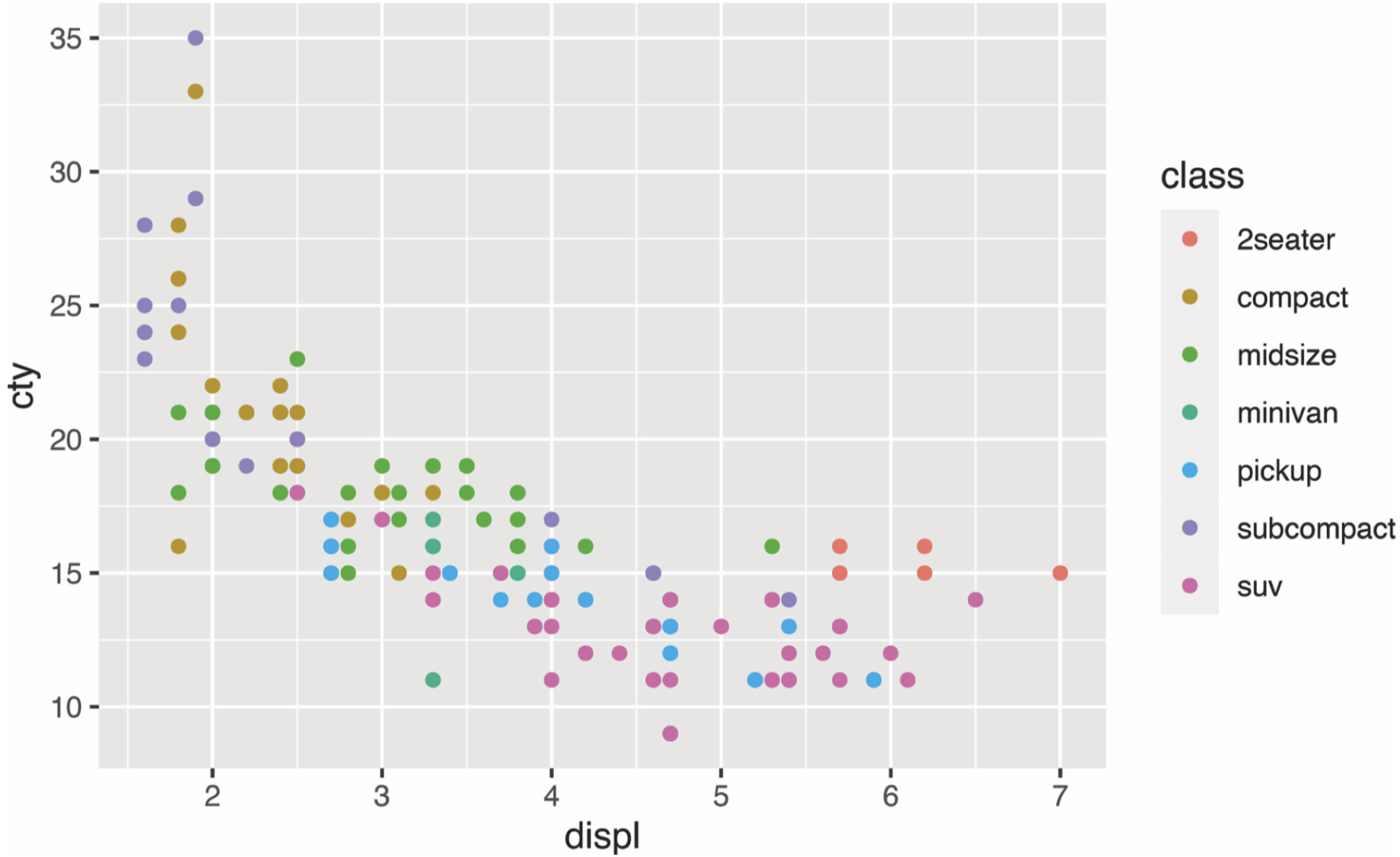


(5.3) Aesthetic Mappings

- The aes() function has some nice additional features that can be used to add extra information to a plot by using data from other variables.
- For example, what if we also wanted to see which class of car each point belonged to?
- We can set the argument color to this class in the aes() function

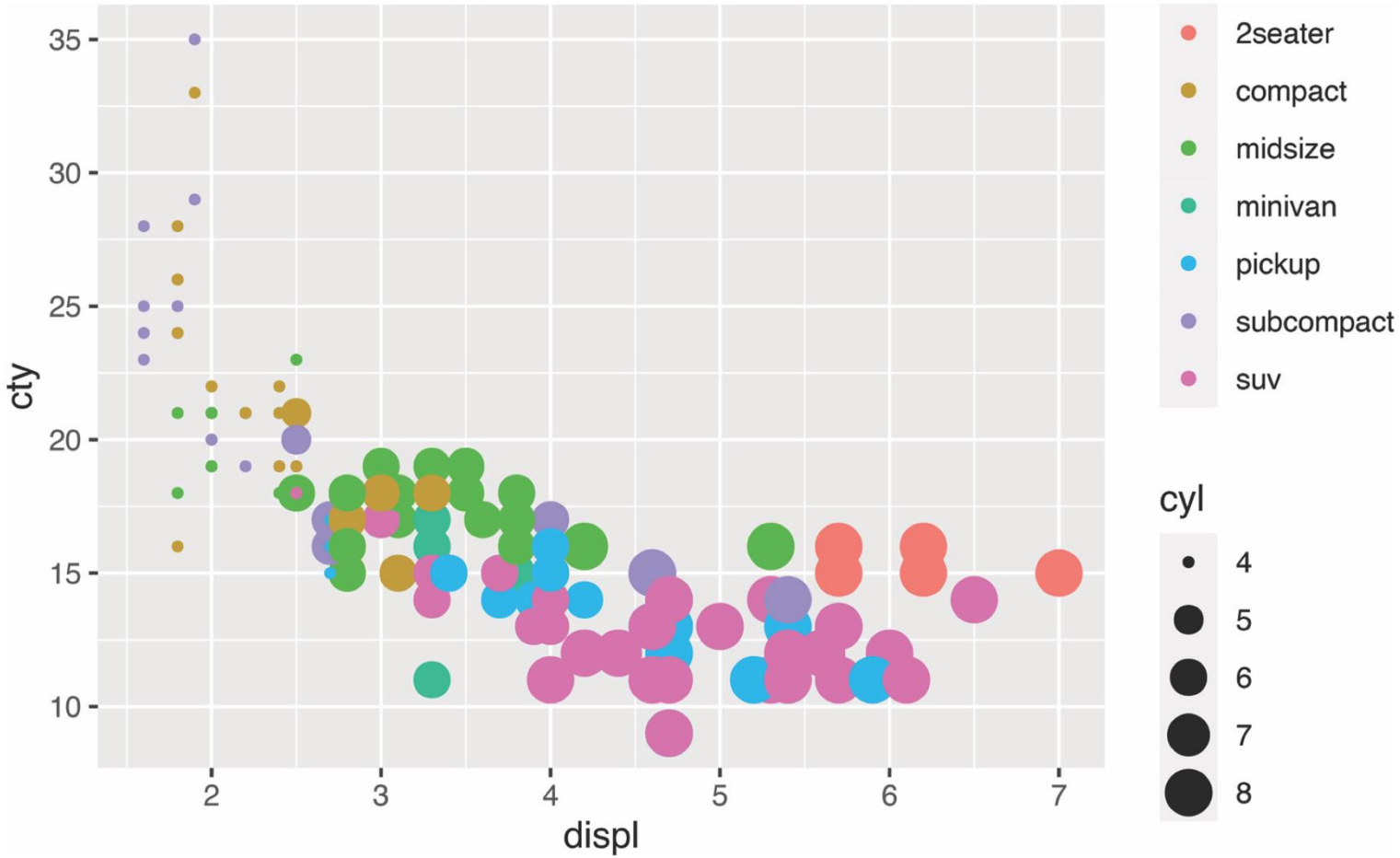
```
unique(mpg$class)
#> [1] "compact"    "midsize"    "suv"        "2seater"
#> [5] "minivan"    "pickup"     "subcompact"
```

```
ggplot(data=mpg,mapping=aes(x=displ,y=cty,color=class))+
  geom_point()
```



The size argument

```
ggplot(data=mpg,mapping=aes(x=displ,y=cty,color=class,size=cyl))+
  geom_point()
```



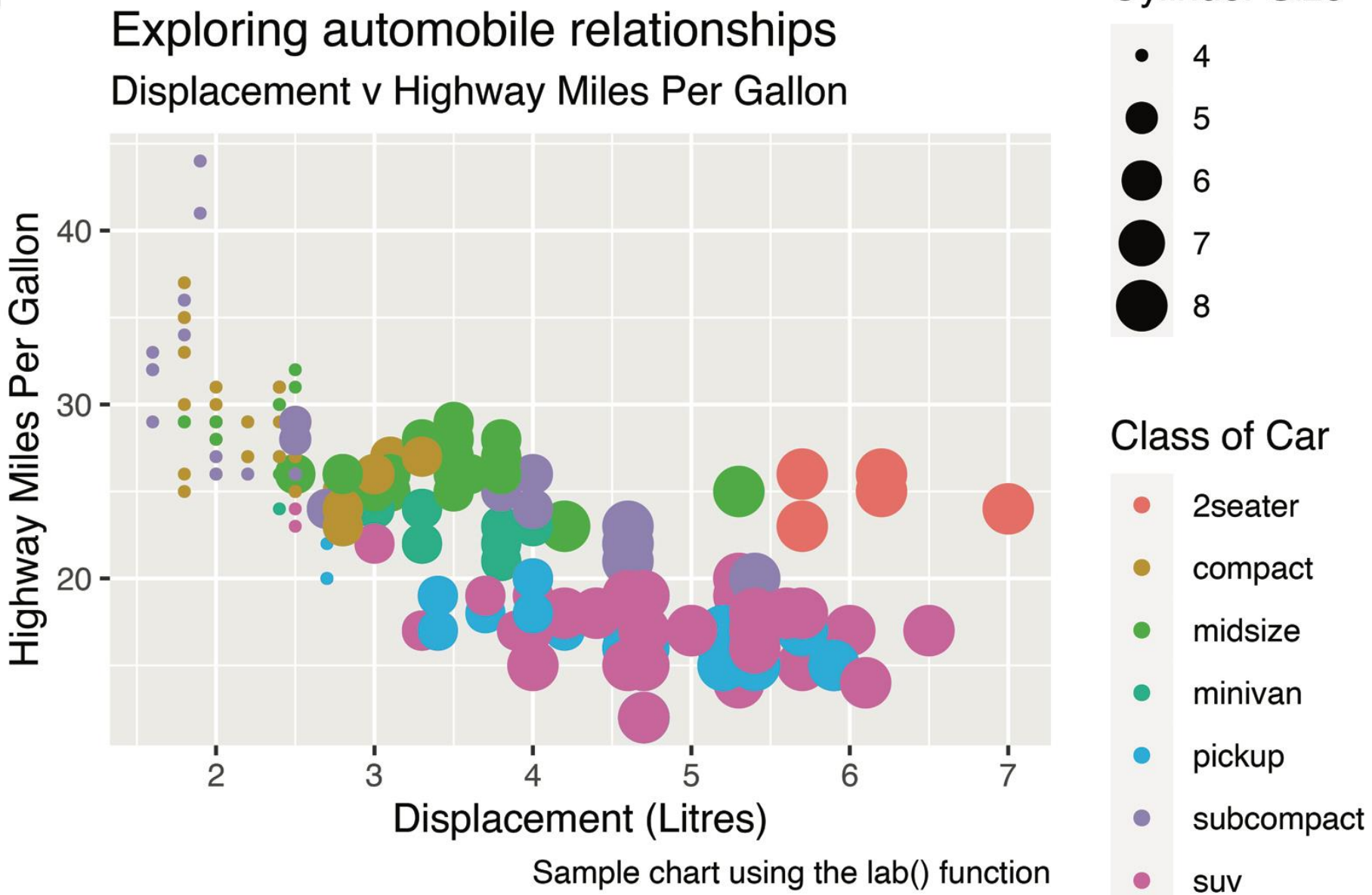
The lab() function – name-value pairs

- title provides an overall title text for the plot.
- subtitle adds a subtitle text.
- color allows you to specify the legend name for the color attribute.
- caption inserts text on the lower right-hand side of the plot.
- size, where you can name the size attribute.
- x to name the x-axis.
- y to name the y-axis.
- tag, the text for the tag label to be displayed on the top left of the plot.

Note that plots can be variables (p1)

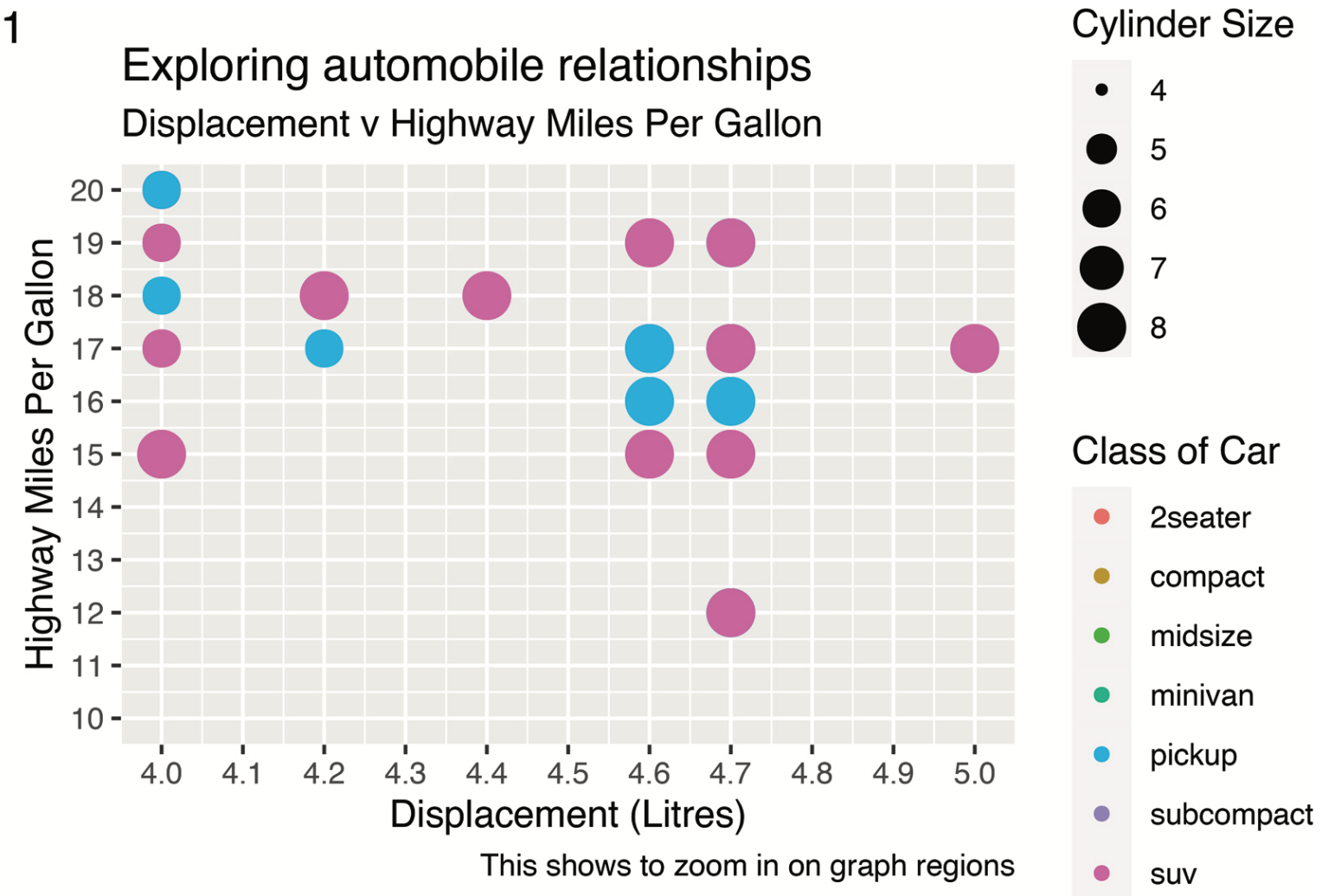
```
p1 <- ggplot(data=mpg,aes(x=displ,y=hwy,size=cyl,color=class))+
        geom_point()

p1 <- p1 +
       labs(
         title = "Exploring automobile relationships",
         subtitle = "Displacement v Highway Miles Per Gallon",
         color = "Class of Car",
         size = "Cylinder Size",
         caption = "Sample chart using the lab() function",
         tag = "Plot #1",
         x = "Displacement (Litres)",
         y = "Highway Miles Per Gallon"
       )

p1
```

Plot #1


Exploring automobile relationships
Displacement v Highway Miles Per Gallon
Highway Miles Per Gallon
40
30
20
2
3
4
5
6
7
Displacement (Litres)
Sample chart using the lab() function
Cylinder Size
4
5
6
7
8
Class of Car
2seater
compact
midsize
minivan
pickup
subcompact
suv

```
p1 <- p1 +
        scale_x_continuous(limits=c(4,5), breaks=seq(4,5,.1))+
        scale_y_continuous(limits=c(10,20),breaks=seq(10,20,1))+
        labs(caption = "This shows to zoom in on graph regions")
```

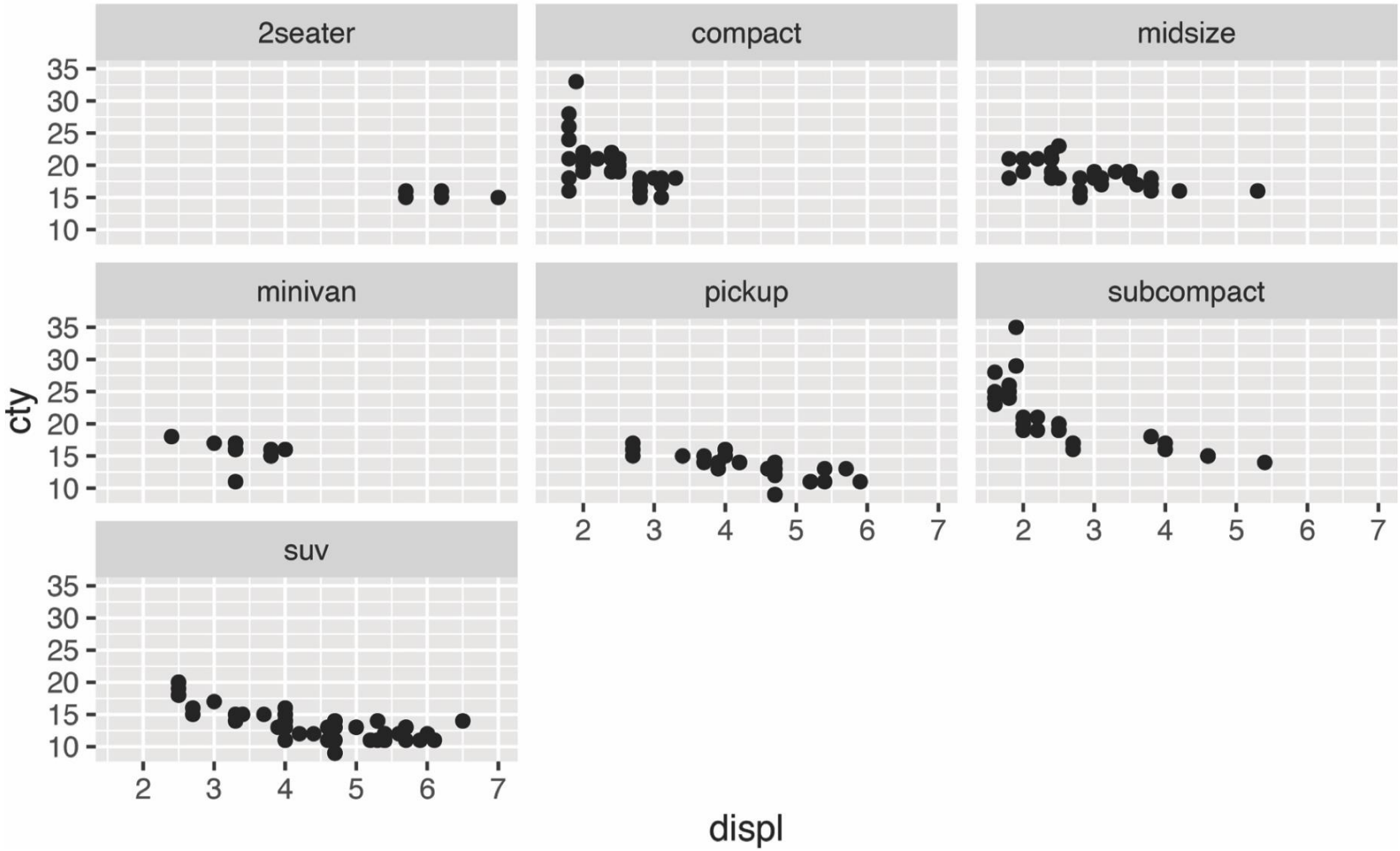
Plot #1



(5.4) Subplots with facets

- What if we needed to drill down on the plots and show, for example, the relationships for each class of car on separate plots?
- Or, in the more general case, sub-divide a plot into multiple plots based on another variable.
- The function facet_wrap() will do this in ggplot2, and all it needs as an argument is the variable for dividing the plots, which must be preceded by the tilde (~) operator.

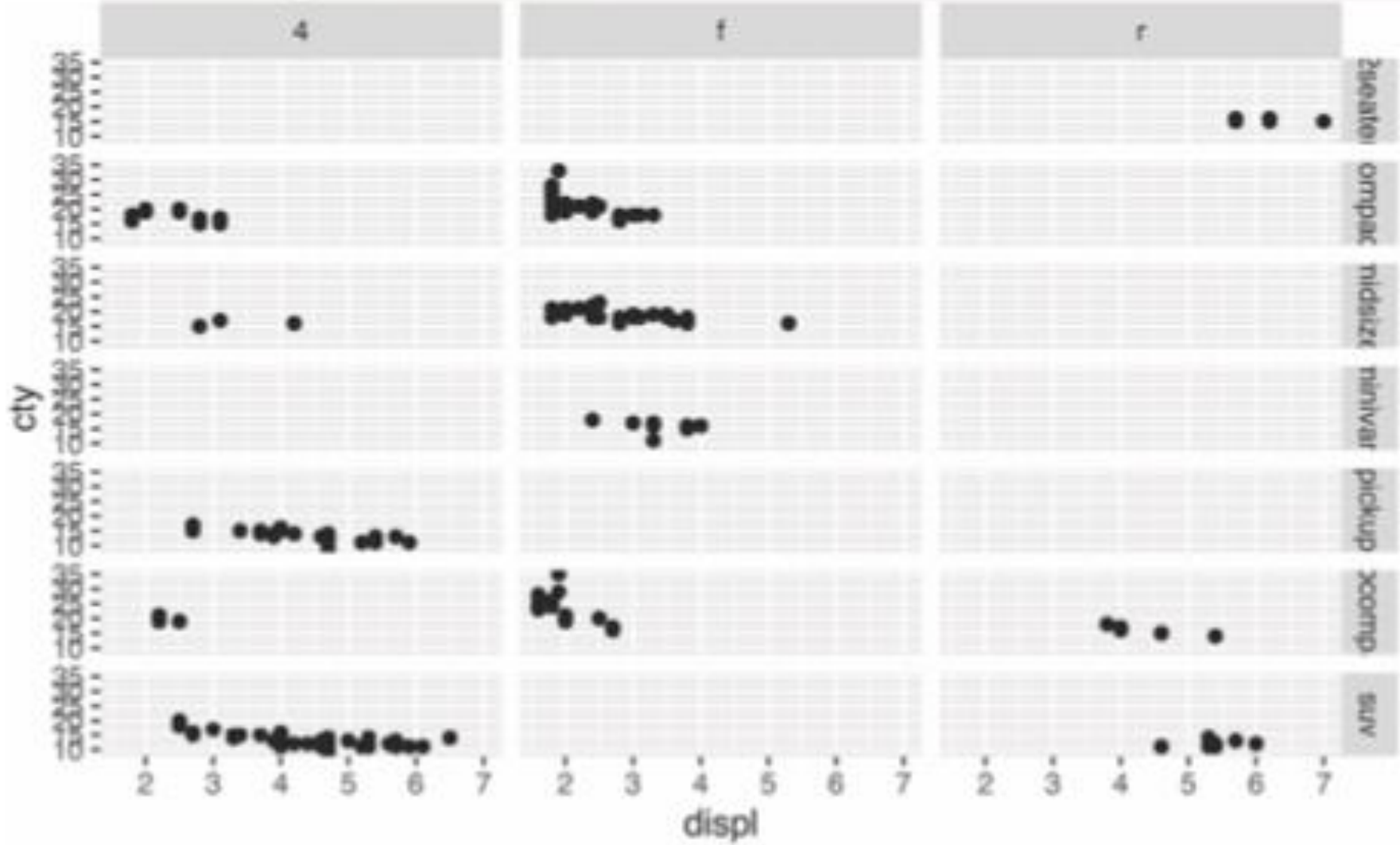
```
ggplot(data=mpg,aes(x=displ,y=cty))+
  geom_point()+
  facet_wrap(~class)
```



Using facet_grid()

- An extra variable can be added to the faceting process by using the related function facet_grid(), which takes two arguments, separated by the ~ operator.
- The first argument specifies which variable is to be mapped to each row, and the second argument identifies the variable to be represented on the columns.
- For example, we may want to generate 21 plots that show the type of drive (drv) on the columns, and the class of car (class) shown on each row.

```
ggplot(data=mpg,mapping = aes(x=displ,y=cty))+
  geom_point()+
  facet_grid(class~drv)
```

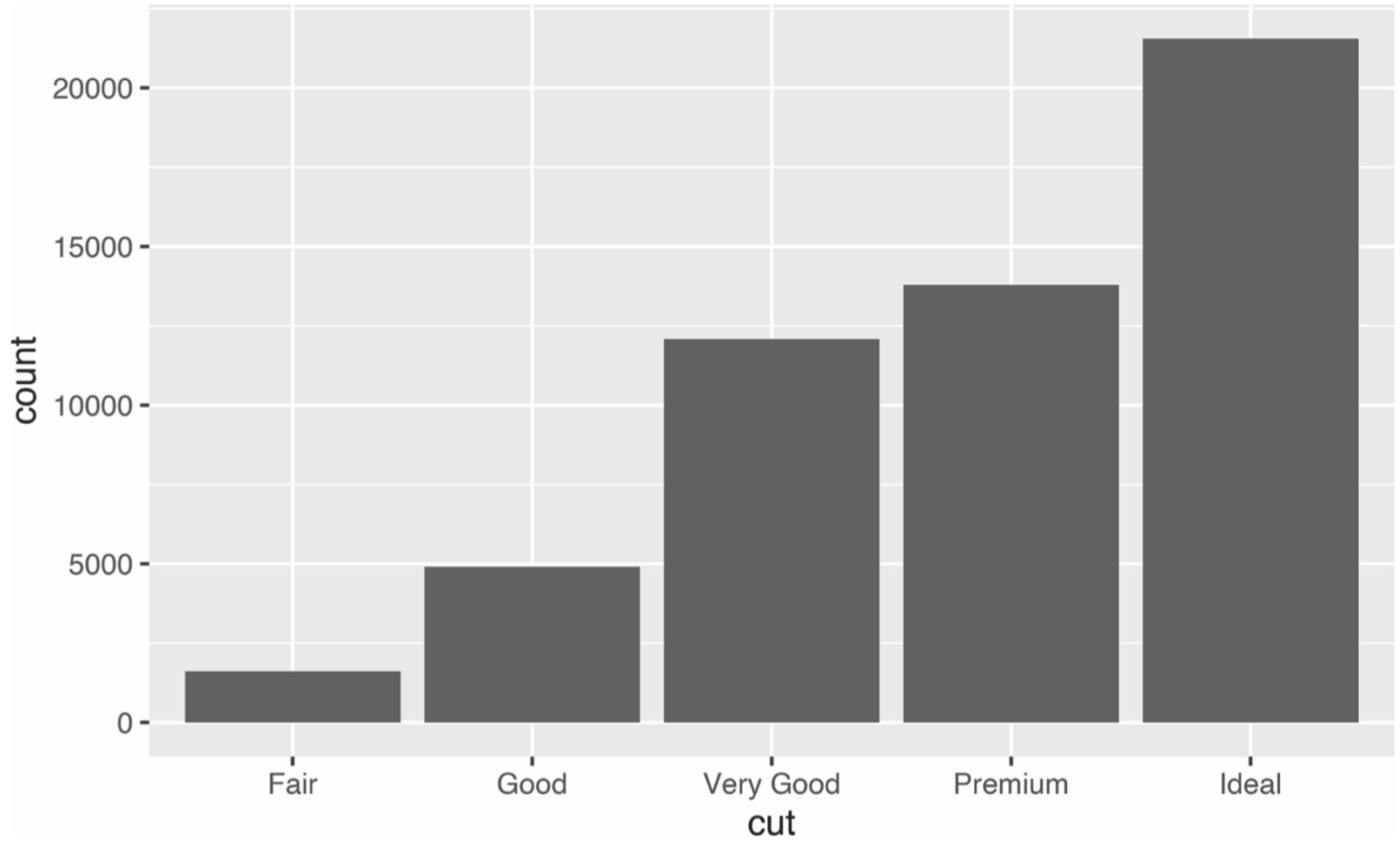


(5.5) Statistical Transformations

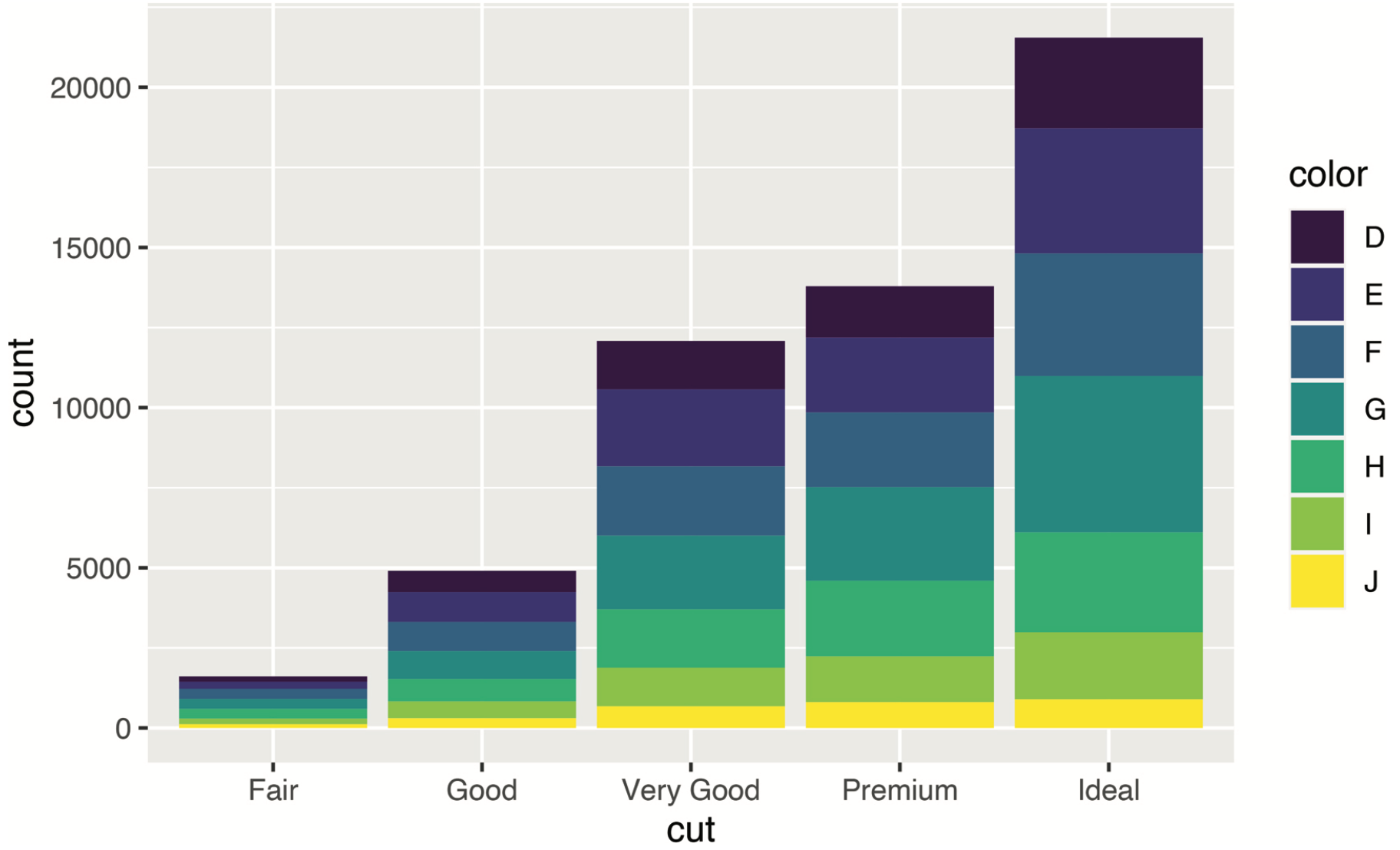
- The scatterplot that we have just explored takes observations from the dataset, and these points are plotted on the graph.
- However, there are graphs that will firstly calculate values, and then plot the results of this calculation
- In ggplot2, this activity is known as what we is termed statistical transformations.

(a) geom_bar() – Frequency count

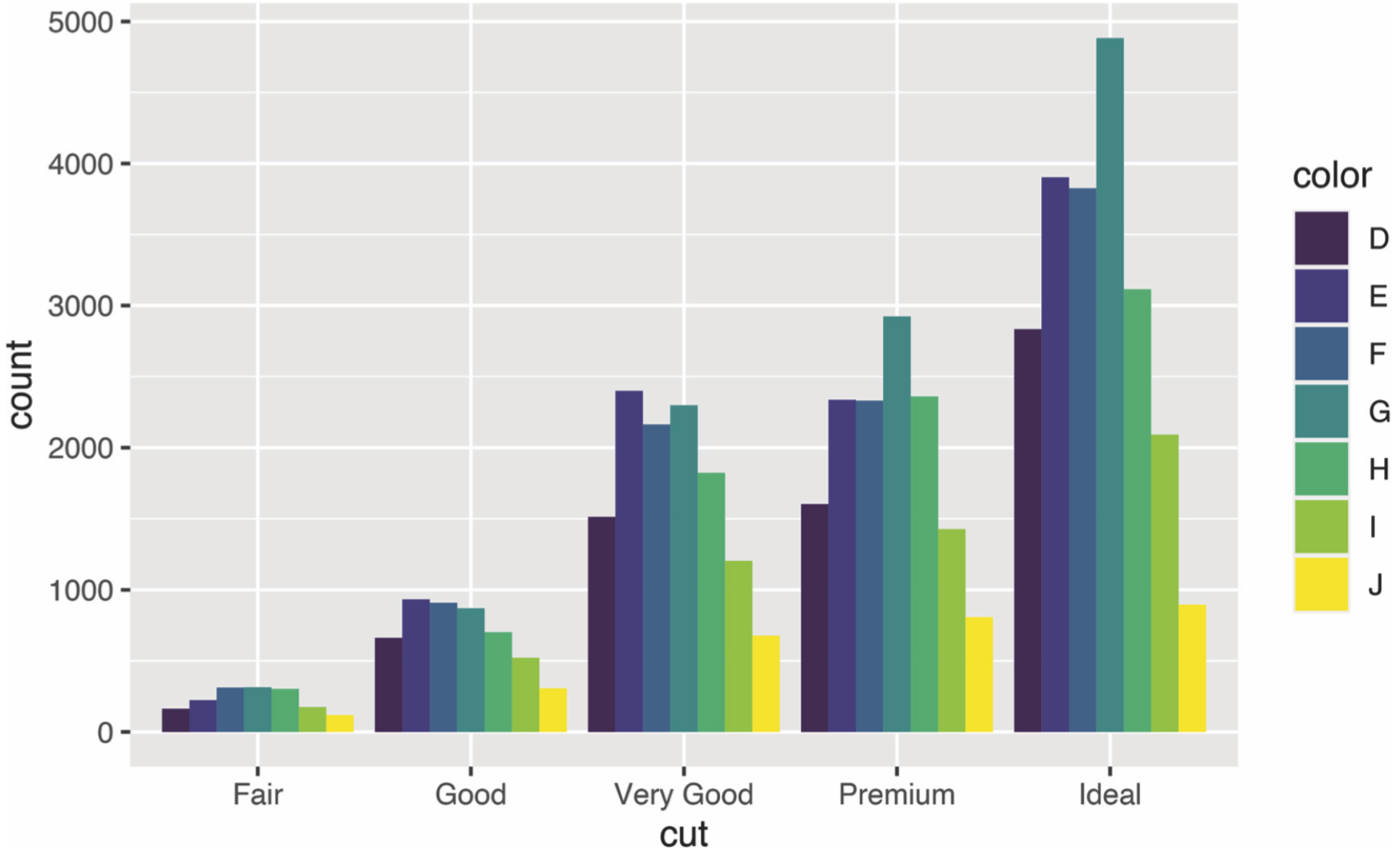
```
ggplot(data=diamonds,mapping=aes(x=cut))+
  geom_bar()
```



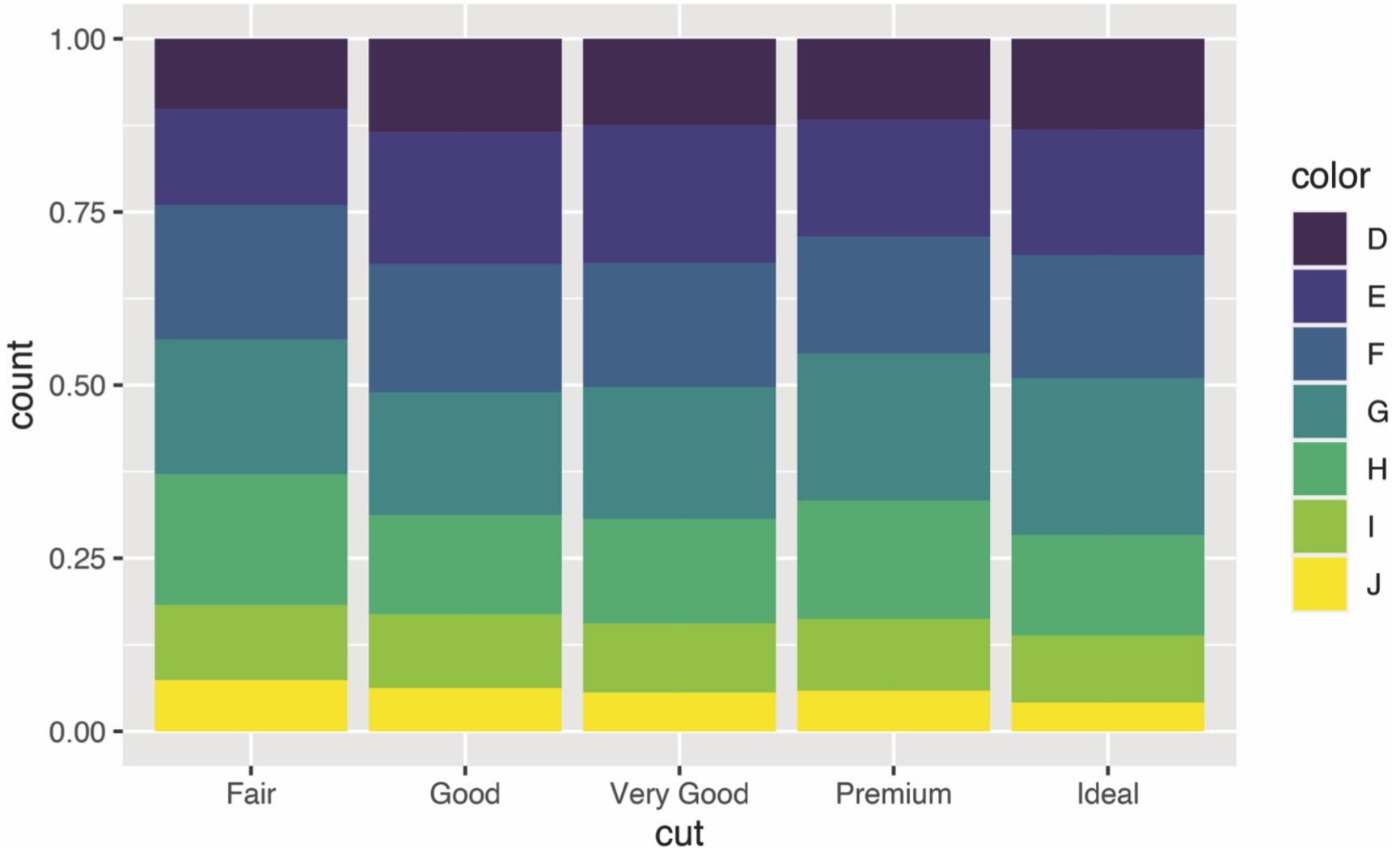
```
ggplot(data=diamonds,mapping=aes(x=cut,fill=color))+
  geom_bar()
```



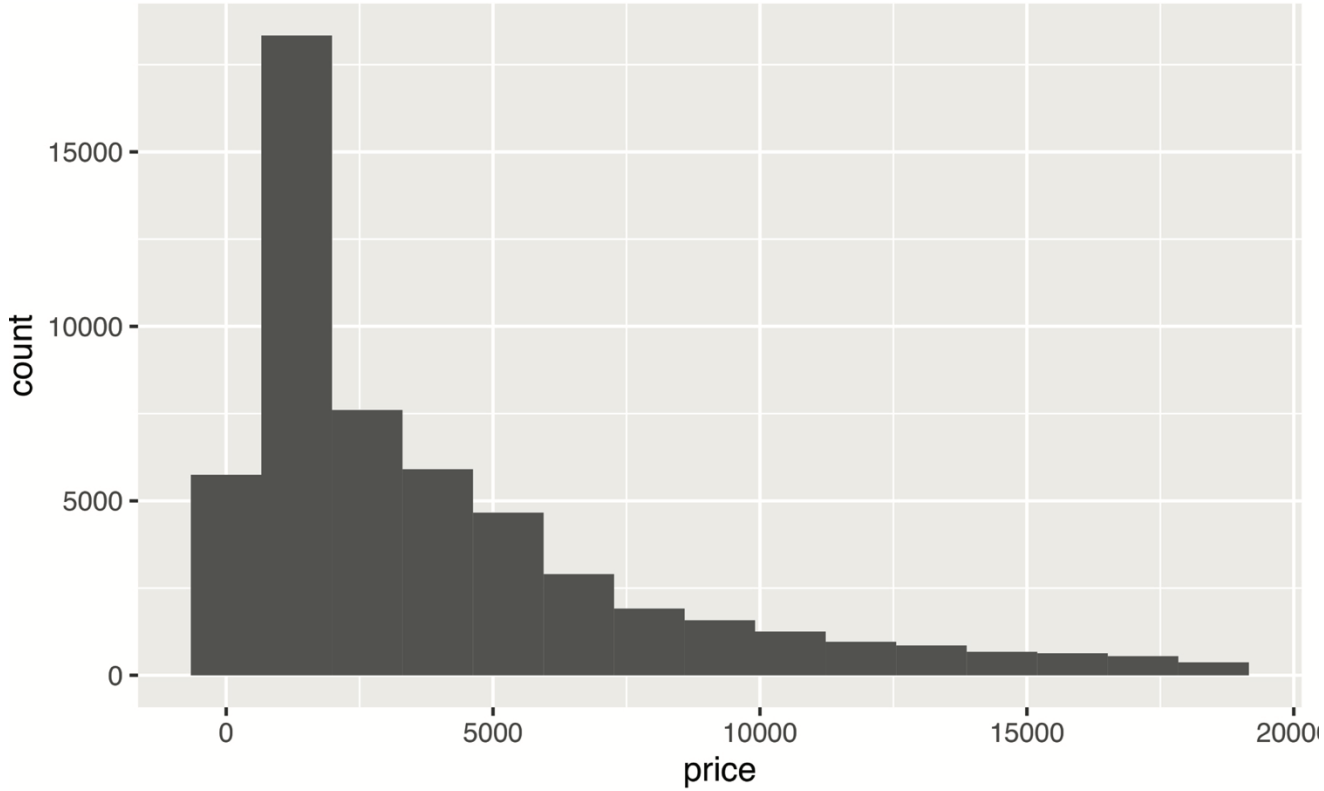
```
ggplot(data=diamonds,mapping=aes(x=cut,fill=color))+
  geom_bar(position="dodge")
```



```
ggplot(data=diamonds,mapping=aes(x=cut,fill=color))+
  geom_bar(position="fill")
```



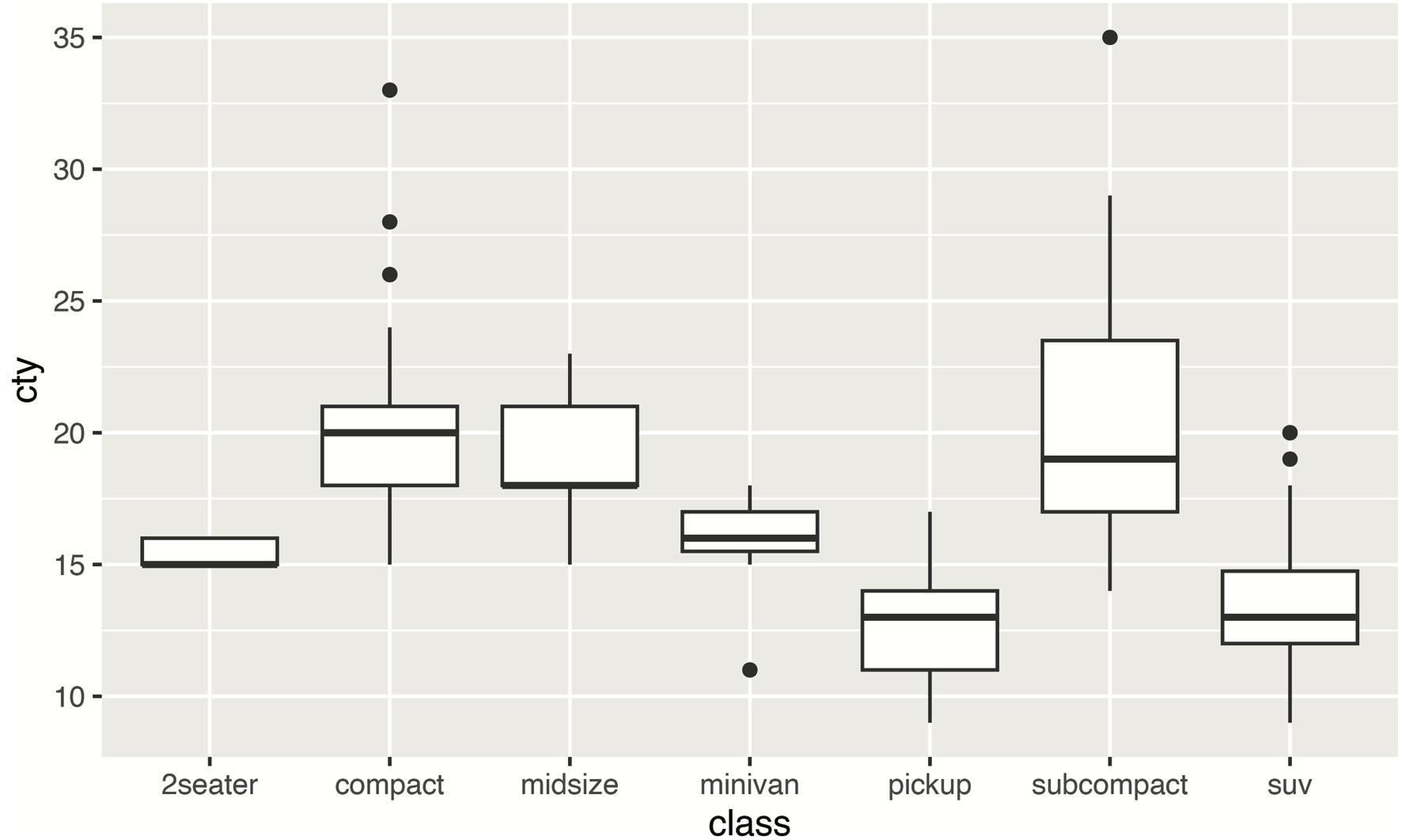
(b) Visualising distributions – geom_histogram()




```
ggplot(data=diamonds,mapping=aes(x=price))+
  geom_histogram(bins = 15)
```

geom_boxplot()

- The boxplot is a valuable graph that visualizes the distribution of a continuous variable, showing the median, the 25th to 75th percentiles, with lines drawn to the position of the value less than or equal to 1.5 times the interquartile range (IQR).
- Values outside 1.5 times the IQR are outliers, and shown as points.

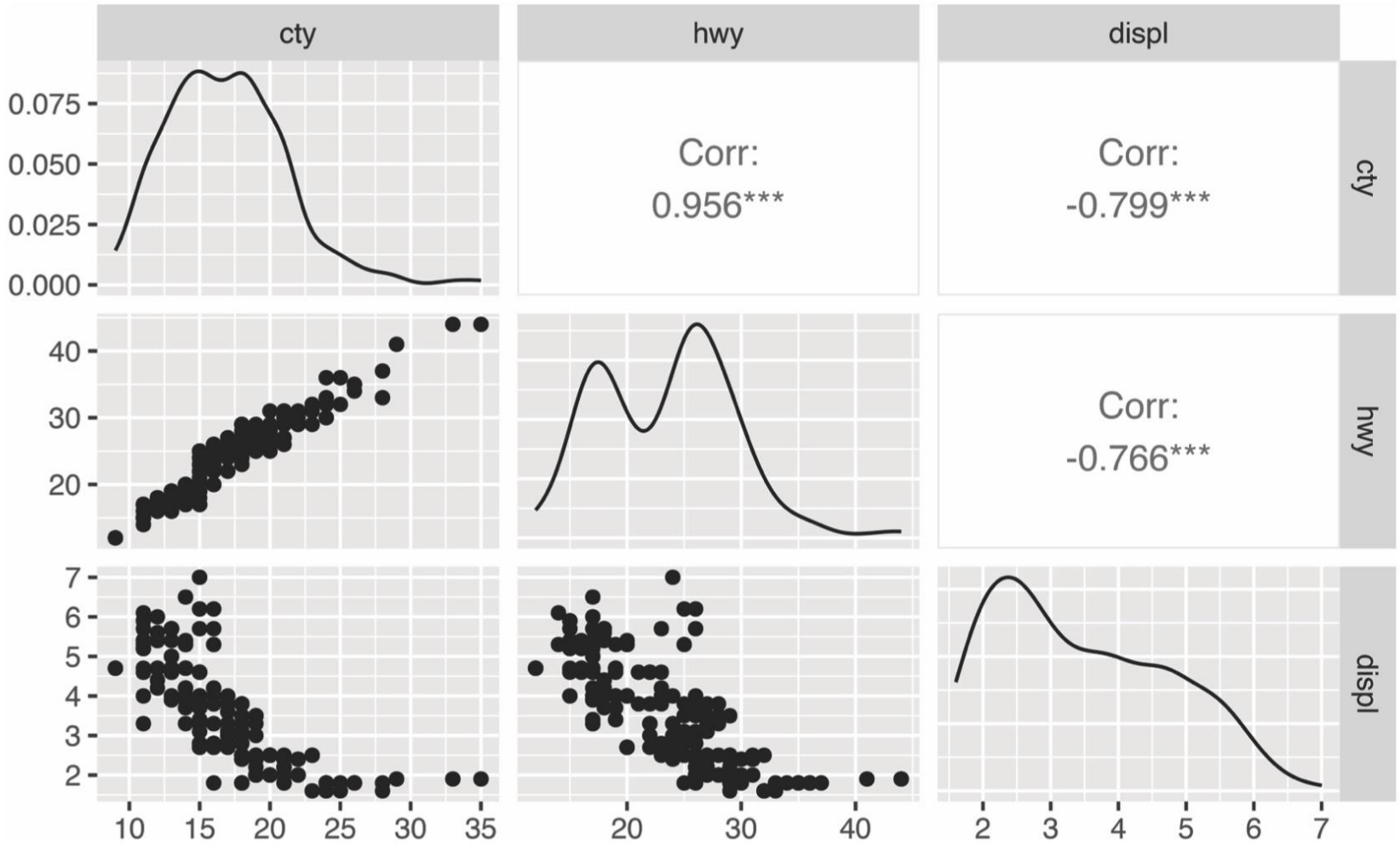



```
ggplot(data=mpg,mapping=aes(y=cty,x=class))+
  geom_boxplot()
```

Covariation with ggpairs()

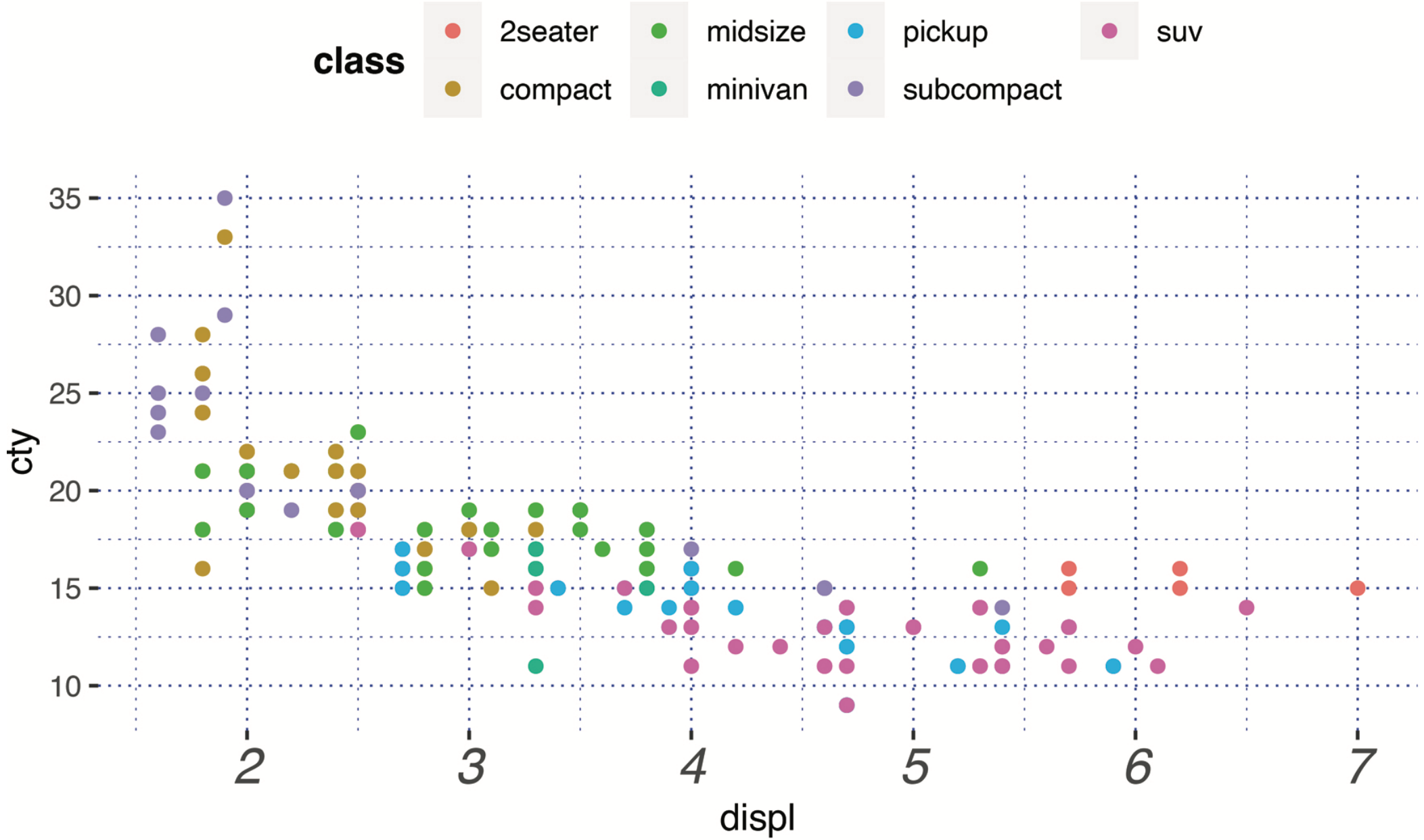
- The package GGally contains the function ggpairs() which visualizes relationships between variables, presents the density plot for each variable, and summarizes the range of correlation coefficients between continuous variables.
- It provides a useful perspective on the data
- For this example, we use the mpg dataset and the three variables cty, hwy, and displ

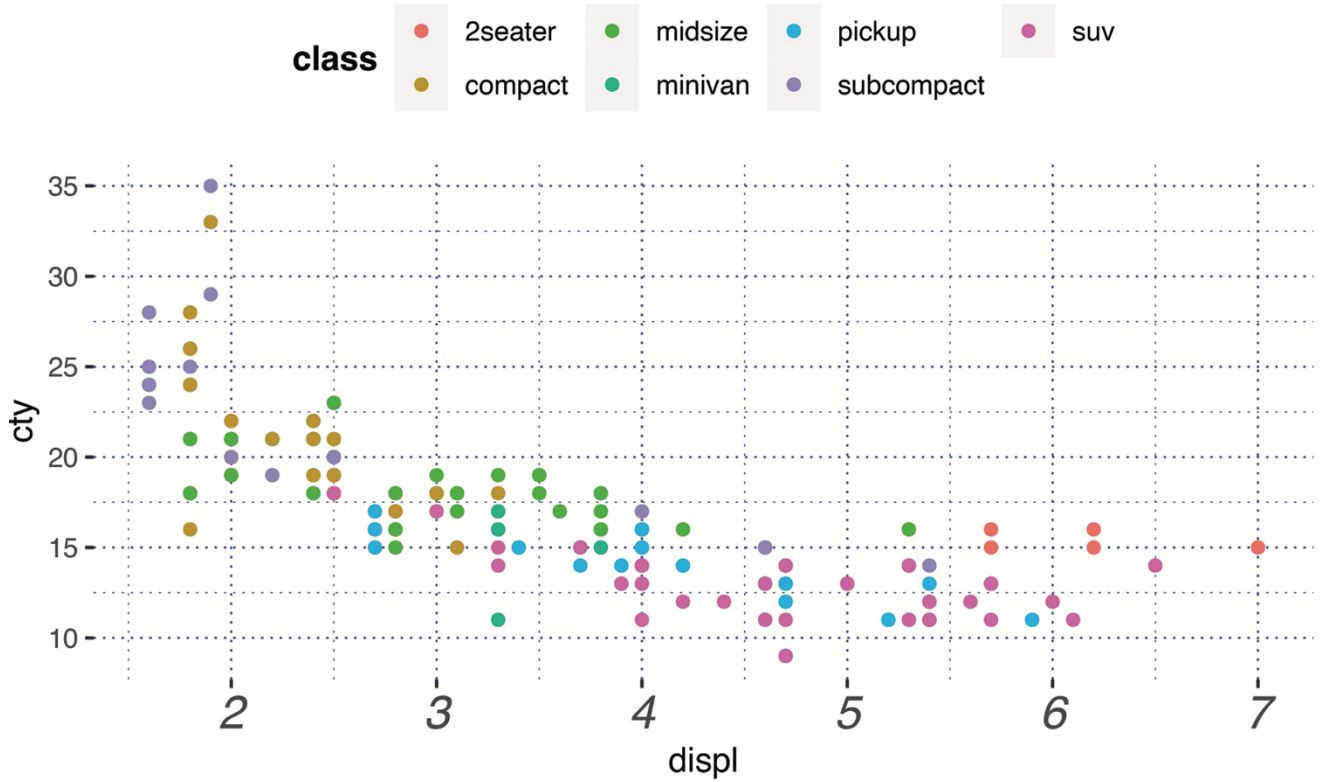
```
library(GGally)
my_vars <- subset(mpg,select=c(cty,hwy,displ))
ggpairs(my_vars)
```



(5.6) Themes

- The ggplot2 theme system enables you to control non-data elements of your plot. For example, control over elements such as fonts, ticks, legends, and backgrounds.
- Components include:
  - Element functions, to describe the visual properties of an element, e.g. element_text() for font size, color, typeface
  - The theme() function to change the default theme elements, e.g. theme(legend.position="top")
  - Complete and ready-to-use themes such as theme_bw()


class
2seater
compact
midsize
minivan
pickup
subcompact
suv
cty
35
30
25
20
15
10
2
3
4
5
6
7
displ



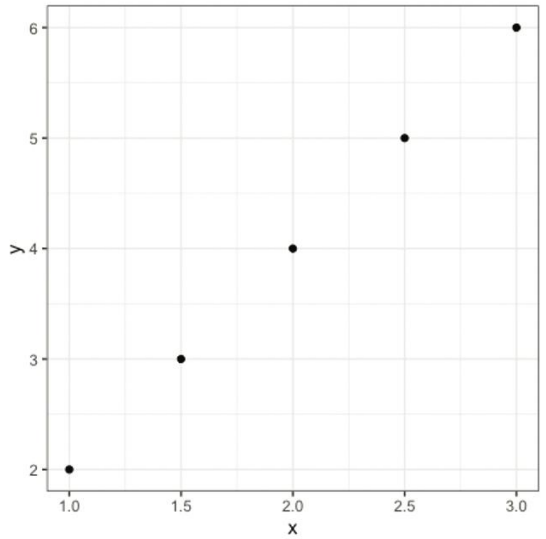

```
ggplot(data=mpg,aes(x=displ,y=cty,color=class))+
  geom_point()+
  theme(legend.title      = element_text(face="bold"),
        legend.position   = "top",
        axis.text.x       = element_text(size=15,face="italic"),
        panel.background  = element_rect(fill="white"),
        panel.grid        = element_line(color = "blue",
                                         linewidth = 0.3,
                                         linetype = 3))
```

Available themes…

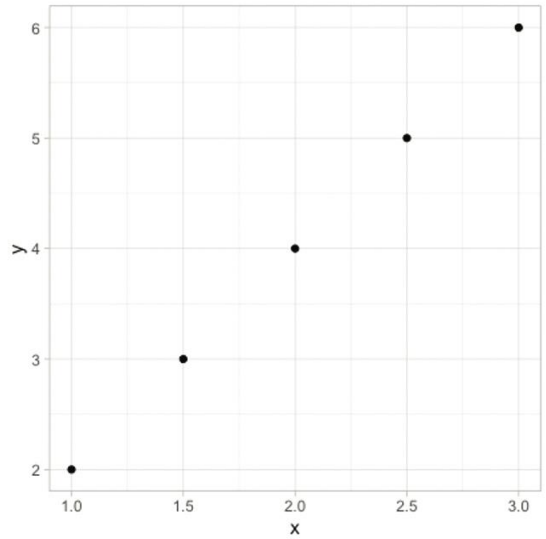
Theme	Description
theme_bw()	a classic dark-on-light theme.
theme_light()	a theme with light gray lines and axes in order to provide more emphasis on the data.
theme_dark()	referred to in the documentation as the dark cousin of theme_light().
theme_minimal()	a minimalistic theme with no background annotations.
theme_classic()	a classic-looking theme, with no gridlines
theme_void()	a completely empty theme.

```
d <- tibble(x=seq(1,3,by=0.5),y=2*x)
plot <- ggplot(data=d,mapping=aes(x=x,y=y))+
          geom_point()
```

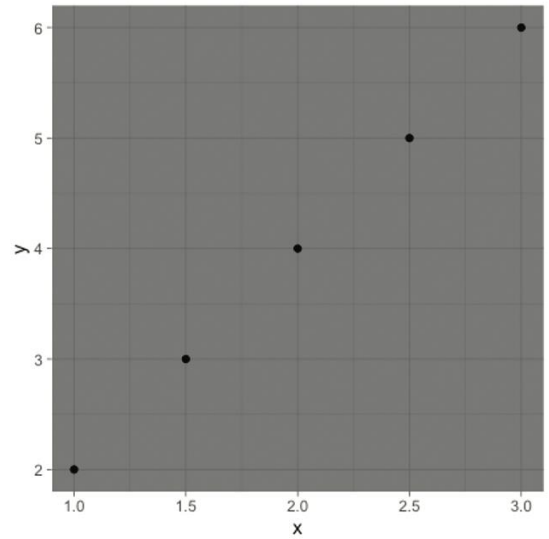
plot+theme_bw()



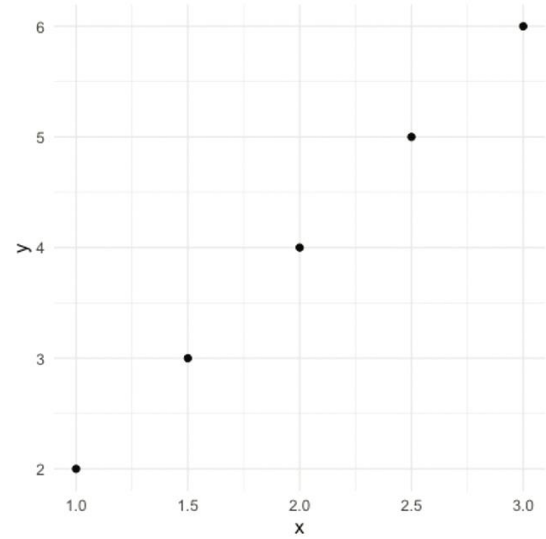
plot+theme_light()



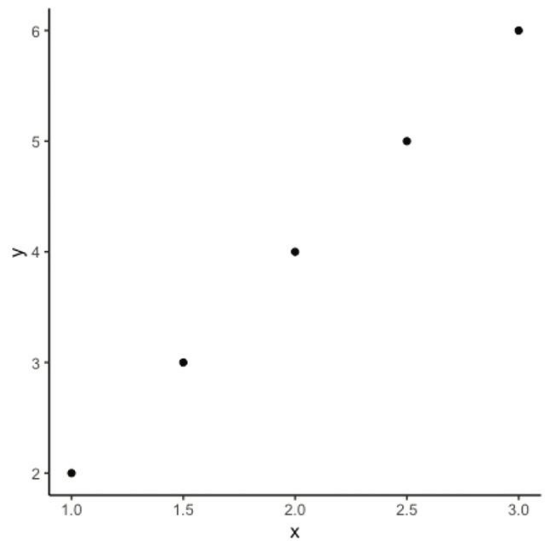
plot+theme_dark()



plot+theme_minimal()



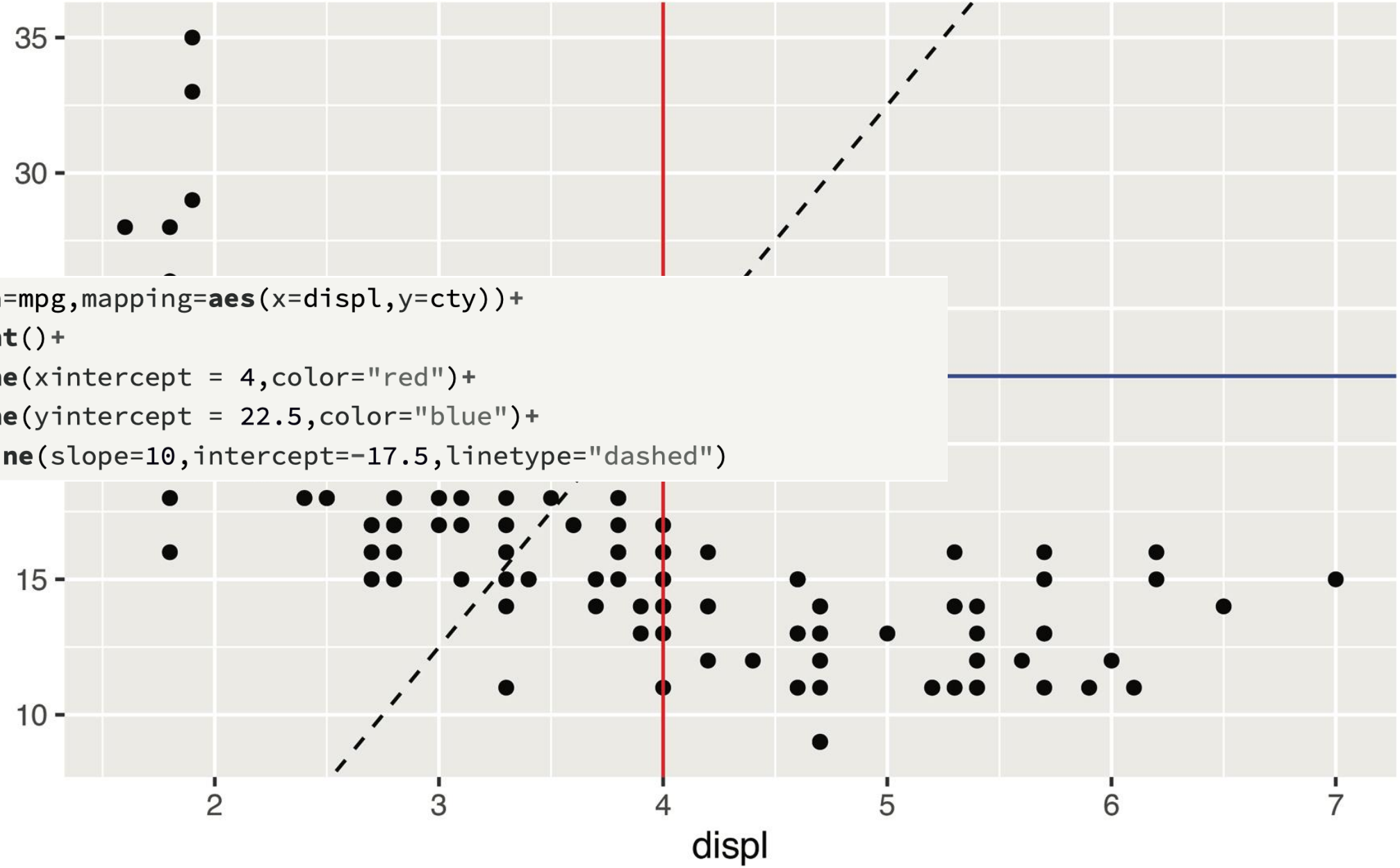
plot+theme_classic()



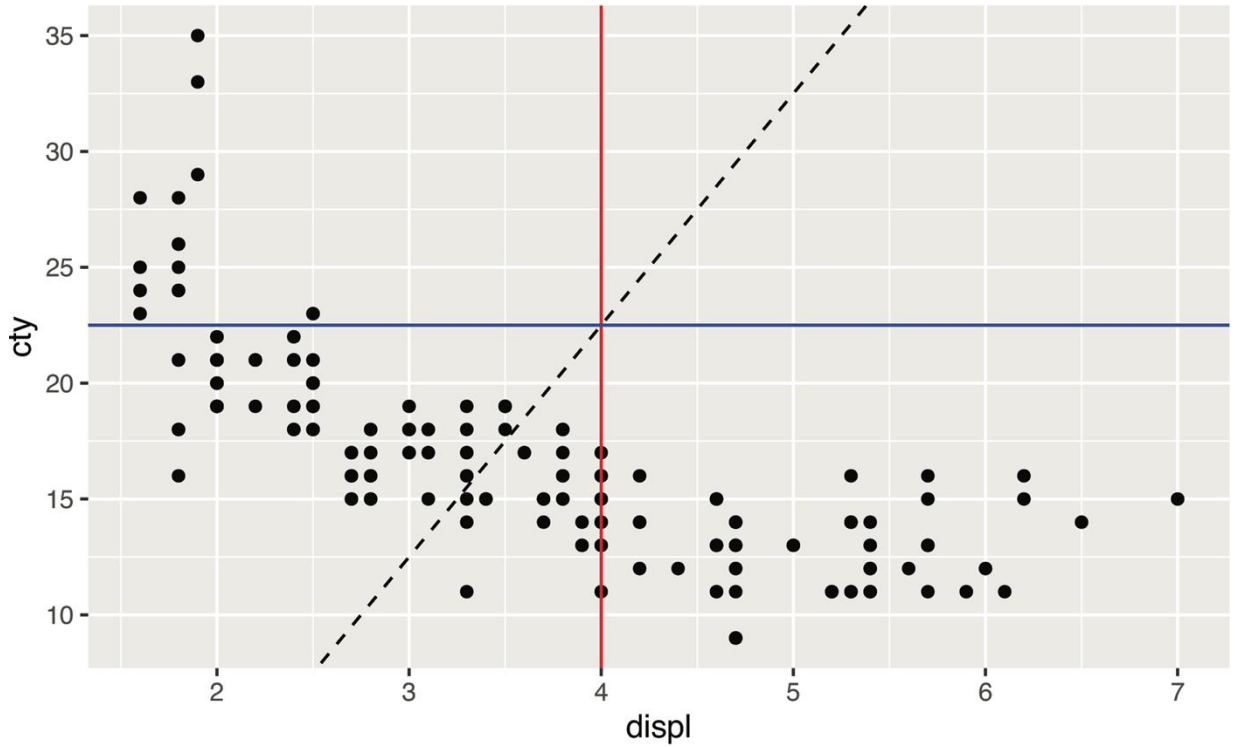
plot+theme_void()

(5.7) Adding lines to a plot

- There may be situations where adding a layer containing a line is required for your plot.
- There are three functions for doing this, specifically:
  - geom_vline(), where the argument xintercept is fixed to position the vertical line.
  - geom_hline(), with the argument yintercept used to position the horiztonal line.
  - geom_abline(), where both the arguments slope and intercept can be set to generate the output line.




```r
ggplot(data=mpg,mapping=aes(x=displ,y=cty))+
  geom_point()+
  geom_vline(xintercept = 4,color="red")+
  geom_hline(yintercept = 22.5,color="blue")+
  geom_abline(slope=10,intercept=-17.5,linetype="dashed")
```




```
ggplot(data=mpg,mapping=aes(x=displ,y=cty))+
  geom_point()+
  geom_vline(xintercept = 4,color="red")+
  geom_hline(yintercept = 22.5,color="blue")+
  geom_abline(slope=10,intercept=-17.5,linetype="dashed")
```

(5.8) Mini-case – Storm Ophelia 2017

- We take our first look at the dataset aimsir17, which contains hourly observations from 25 weather stations in Ireland, all recorded in 2017
- There are 219,000 records in all (25 stations × 365 days × 24 hourly observations)
- Data recorded includes the month, day, hour, time, rainfall, temperature, humidity, mean sea level pressure, wind speed, and wind direction.
- Note that the date variable is a special R data format that is recognized by ggplot2 for time series plots

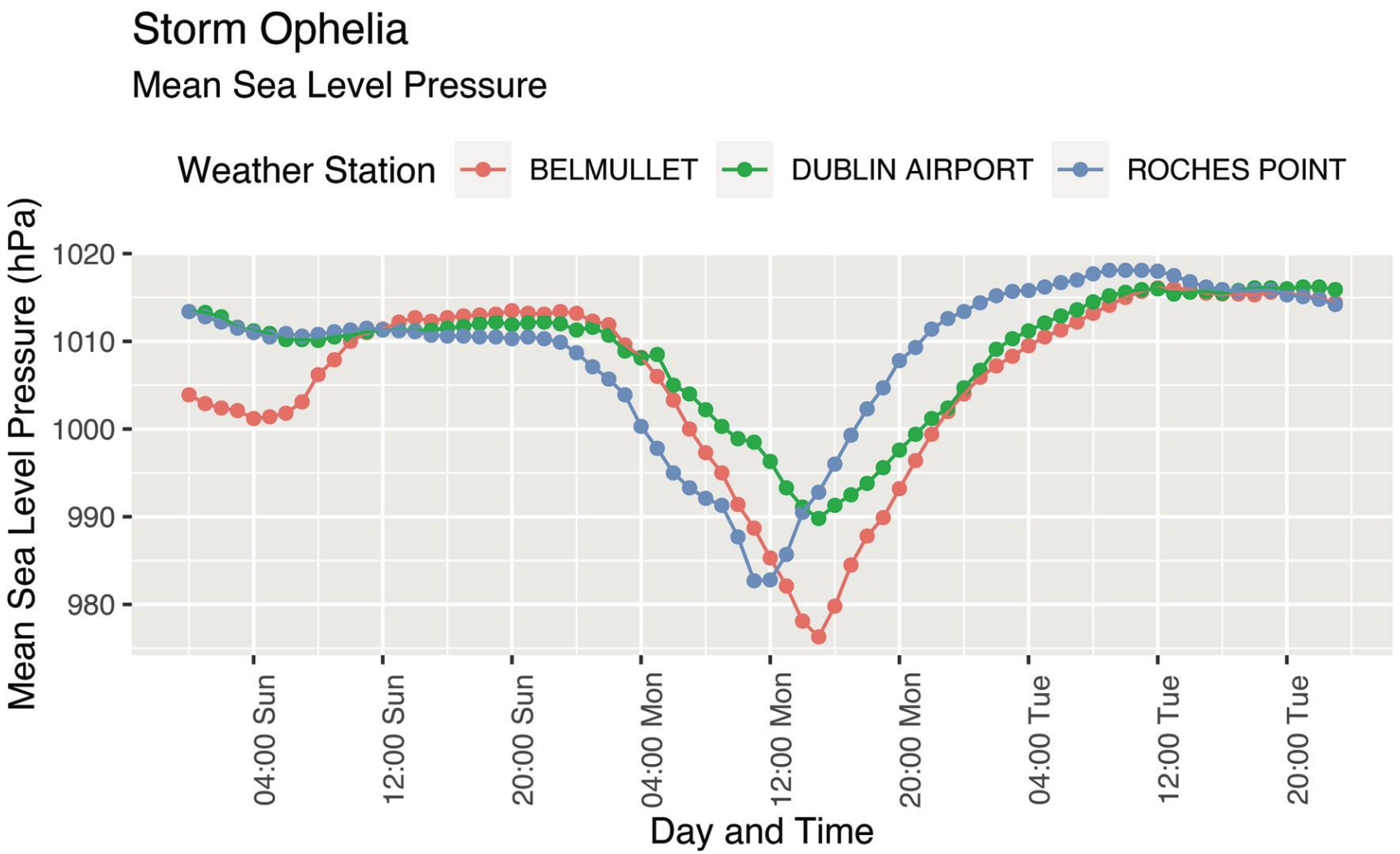
```
library(aimsir17)

observations
#> # A tibble: 219,000 x 12
#>    station  year month   day  hour date                 rain
#>    <chr>   <dbl> <dbl> <int> <int> <dttm>              <dbl>
#>  1 ATHENRY  2017     1     1     0 2017-01-01 00:00:00   0
#>  2 ATHENRY  2017     1     1     1 2017-01-01 01:00:00   0
#>  3 ATHENRY  2017     1     1     2 2017-01-01 02:00:00   0
#>  4 ATHENRY  2017     1     1     3 2017-01-01 03:00:00   0.1
#>  5 ATHENRY  2017     1     1     4 2017-01-01 04:00:00   0.1
#>  6 ATHENRY  2017     1     1     5 2017-01-01 05:00:00   0
#>  7 ATHENRY  2017     1     1     6 2017-01-01 06:00:00   0
#>  8 ATHENRY  2017     1     1     7 2017-01-01 07:00:00   0
#>  9 ATHENRY  2017     1     1     8 2017-01-01 08:00:00   0
#> 10 ATHENRY  2017     1     1     9 2017-01-01 09:00:00   0
#> # ... with 218,990 more rows, and 5 more variables: temp <dbl>,
#> #   rhum <dbl>, msl <dbl>, wdsp <dbl>, wddir <dbl>
```

Data preparation

- We first need to “drill down” and generate a reduced dataset for the three days in question, and we will also select a number of weather stations from different parts of Ireland.
- These are Belmullet (north west), Roches Point (south), and Dublin Airport (east).
- The following code uses the function subset() to filter the dataset, and the & operator is used as we have more than one filtering condition.
- We also use the %in% operator, which is convenient

```
storm <- observations |>
           subset(month==10 &
                  day %in% 15:17 &
                  station %in% c("BELMULLET",
                                 "ROCHES POINT",
                                 "DUBLIN AIRPORT"),
                  select=c(station,date,temp,msl,wdsp))
storm
#> # A tibble: 216 x 5
#>    station   date                 temp   msl  wdsp
#>    <chr>     <dttm>              <dbl> <dbl> <dbl>
#>  1 BELMULLET 2017-10-15 00:00:00  15.5 1004.    30
#>  2 BELMULLET 2017-10-15 01:00:00  15.9 1003.    33
#>  3 BELMULLET 2017-10-15 02:00:00  15.3 1002.    33
#>  4 BELMULLET 2017-10-15 03:00:00  15.1 1002.    35
#>  5 BELMULLET 2017-10-15 04:00:00  15.1 1001.    33
```

Exploration 1: Mean sea level pressure



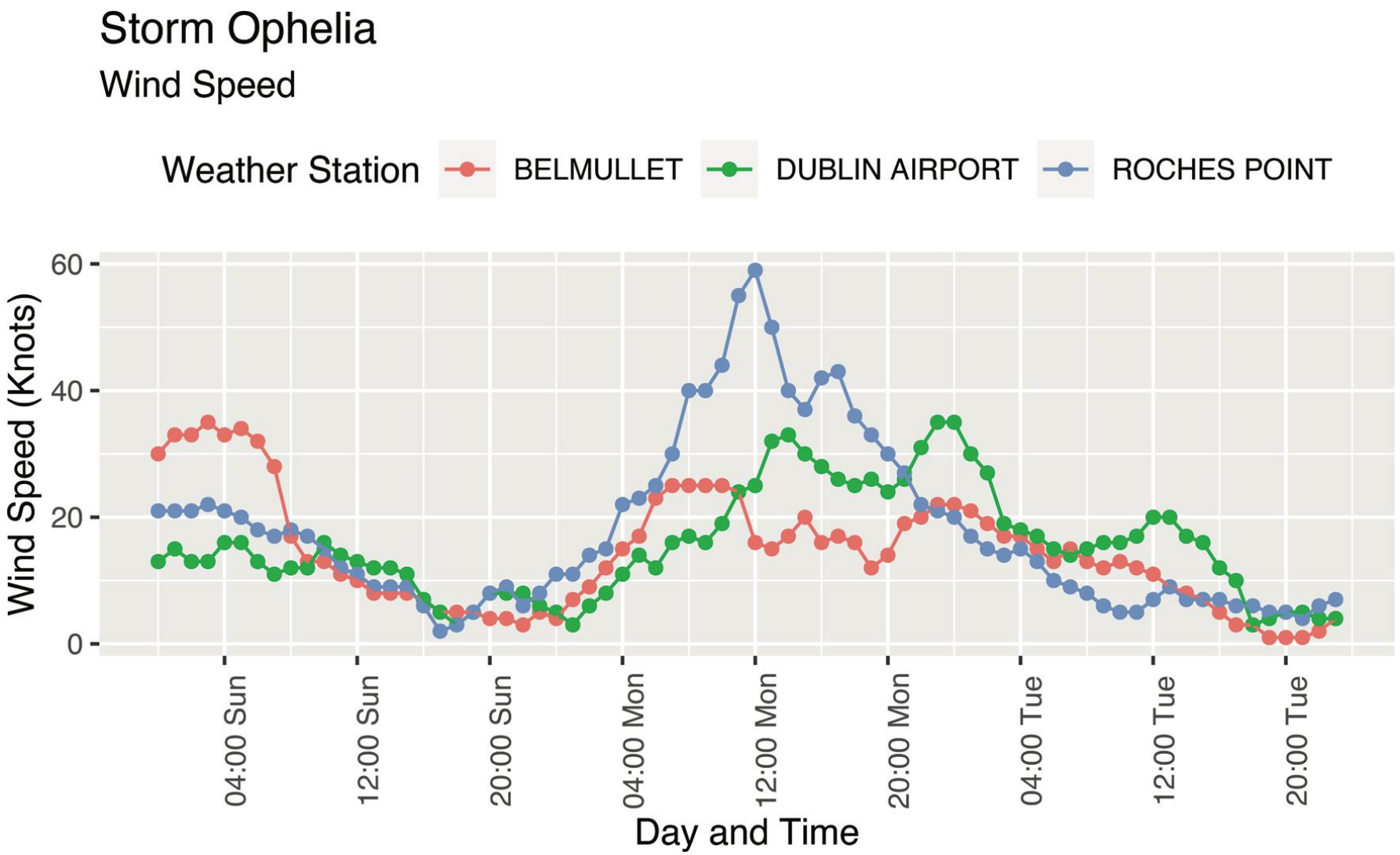
```
ggplot(data=storm,aes(x=date,y=msl,color=station))+
  geom_point()+
  geom_line() +
  theme(legend.position = "top",
        axis.text.x = element_text(angle = 90))+
  scale_x_datetime(date_breaks = "8 hour",date_labels = "%H:%M %a")+
  labs(title="Storm Ophelia",
       subtitle = "Mean Sea Level Pressure",
       x="Day and Time",
       y="Mean Sea Level Pressure (hPa)",
       color="Weather Station")
```

scale_x_datetime()

- scale_x_datetime() provides an excellent way to format dates, and has two arguments
  - date_breaks which specifies the x-axis points that will contain date labels (every 8 hours)
  - date_labels which allows you to configure what will be printed (hour, minute, and day of week).
  - '%H:%M %a" will combine the hour (24-hour clock), a colon, the minute (00–59), a blank space followed by the abbreviated day of the week.

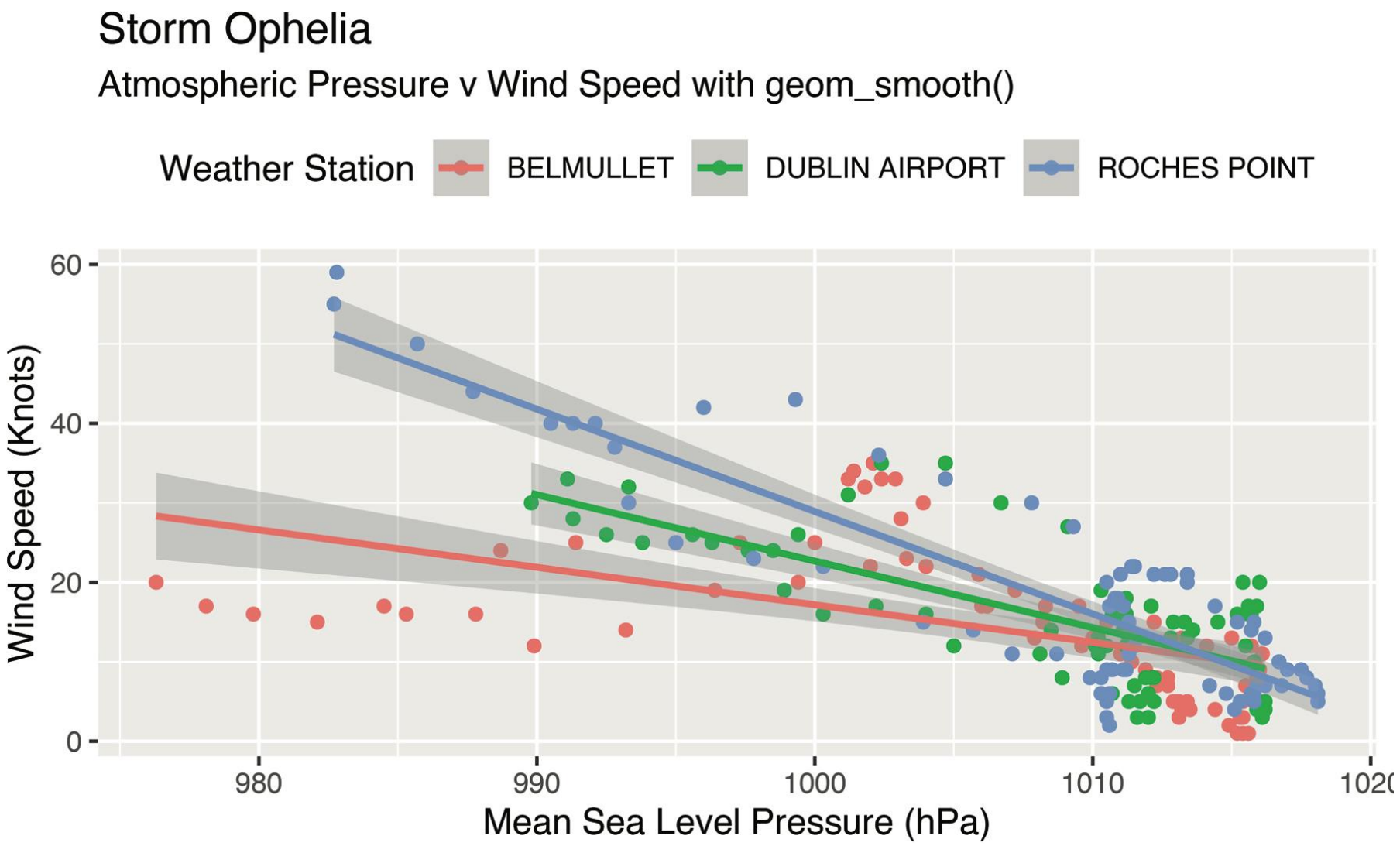
String	Meaning for `date_labels`
%S	Second (00–59)
%M	Minute (00–59)
%l	Hour, 12 hour clock (1–12)
%I	Hour, 12 hour clock (01–12)
%p	am/pm
%H	Hour, 24-hour clock (00–23)
%a	Day of week, abbreviated (Mon–Sun)
%A	Day of week, full (Monday–Sunday)
%e	Day of month (1–31)
%d	Day of month (00–31)
%m	Month, numeric (01–12)
%b	Month, abbreviated (Jan–Dec)
%B	Month, full (January–December)
%y	Year, without century (00–99)
%Y	Year, with century (0000–9999)

Exploration 2: Average hourly wind speed



```
ggplot(data=storm,aes(x=date,y=wdsp,color=station))+
  geom_point()+
  geom_line()+
  theme(legend.position = "top",
        axis.text.x = element_text(angle = 90))+
  scale_x_datetime(date_breaks = "8 hour",date_labels = "%H:%M %a")+
  labs(title="Storm Ophelia",
       subtitle = "Wind Speed",
       x="Day and Time",
       y="Wind Speed (Knots)",
       color="Weather Station")
```

Exploration 3: Associations between MSL and wind



```
ggplot(data=storm,aes(x=msl,y=wdsp,color=station))+
  geom_point()+
  geom_smooth(method="lm")+
  theme(legend.position = "top")+
  labs(title="Storm Ophelia",
       subtitle="Atmospheric Pressure v Wind Speed with geom_smooth()",
       x="Mean Sea Level Pressure (hPa)",
       y="Wind Speed (Knots)",
       color="Weather Station")
```

(5.9) Summary Functions

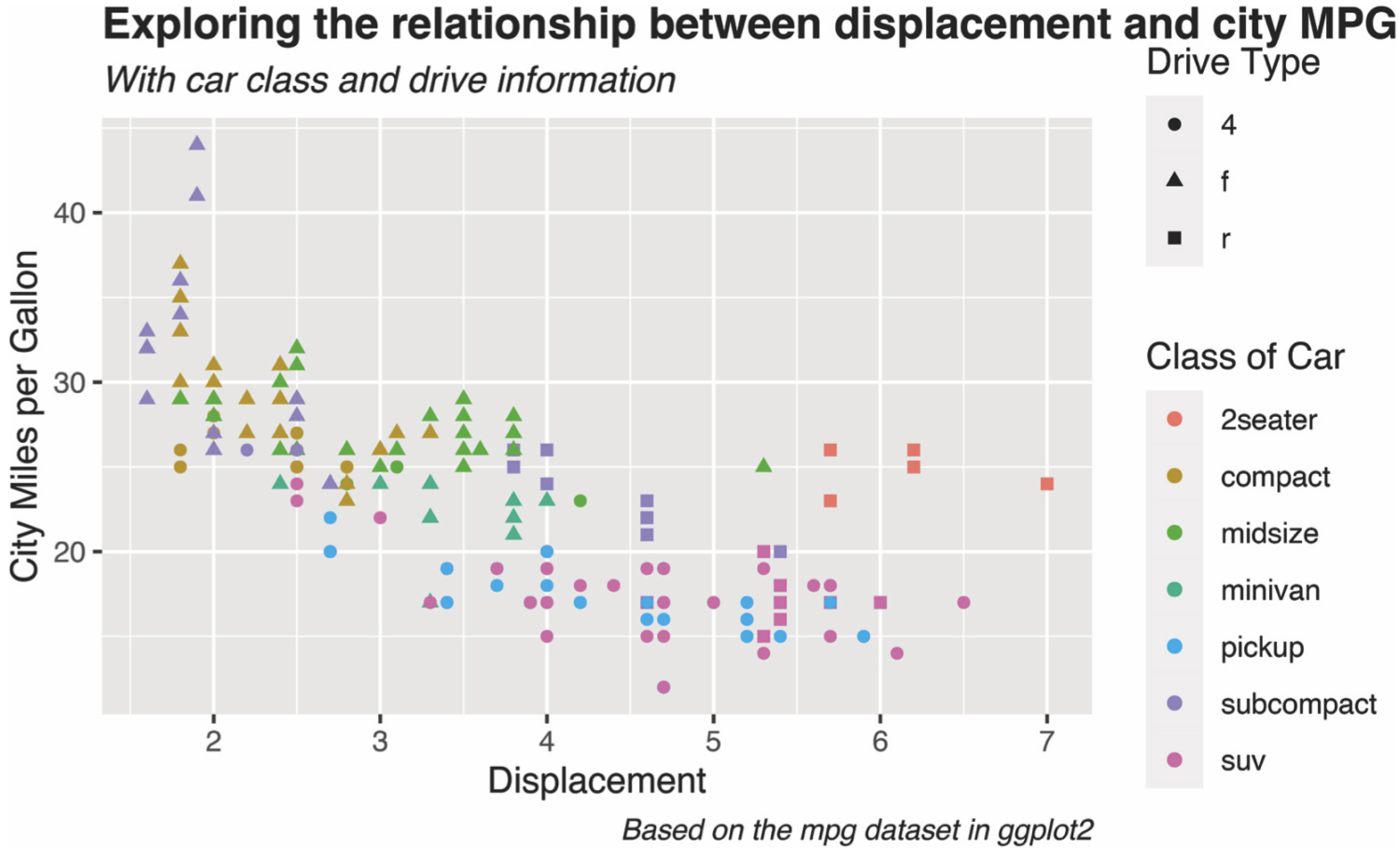
Function	Description
aes()	Maps variables to visual properties of geoms.
element_line()	Specify properties of a line, used with theme().
element_rect()	Specify properties of borders and backgrounds.
element_text()	Specify properties of text.
facet_wrap()	Creates panels based on an input variable
facet_grid()	Creates a matrix of panels for two variables.
ggplot()	Initializes a ggplot object.
geom_point()	Used to create scatterplots.
geom_abline()	Draws a line with a given slope and intercept.
geom_bar()	Constructs a bar chart based on count data.

geom_boxplot()	Displays summary of distribution.
geom_density()	Computes and draws the kernel density estimate.
geom_freqpoly()	Visualize counts of one or more variables.
geom_histogram()	Visualize the distribution of a variable.
geom_hline()	Draws a horizonal line.
geom_vline()	Draws a vertical line.
geom_smooth()	Visualization aid to assist in showing patterns.
ggpairs()	Creates a matrix of plots (library GGally).
labs()	Used to modify axis, legend and plot labels.

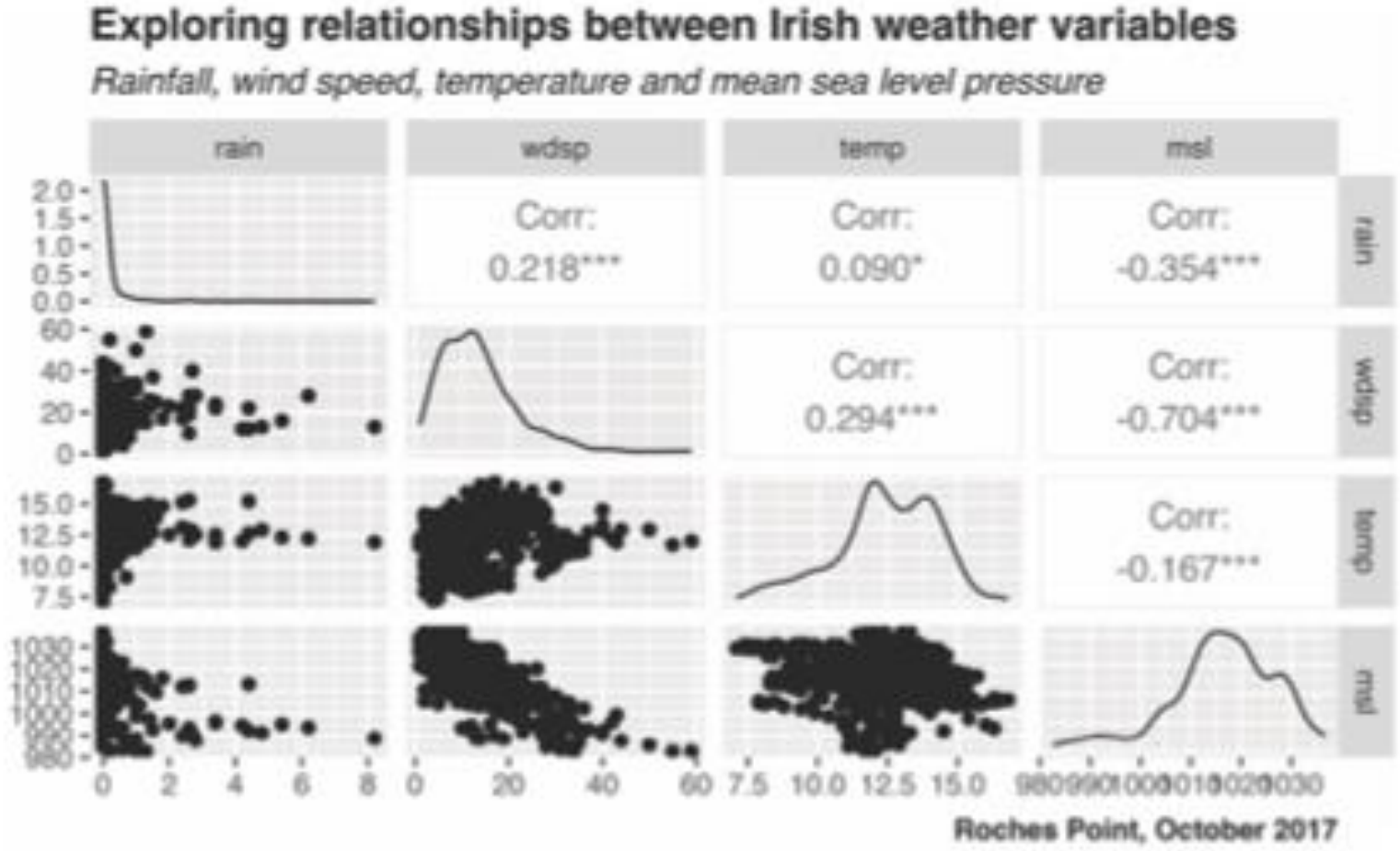
Function	Description
scale_x_continuous()	Configure the x-axis scales.
scale_y_continuous()	Configure the y-axis scales.
scale_x_datetime()	Used to configure date objects on the x-axis.
scale_y_datetime()	Configure date objects on the y-axis.
theme()	Customize the non-data components of your plots.
theme_grey()	Signature theme for ggplot2.
theme_bw()	Dark-on-light theme.
theme_light()	Theme with light gray lines and axes.
theme_dark()	Similar to theme_light(), darker background.
theme_minimal()	Minimalistic theme.
theme_classic()	Theme with no background annotations.
theme_void()	Completely empty theme.

(5.10) Exercises

1. Generate the following plot from the `mpg` tibble in `ggplot2`. The x-variable is `displ` and the y-variable `cty`. Make use of the `lab()` and `theme()` functions.



2. Use `ggpairs()` to explore a subset of the Irish weather dataset (`aimsir17`), and the possible relationships between the variables `rain`, `wdsp`, `temp` and `msl`. Use the station "ROCHES POINT" and all the observations from the month of October (month 10), which can be retrieved using the `subset()` function.



3. Using the function `subset()`, extract observations for three weather stations *Dublin Airport*, *Malin Head*, and *Belmullet*, and generate the following comparative temperature box plot summaries over the twelve months of the year. In the plot, convert the month variable to a factor value, using the function `factor()`. Note that the station names are stored as uppercase.