Software Sustainability Hands-on

andy south

2016-07-27

This tutorial will be about you doing stuff. I will get you to try things probably before you understand fully what they are. Worry not.

1. It doesn't matter if you make mistakes.
2. We will come back to some of the concepts later.
3. Questions are encouraged.
4. This should at least give you a start to follow up on later.

The beauty (& sometimes otherwise) of R is that there are usually multiple ways of doing the same thing. Here I will show some modern ways you may later find that other ways are more suited to you.

We will mostly start looking at ggplot2 by Hadley Wickham. This is a newer alternative to base graphics. Base graphics are still useful for creating highly customised plots but we will not look at them here.

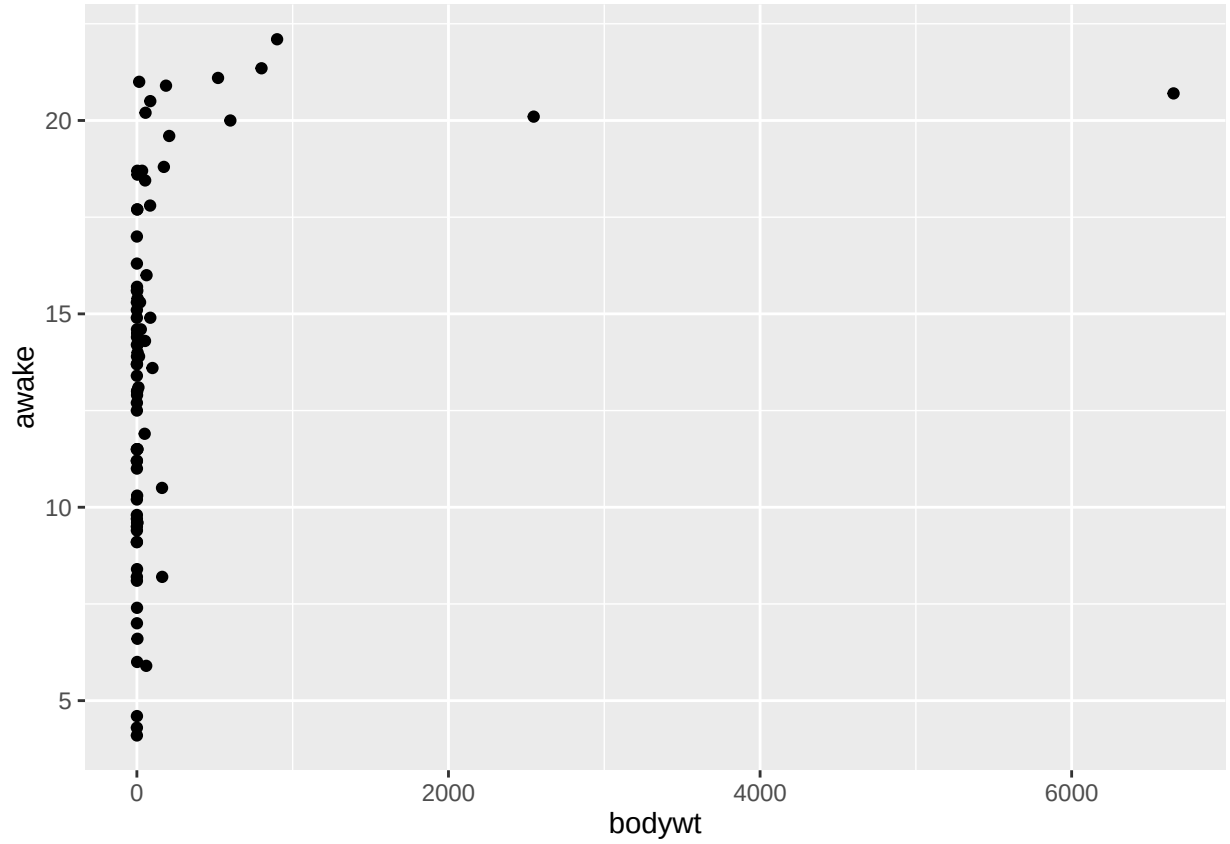
- A,B,C ... ggplot
- 1,2,3 ... maps
- At the end ... going further

A) point plots with ggplot2 first go

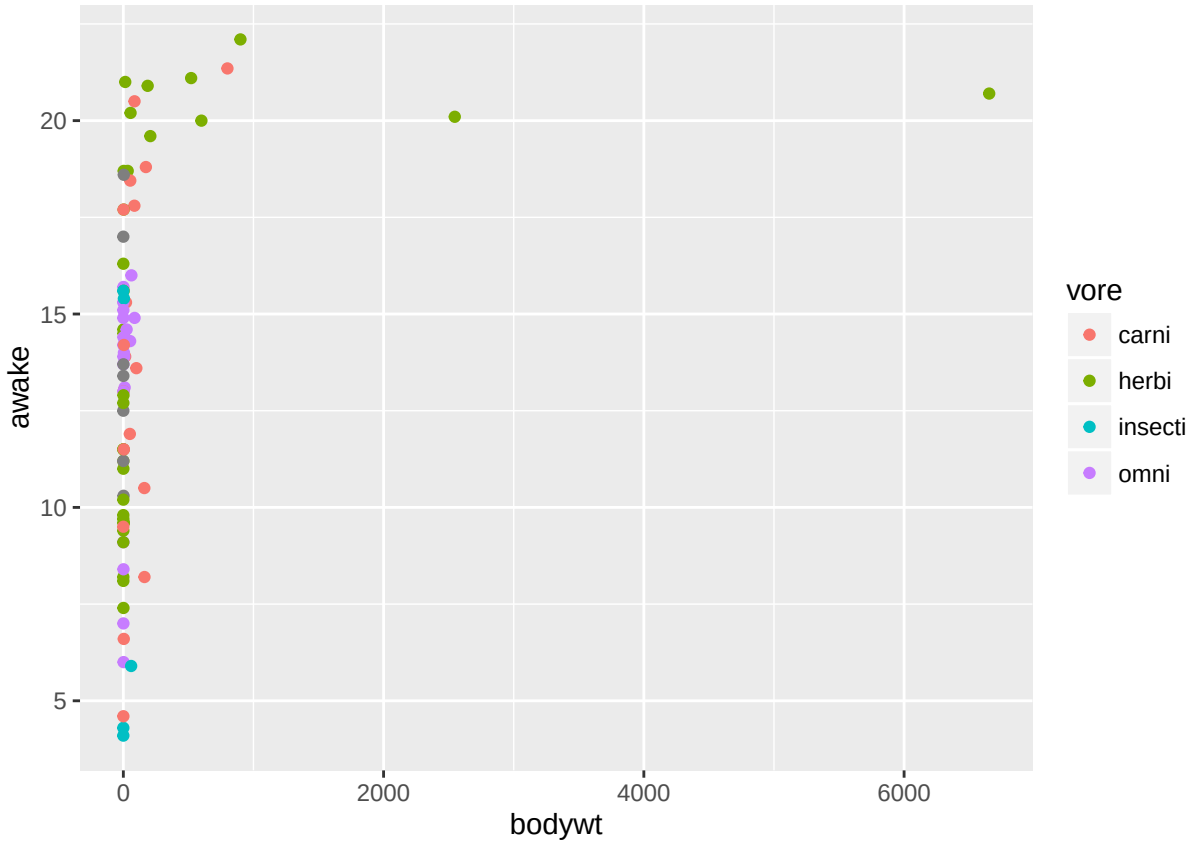
```
# load the package
library("ggplot2")

# load some data on mammal sleep patterns from the package
data(msleep)

# to show the 'str'ucture of the data
str(msleep)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    83 obs. of  11 variables:
##  $ name        : chr  "Cheetah" "Owl monkey" "Mountain beaver" "Greater short-tailed shrew" ...
##  $ genus       : chr  "Acinonyx" "Aotus" "Aplodontia" "Blarina" ...
##  $ vore        : chr  "carni" "omni" "herbi" "omni" ...
##  $ order       : chr  "Carnivora" "Primates" "Rodentia" "Soricomorpha" ...
##  $ conservation: chr  "lc" NA "nt" "lc" ...
##  $ sleep_total : num  12.1 17 14.4 14.9 4 14.4 8.7 7 10.1 3 ...
##  $ sleep_rem   : num  NA 1.8 2.4 2.3 0.7 2.2 1.4 NA 2.9 NA ...
##  $ sleep_cycle : num  NA NA NA 0.133 0.667 ...
##  $ awake       : num  11.9 7 9.6 9.1 20 9.6 15.3 17 13.9 21 ...
##  $ brainwt     : num  NA 0.0155 NA 0.00029 0.423 NA NA NA 0.07 0.0982 ...
##  $ bodywt      : num  50 0.48 1.35 0.019 600 ...
```

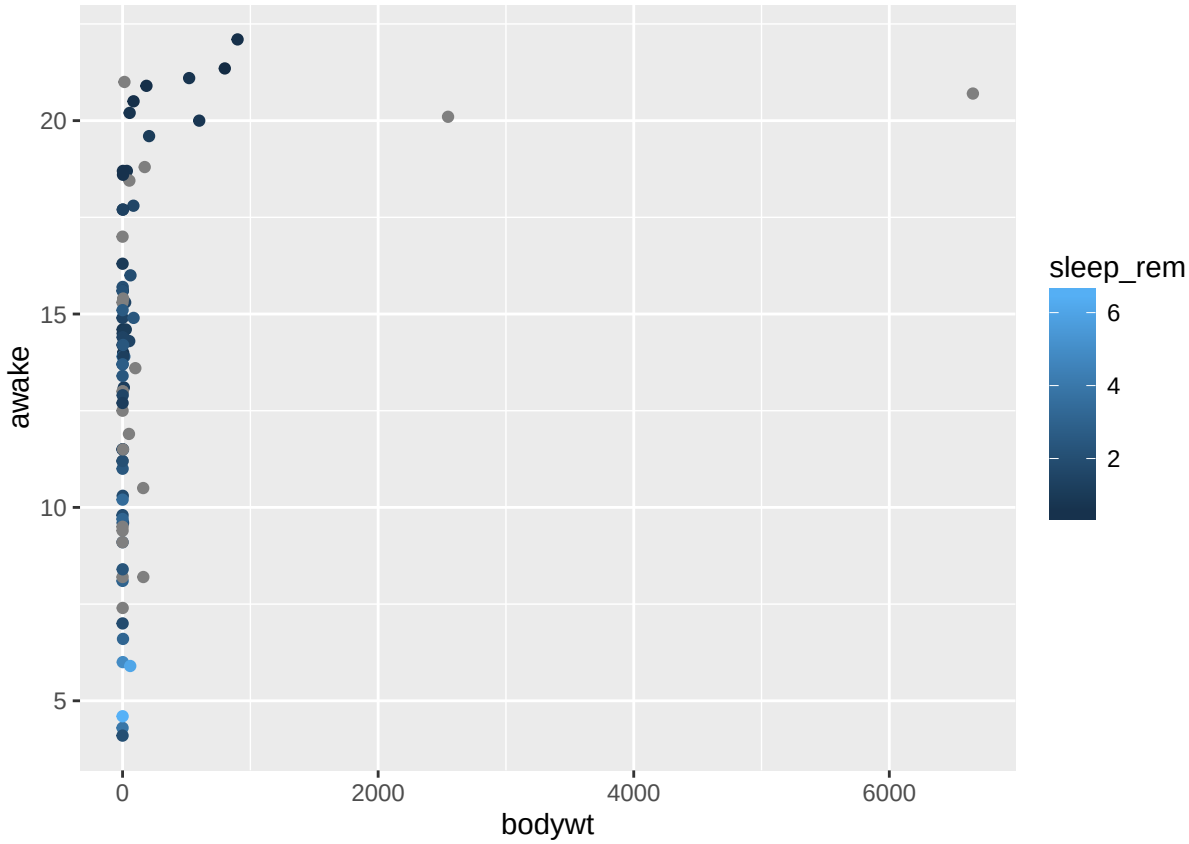
```
# first plot
ggplot(data = msleep, aes(x = bodywt, y = awake)) +
    geom_point()
```



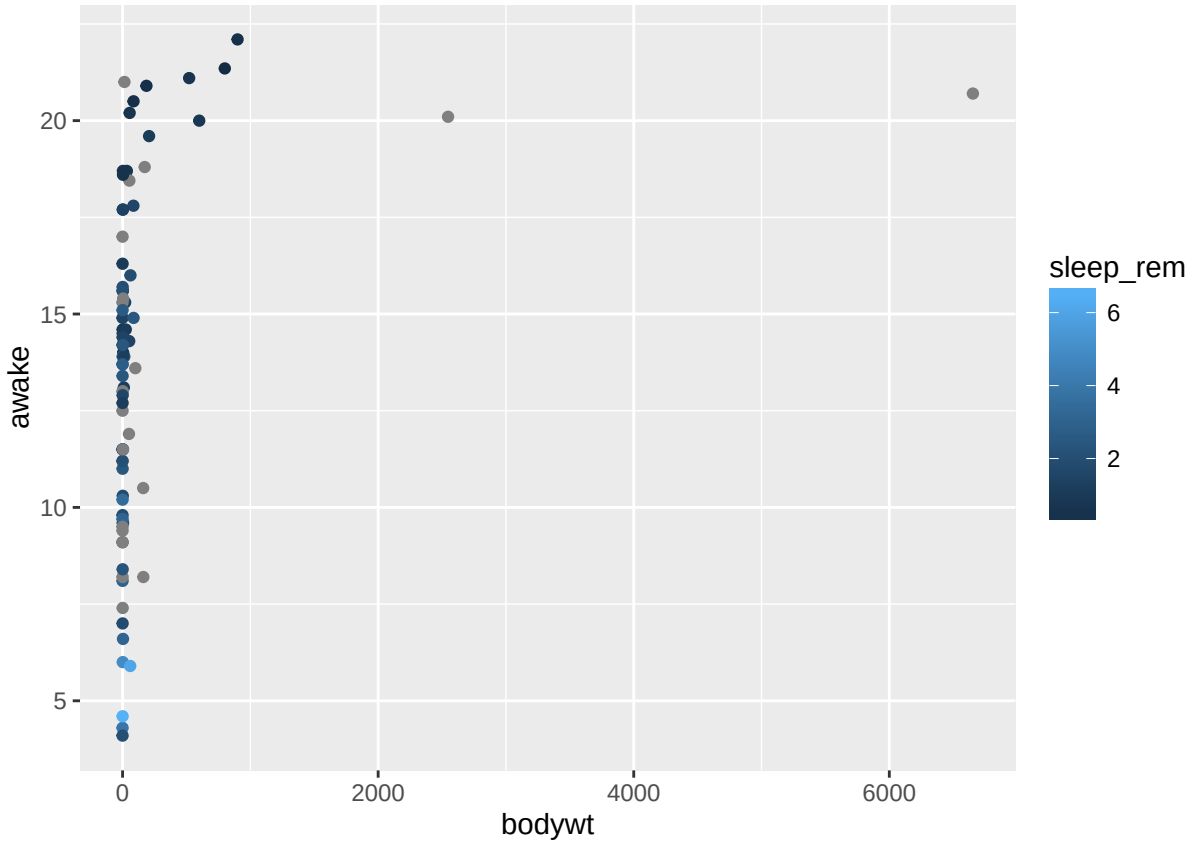

```
# aes (for aesthetic) sets up links between the data and aspects of the plot
# + adds layers to the plot, in this case points
# geom* stands for the geometric objects you can add to a plot
# bodywt and awake are columns within the data

# add an extra column to the aes
ggplot(data = msleep, aes(x = bodywt, y = awake, col = vore)) +
    geom_point()
```



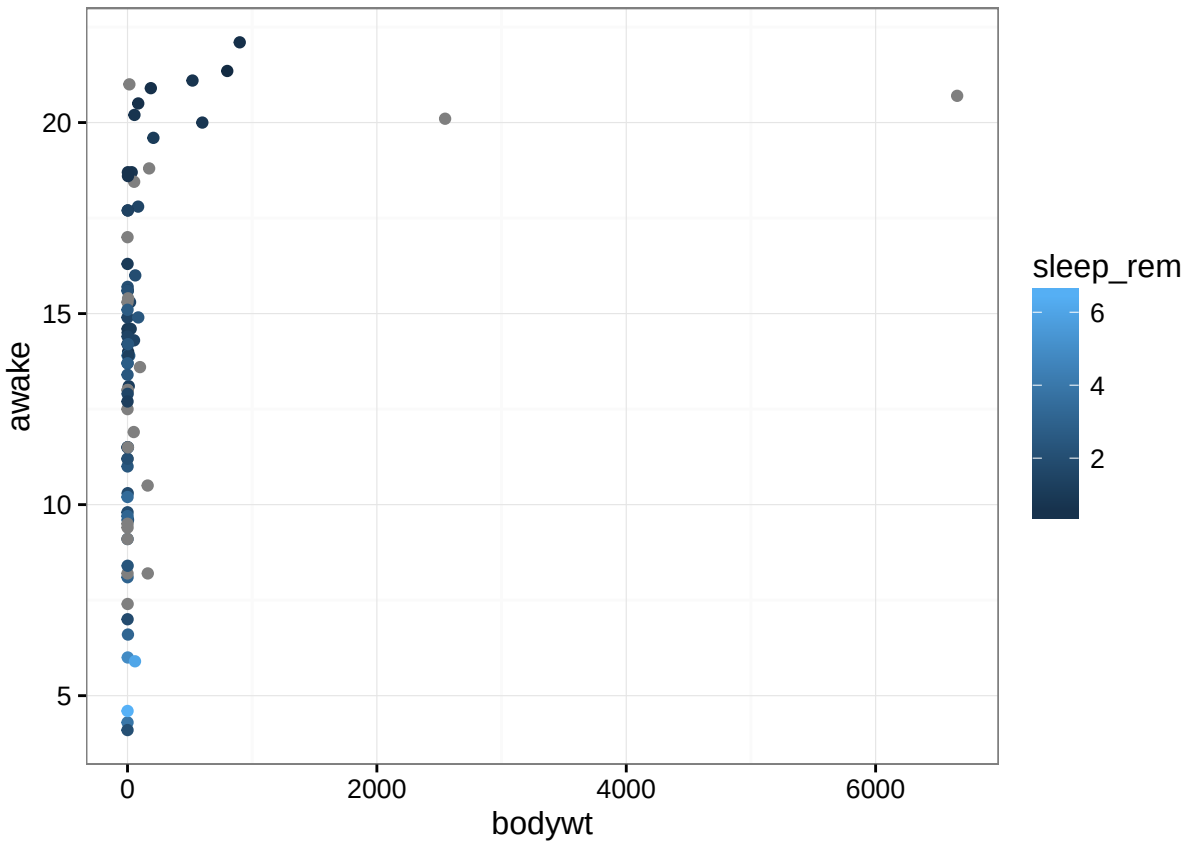

```
# if you change col= to a column that is numeric it changes the legend
ggplot(data = msleep, aes(x = bodywt, y = awake, col = sleep_rem)) +
    geom_point()
```




```
# you can store the plot as an object (in this case called p) which can make modifying it easier
p <- ggplot(data = msleep, aes(x = bodywt, y = awake, col = sleep_rem))
p <- p + geom_point()
# type the variable name to display the plot
p
```




```
# use themes to change the overall appearance of plots
# note that when typing this in RStudio it will show you the options once you've typed th...
p + theme_bw()
```



A1) Exercise

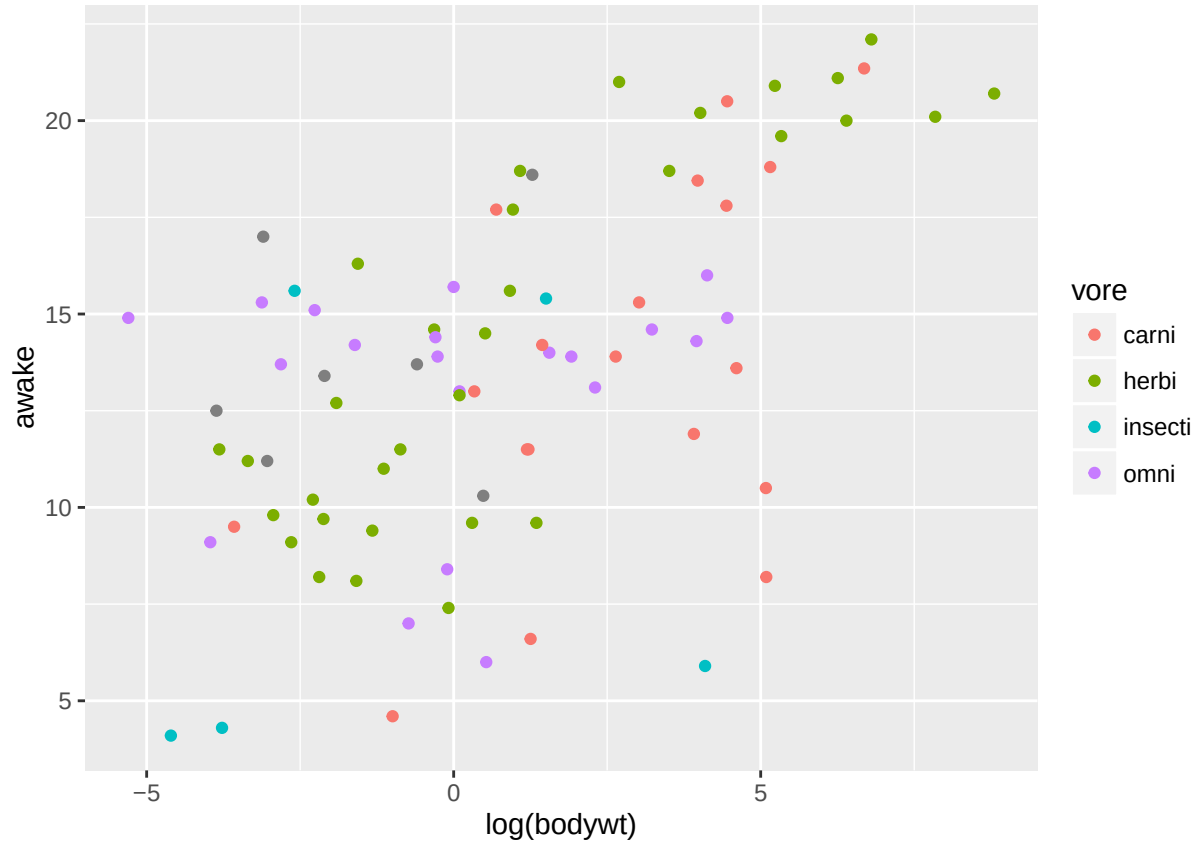
Have a quick look at the online ggplot2 documentation here : http://docs.ggplot2.org/current/ Particularly look at geom_point and scroll down to the examples.

From within R you can get similar help but without the helpful plots by : `?geom_point`

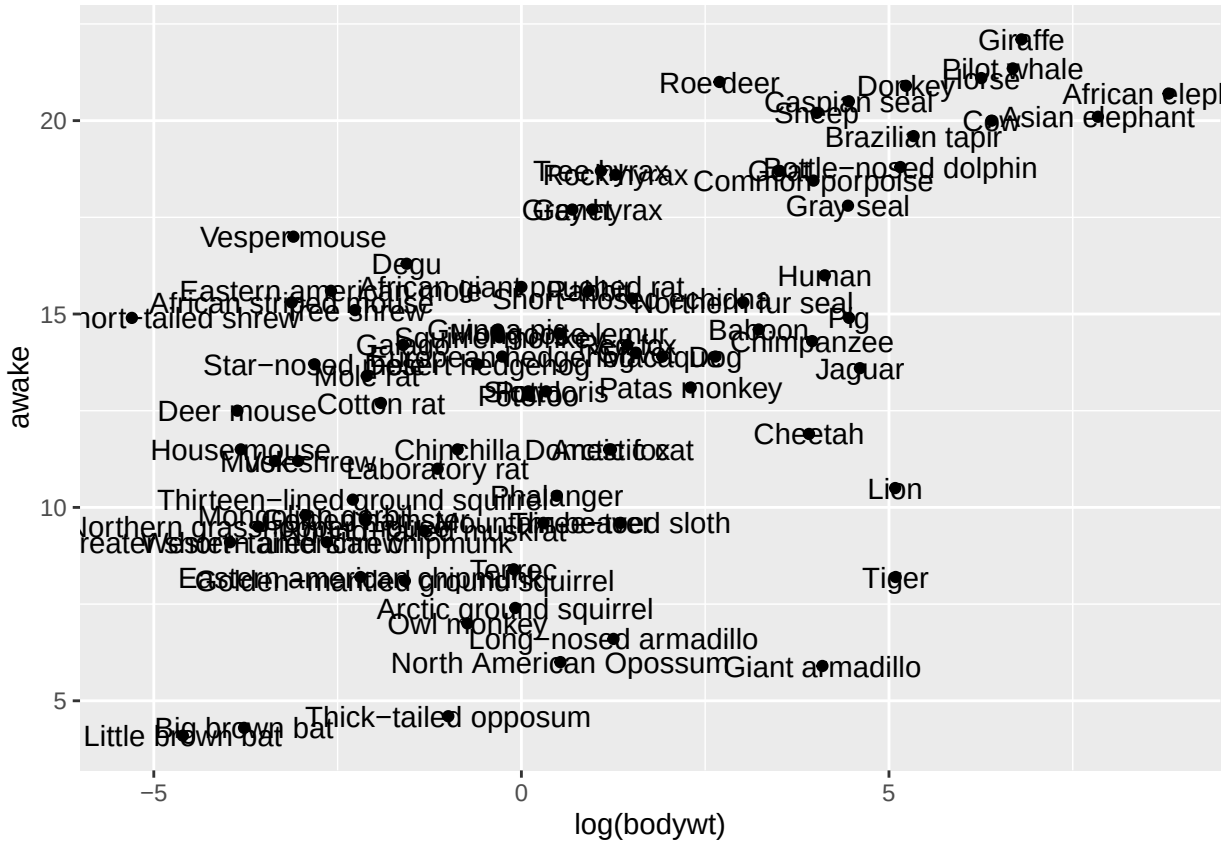
Try modifying a point plot with the msleep data.

B) point plots, modifying axes, label points, divide into subplots

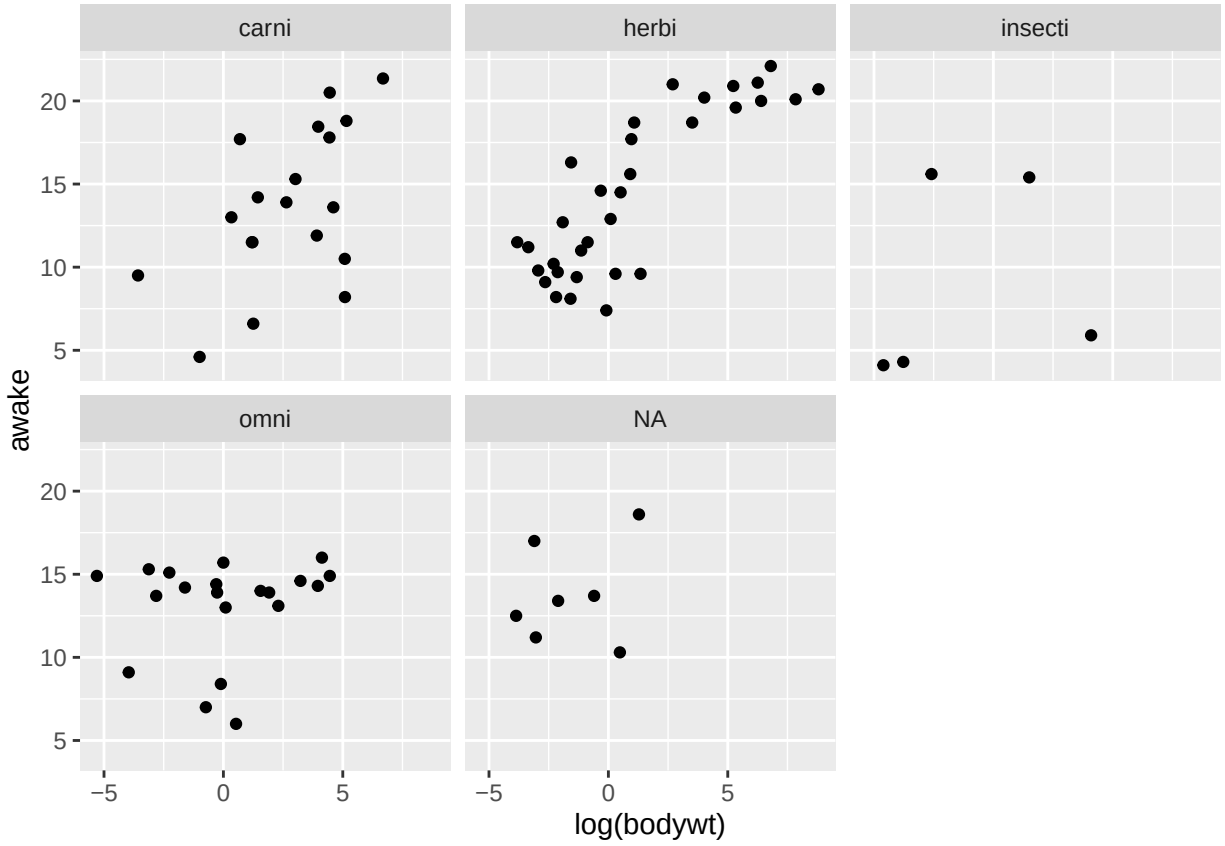
```
# we can log bodywt to space the points out more
p <- ggplot(data = msleep, aes(x = log(bodywt), y = awake, col = vore)) +
     geom_point()
p
```




```
# label the points for data exploration
p <- ggplot(data = msleep, aes(x = log(bodywt), y = awake, label = name)) +
     geom_point() +
     geom_text()
p
```




```
# divide into subplots using facet_wrap()
p <- ggplot(data = msleep, aes(x = log(bodywt), y = awake, label = name)) +
    geom_point() +
    facet_wrap(~vore)
p
```

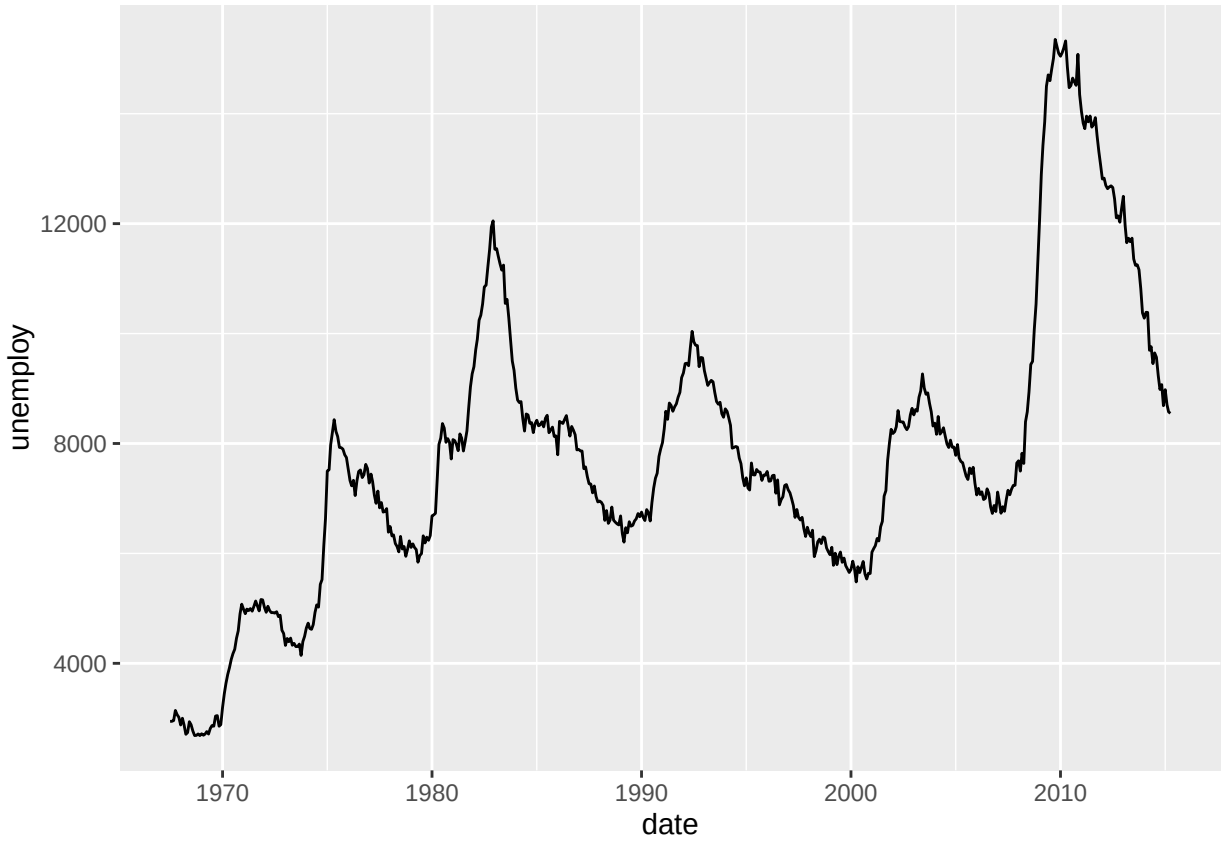


C) line plots with ggplot2

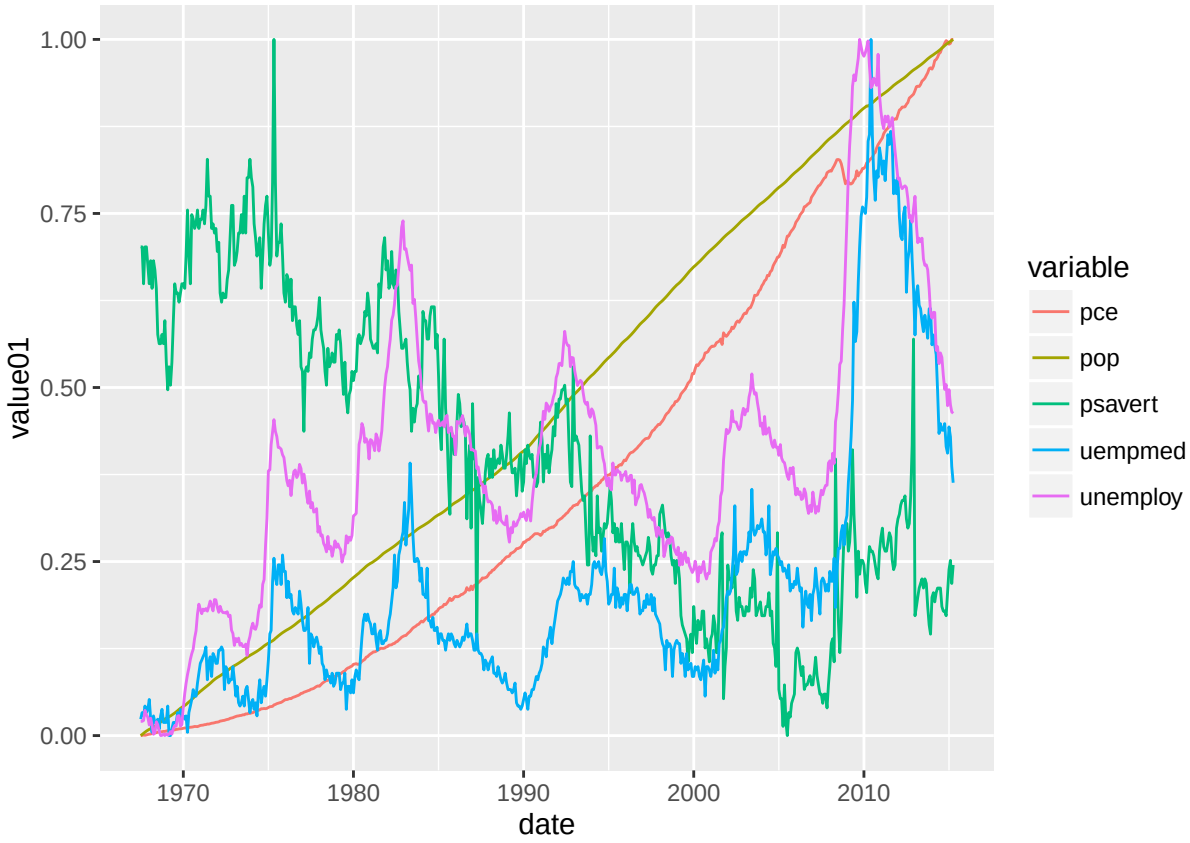
```
#load some time-series data from ggplot2
data(economics)

# to show the 'str'ucture of the data
str(economics)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    574 obs. of  6 variables:
##  $ date    : Date, format: "1967-07-01" "1967-08-01" ...
##  $ pce     : num  507 510 516 513 518 ...
##  $ pop     : int  198712 198911 199113 199311 199498 199657 199808 199920 200056 200208 ...
##  $ psavert : num  12.5 12.5 11.7 12.5 12.5 12.1 11.7 12.2 11.6 12.2 ...
##  $ uempmed : num  4.5 4.7 4.6 4.9 4.7 4.8 5.1 4.5 4.1 4.6 ...
##  $ unemploy: int  2944 2945 2958 3143 3066 3018 2878 3001 2877 2709 ...
```

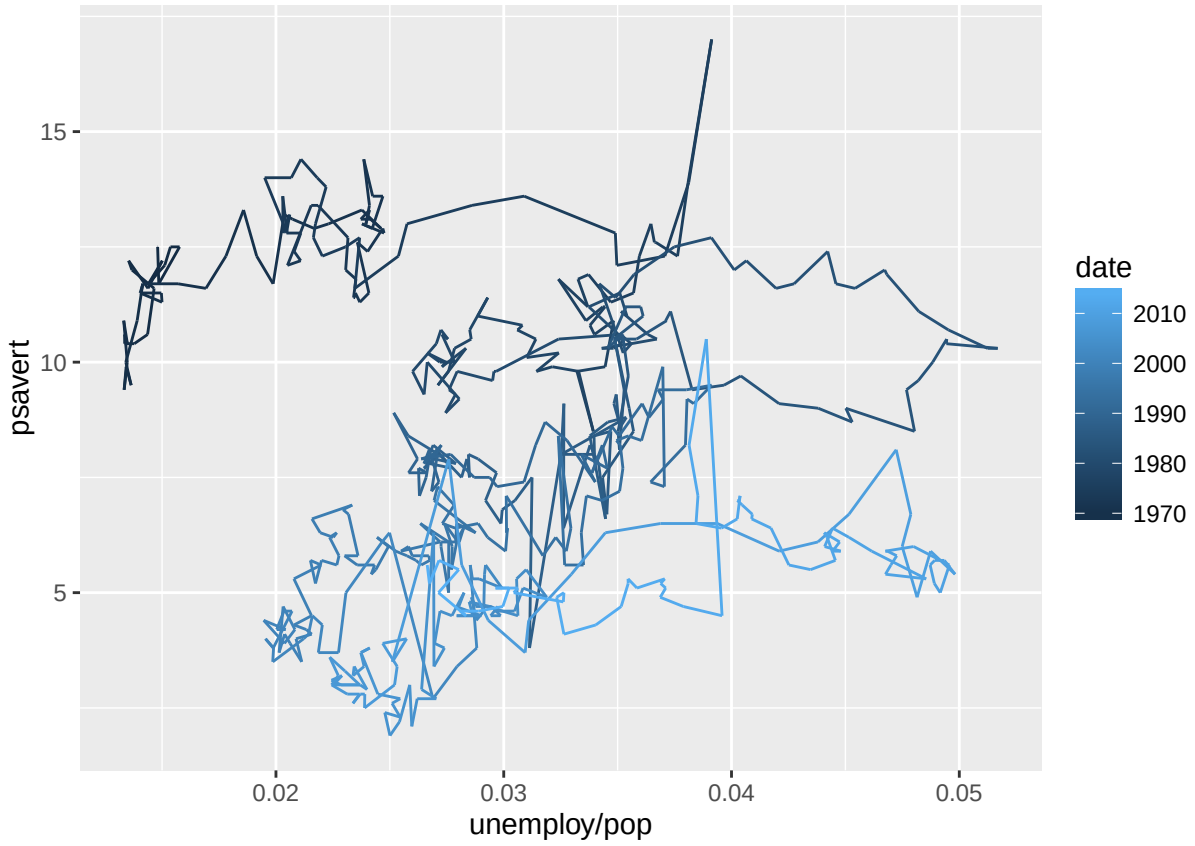
```
# geom_line()
ggplot(economics, aes(date, unemploy)) + geom_line()
```




```
#note that above aes(date, unemploy) assumes that these are to set x & y respectively
# this : aes(x=date, y=unemploy) would give an identical result

# another dataset economics_long is in 'long' format
# instead of having one column for each attribute (e.g. pop and unemploy)
# it has a column (variable) which has a repeated factor values for each date
# this allows us to create a plot with multiple lines

data(economics_long)

ggplot(economics_long, aes(date, value01, colour = variable)) +
  geom_line()
```




```
#note that for each variable the value01 column has been scaled between 0 & 1 to fit on same scale


# geom_path can be used to look at the relationship of 2 variables over time
# unemployment and savings rate
p <- ggplot(economics, aes(unemploy/pop, psavert))
p + geom_path(aes(colour = date))
```




```
# note how the unemployment rate is calculated by unemploy/pop within the aes call
# you can do other calculations between columns in there
```

C1) Exercise

Try using facet_wrap with the economics_long line plot to get subplots for each variable.

...

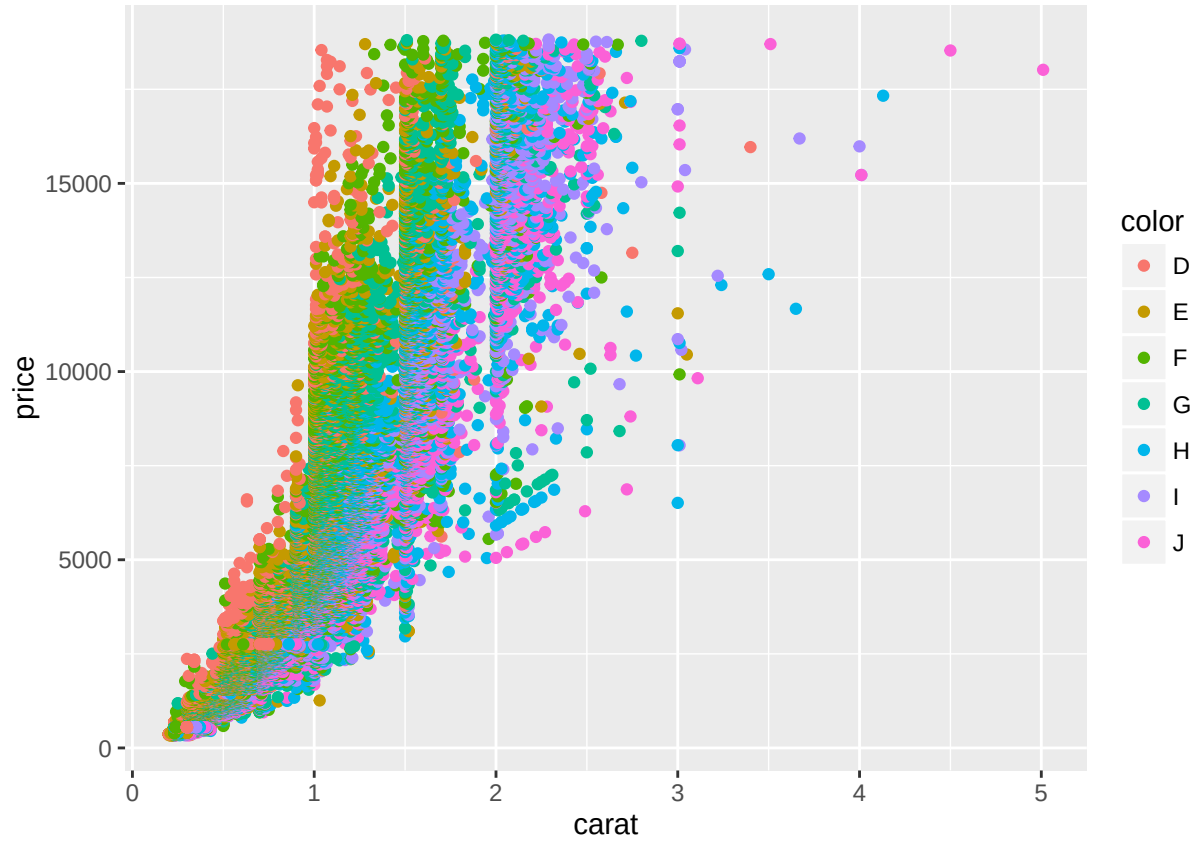
D) bigger (but still small!) data, histograms etc.

```
# the diamonds dataset has > 50K diamonds (rows) in it

data(diamonds)
str(diamonds)
```

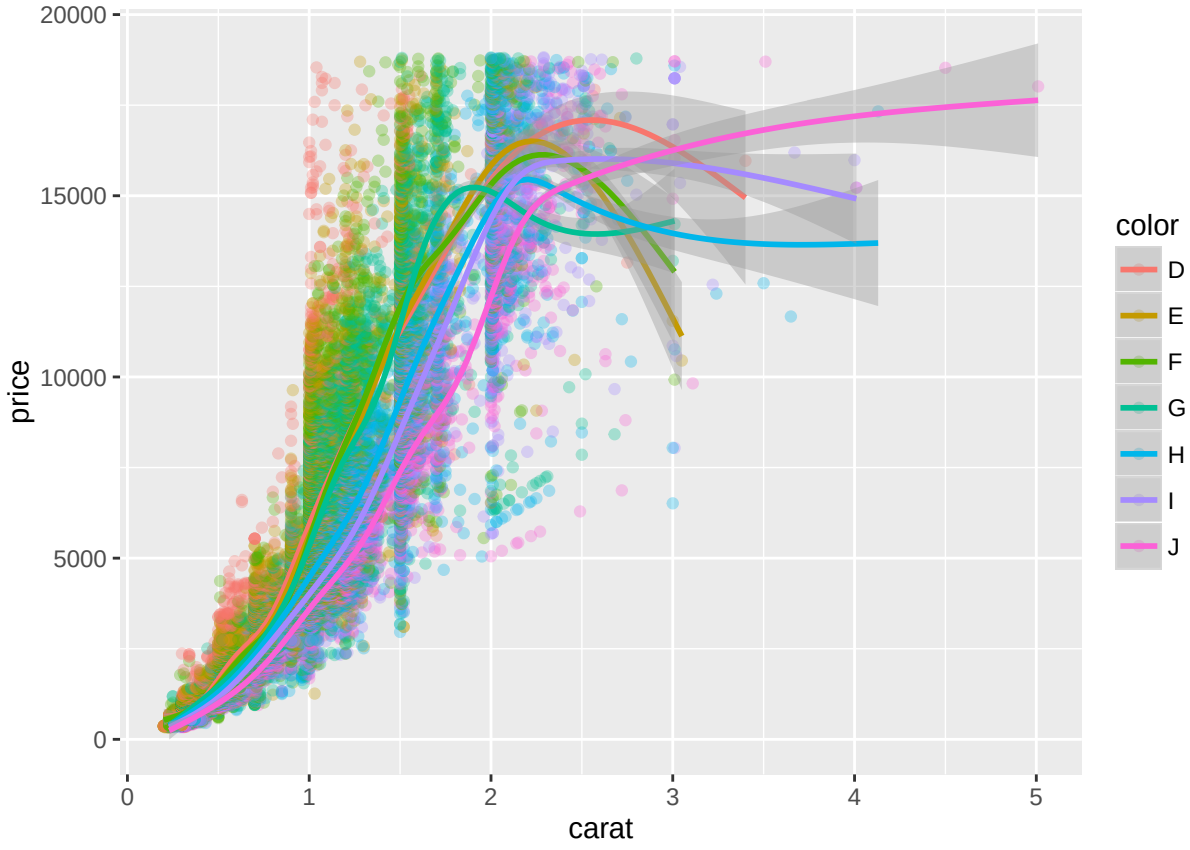
```
## Classes 'tbl_df', 'tbl' and 'data.frame':    53940 obs. of  10 variables:
##  $ carat  : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
##  $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...
##  $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...
##  $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...
##  $ depth  : num  61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
##  $ table  : num  55 61 65 58 58 57 57 55 61 61 ...
```

```
##  $ price  : int  326 326 327 334 335 336 336 337 337 338 ...
##  $ x      : num  3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
##  $ y      : num  3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
##  $ z      : num  2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
```

```
# a scatter plot is not very informative
p <- ggplot(diamonds, aes(x = carat, y = price, col = color))
p + geom_point()
```

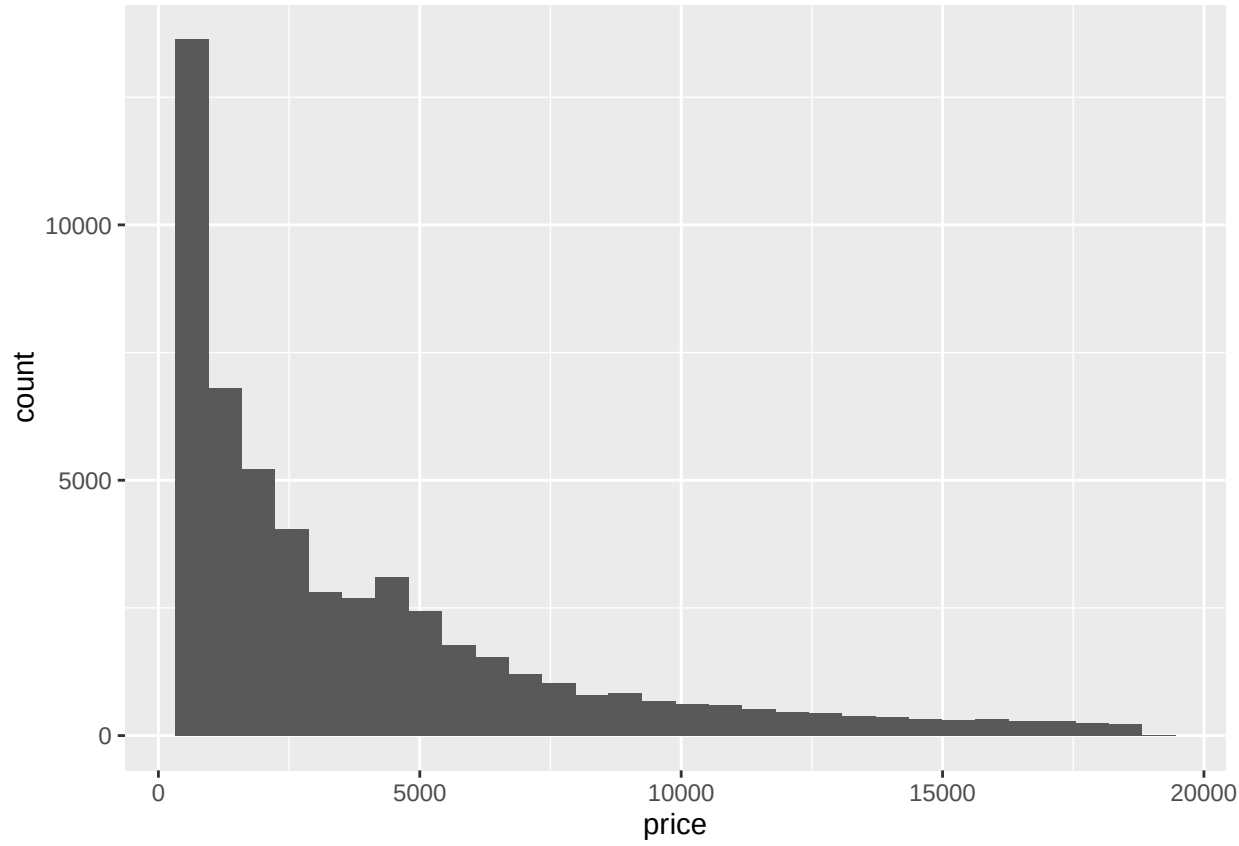



```
# setting the alpha (transparency) level to < 1 can improve things but not much
p <- ggplot(diamonds, aes(x = carat, y = price, col = color))
p <- p + geom_point(alpha=0.3)

# geom_smooth adds a smoothed conditional mean which can help reveal patterns
p + geom_smooth()
```




```
# geom_histogram can be used to look at distributions
p <- ggplot(data=diamonds) + geom_histogram(aes(x=price))
p
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
# most of the diamonds are cheap ! (relatively)
```

D1) Exercise

Use any of your ggplot skills to find out something interesting about diamonds . . .

. . .

1 maps

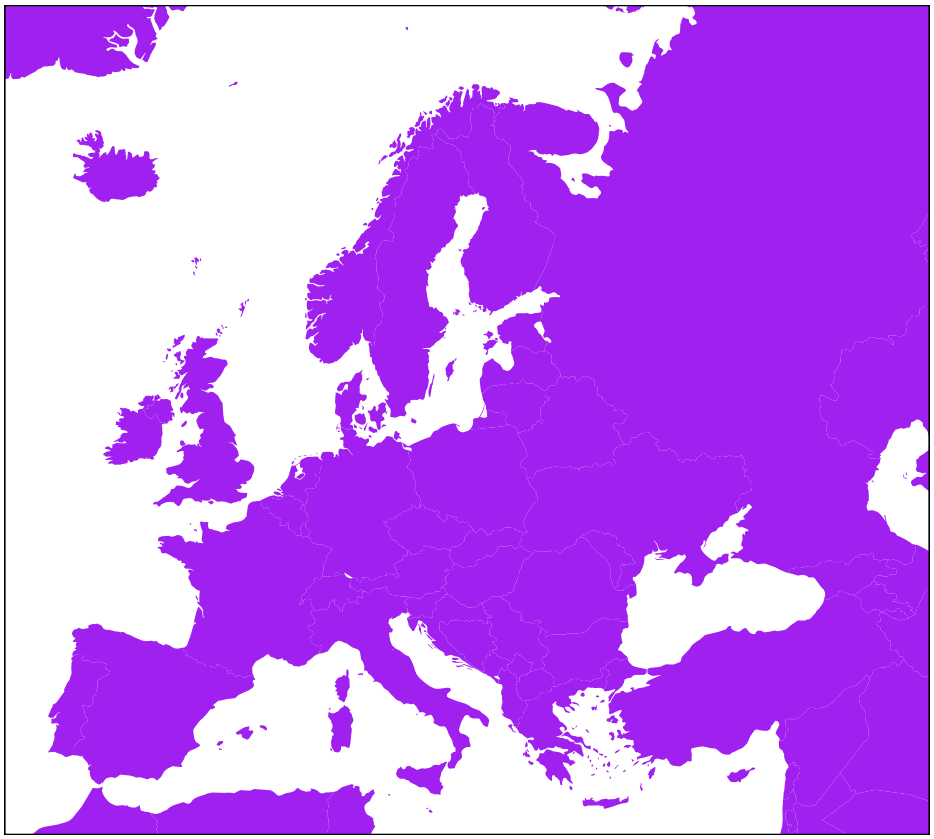
Install and load the tmap package

```
install.packages("tmap")
```

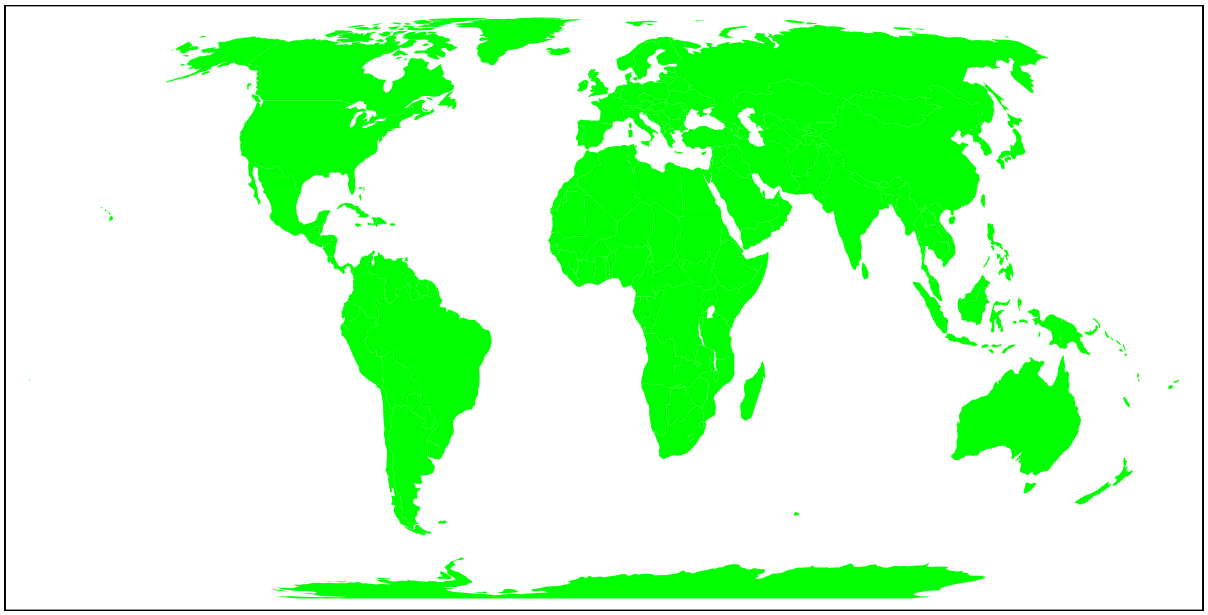
```
library("tmap")
```

1.1 Getting started with polygon maps

```
# load data from tmap
data(Europe)
# set shapes and fill them
tm_shape(Europe) +
  tm_fill("purple")
```



```
# load data from tmap
data(World)
# set shapes and fill them
tm_shape(World) +
  tm_fill("green")
```



```
# check what data are associated with a map using 'str(*@data)', str stands for structure
str(World@data)
```

```
## 'data.frame':    177 obs. of  15 variables:
##  $ iso_a3      : Factor w/ 235 levels "ABW","AFG","AGO",..: 2 3 6 8 9 10 12 13 15 16 ...
##  $ name        : Factor w/ 241 levels "Afghanistan",..: 1 7 3 228 11 12 9 76 15 16 ...
##  $ sovereignt  : Factor w/ 200 levels "Afghanistan",..: 1 5 2 187 8 9 6 60 10 11 ...
##  $ continent   : Factor w/ 8 levels "Africa","Antarctica",..: 3 1 4 3 8 3 2 7 6 4 ...
##  $ subregion   : Factor w/ 24 levels "Antarctica","Australia and New Zealand",..: 20 11 21 23 18 23 
##  $ area        : num  652860 1246700 27400 83600 2736690 ...
##  $ pop_est     : num  28400000 12799293 3639453 4798491 40913584 ...
##  $ pop_est_dens: num  43.5 10.3 132.8 57.4 15 ...
##  $ gdp_md_est  : num  22270 110300 21810 184300 573900 ...
##  $ gdp_cap_est : num  784 8618 5993 38408 14027 ...
##  $ economy     : Factor w/ 7 levels "1. Developed region: G7",..: 7 7 6 6 5 6 6 6 2 2 ...
##  $ income_grp  : Ord.factor w/ 5 levels "1. High income: OECD"<..: 5 3 4 2 3 4 2 2 1 1 ...
##  $ life_exp    : num  48.7 51.1 76.9 76.5 75.9 74.2 NA NA 81.9 80.9 ...
##  $ well_being  : num  4.76 4.21 5.27 7.2 6.44 ...
##  $ HPI         : num  36.8 33.2 54.1 31.8 54.1 ...
```

```
str(Europe@data)
```

```
## 'data.frame':    68 obs. of  16 variables:
##  $ iso_a3      : Factor w/ 67 levels "ALA","ALB","AND",..: 2 1 3 4 5 6 7 8 9 10 ...
##  $ name        : Factor w/ 68 levels "Aland","Albania",..: 2 1 4 5 6 7 9 11 10 8 ...
```

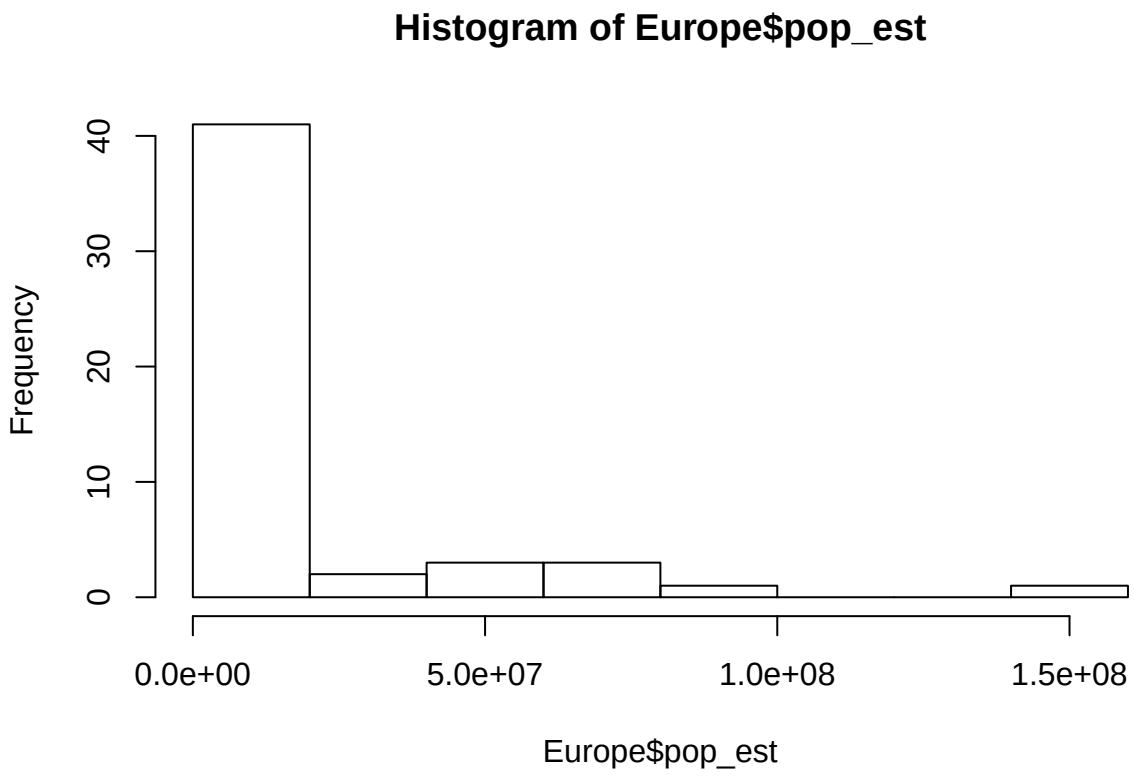
```
##  $ sovereignt  : Factor w/ 61 levels "Albania","Algeria",..: 1 16 3 4 5 6 8 10 9 7 ...
##  $ continent   : Factor w/ 4 levels "Africa","Asia",..: 3 3 3 2 3 2 3 3 3 3 ...
##  $ part        : Factor w/ 4 levels "Northern Europe",..: 3 1 3 NA 2 NA 2 4 3 4 ...
##  $ EU_Schengen : Factor w/ 4 levels "EU Schengen",..: NA NA NA NA 1 NA 1 2 NA NA ...
##  $ area        : num  27400 674 470 NA 82409 ...
##  $ pop_est     : num  3639453 27153 83888 NA 8210281 ...
##  $ pop_est_dens: num  132.8 40.3 178.5 NA 99.6 ...
##  $ gdp_md_est  : num  21810 1563 3660 NA 329500 ...
##  $ gdp_cap_est : num  5993 57563 43630 NA 40133 ...
##  $ economy     : Factor w/ 7 levels "1. Developed region: G7",..: 6 2 2 NA 2 NA 2 2 6 6 ...
##  $ income_grp  : Ord.factor w/ 5 levels "1. High income: OECD"<..: 4 1 2 NA 1 NA 1 3 3 3 ...
##  $ life_exp    : num  76.9 NA NA NA 80.9 NA 80 73.4 75.7 70.3 ...
##  $ well_being  : num  5.27 NA NA NA 7.35 ...
##  $ HPI         : num  54.1 NA NA NA 47.1 ...
```

```
# note that str(Europe) shows all the geometry information too and is not very useful here

# check the contents of Factor variables using 'levels'
levels(Europe$part)
```

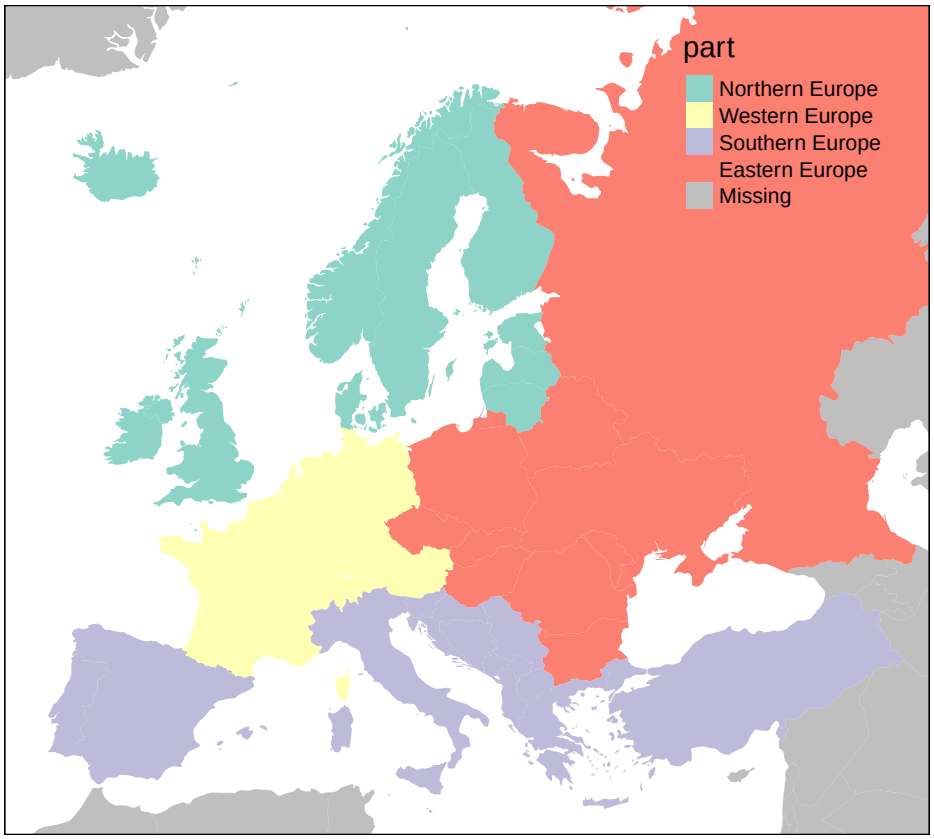
```
## [1] "Northern Europe" "Western Europe"  "Southern Europe" "Eastern Europe"
```

```
# levels(Europe@data$part) gives the same output, you don't always need the @data to get at the variabl

# plot a subset of the map using '[which(*),]'
tm_shape(Europe[which(Europe$part=='Northern Europe'),]) +
  tm_fill("red")
```

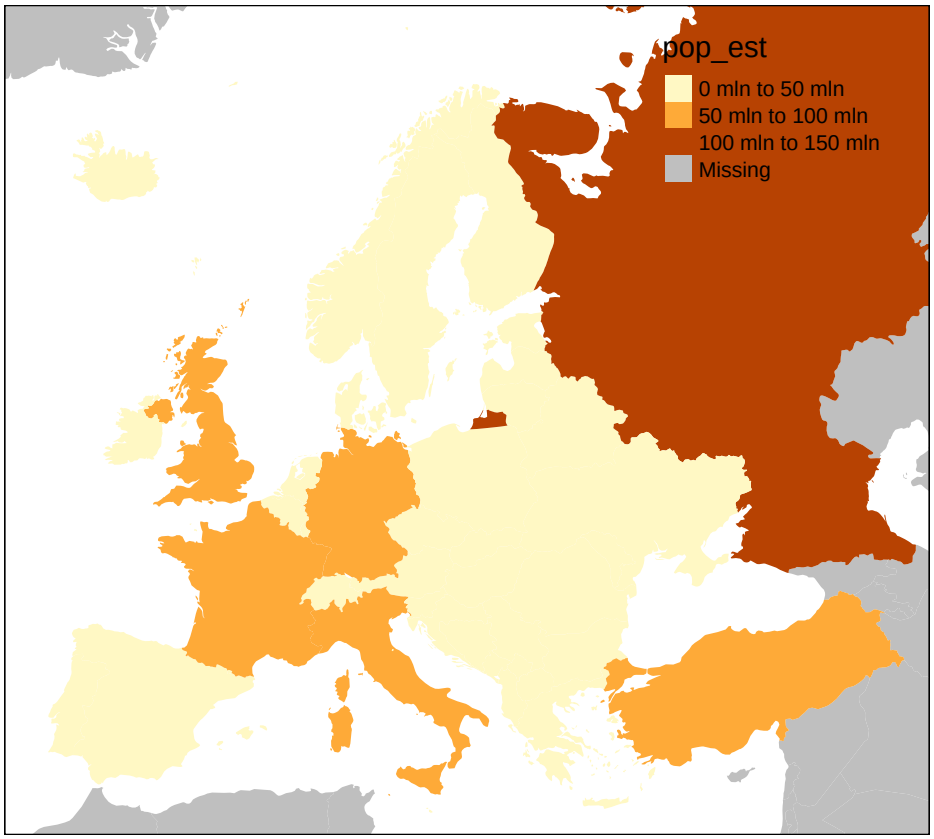
```r
# check the contents of numeric variables using 'hist'
hist(Europe$pop_est)
```



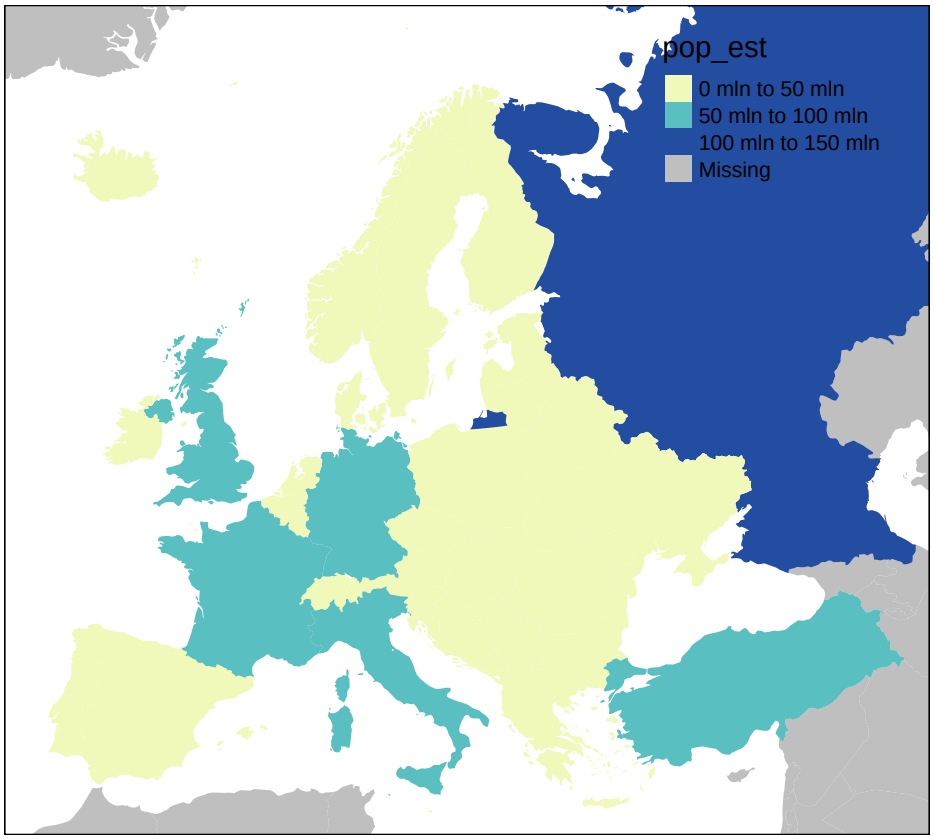

```
# fill polygons on a map based on data variables
# factor
tm_shape(Europe) +
  tm_fill("part")
```



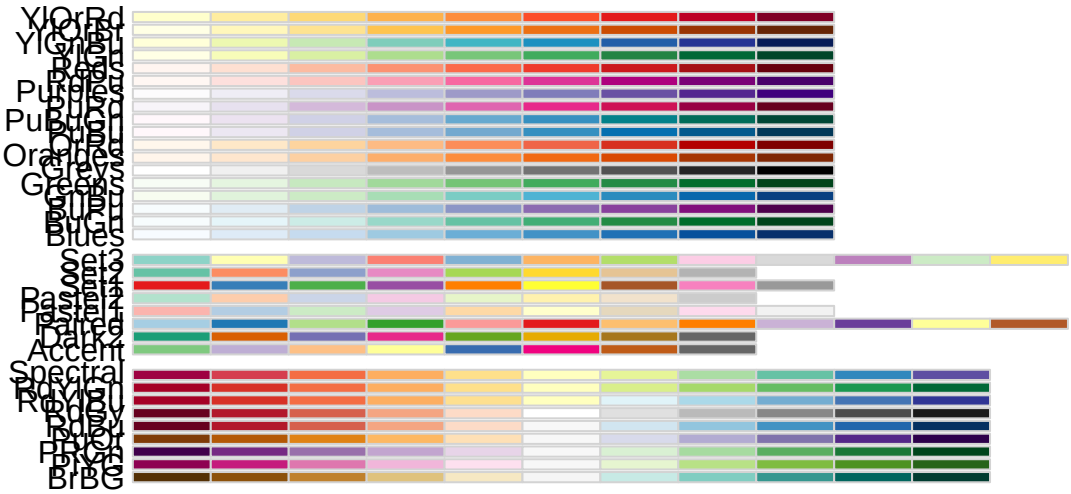

```
# numeric
tm_shape(Europe) +
  tm_fill("pop_est")
```



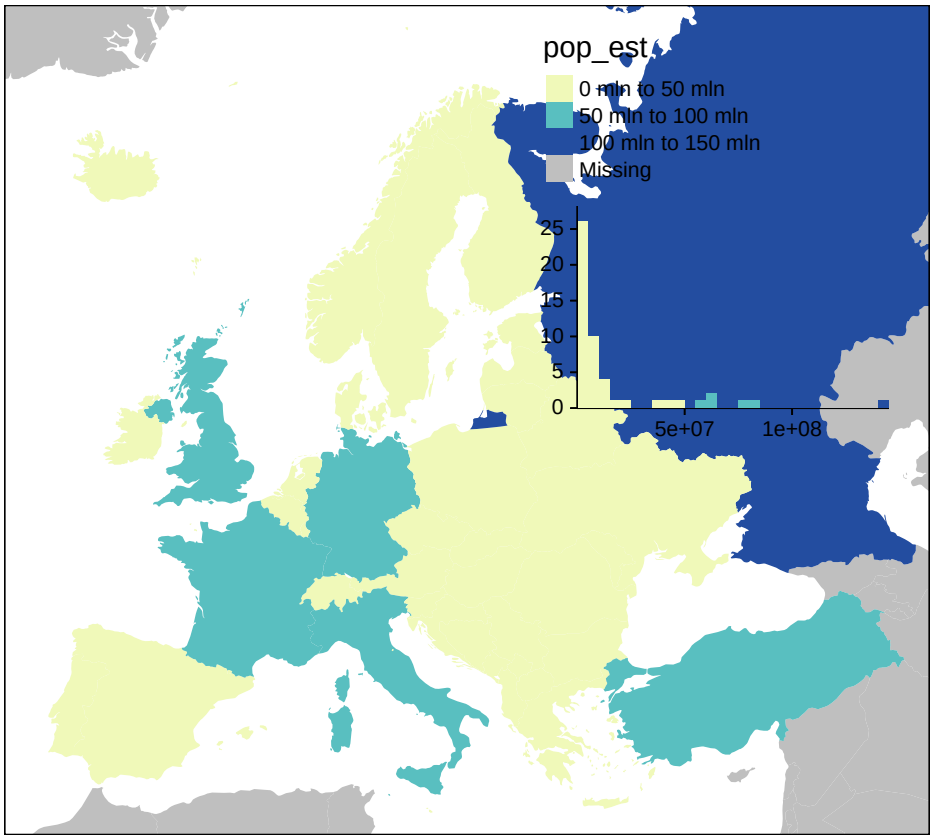

```
# the previous commands used default colours, you can modify these with 'palette'
tm_shape(Europe) +
  tm_fill("pop_est", palette = "YlGnBu")
```




```
# to see available palettes
RColorBrewer::display.brewer.all()
```



```
# to add a histogram to the legend
tm_shape(Europe) +
  tm_fill("pop_est", palette = "YlGnBu", legend.hist = TRUE)
```




```
# maps are stored as a SpatialPolygonsDataFrame from the package sp
# you can find this out by typing
class(Europe)
```

```
## [1] "SpatialPolygonsDataFrame"
## attr(,"package")
## [1] "sp"
```

1.2 Exercise : can you plot different areas, different variables, different colours ...

```
# to see help use '?'
?tm_fill

# syntax is like ggplot2 (e.g. + to add layers)

# start with one of the code examples above and change something ...
```

2.1 Getting started with point maps

```
# load some tmap point data for metropolitan areas
data(metro)
```

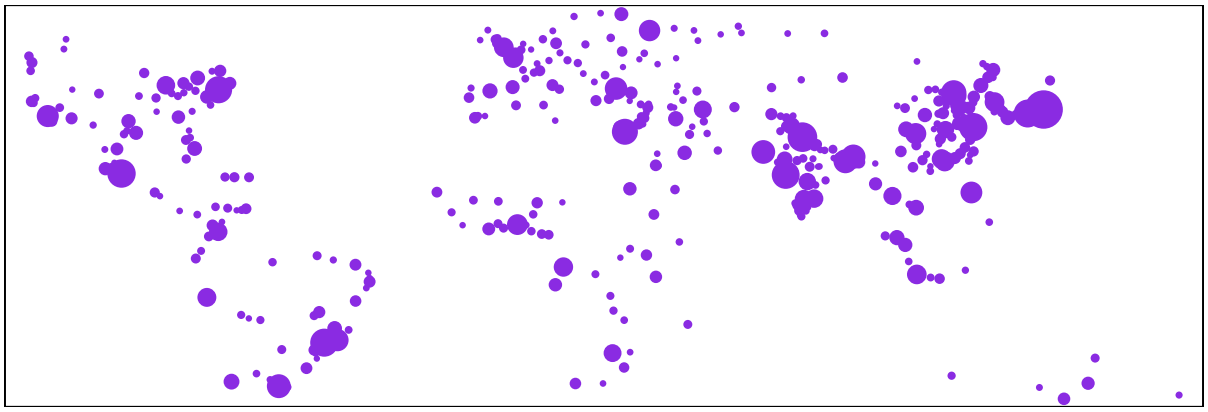
```r
# ?metro gives information about the data
# this is also an sp object with associated variables
class(metro)
```

```
## [1] "SpatialPointsDataFrame"
## attr(,"package")
## [1] "sp"
```

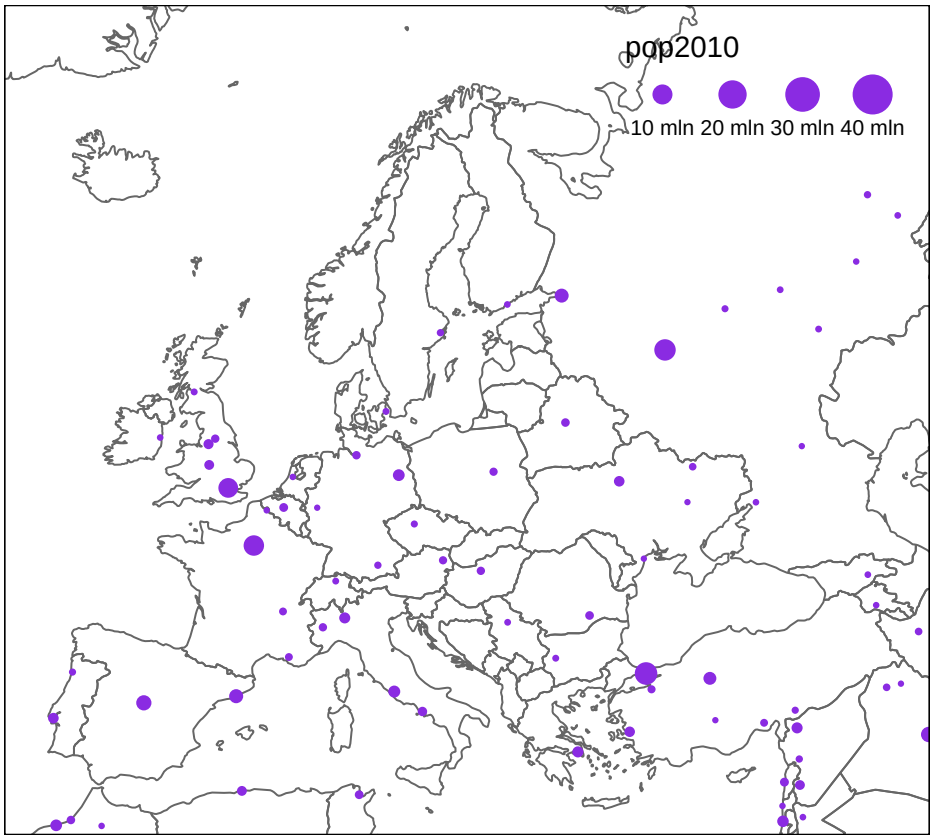
```r
str(metro@data)
```

```
## 'data.frame':    436 obs. of  12 variables:
##  $ name     : chr  "Kabul" "Algiers" "Luanda" "Buenos Aires" ...
##  $ name_long: chr  "Kabul" "El Djazair  (Algiers)" "Luanda" "Buenos Aires" ...
##  $ iso_a3   : chr  "AFG" "DZA" "AGO" "ARG" ...
##  $ pop1950  : num  170784 516450 138413 5097612 429249 ...
##  $ pop1960  : num  285352 871636 219427 6597634 605309 ...
##  $ pop1970  : num  471891 1281127 459225 8104621 809794 ...
##  $ pop1980  : num  977824 1621442 771349 9422362 1009521 ...
##  $ pop1990  : num  1549320 1797068 1390240 10513284 1200168 ...
##  $ pop2000  : num  2401109 2140577 2591388 12406780 1347561 ...
##  $ pop2010  : num  3722320 2432023 4508434 14245871 1459268 ...
##  $ pop2020  : num  5721697 2835218 6836849 15894307 1562509 ...
##  $ pop2030  : num  8279607 3404575 10428756 16956491 1718192 ...
```

```r
# plot points as bubbles
tm_shape(metro) +
    tm_bubbles("pop2010", legend.size.show = FALSE)
```



```
# plot points on top of a European map
tm_shape(Europe) +
  tm_borders() +
tm_shape(metro) +
  tm_bubbles("pop2010")
```



2.2 Exercise - can you cahnge one of the point maps ?

3.1 Getting started with raster maps

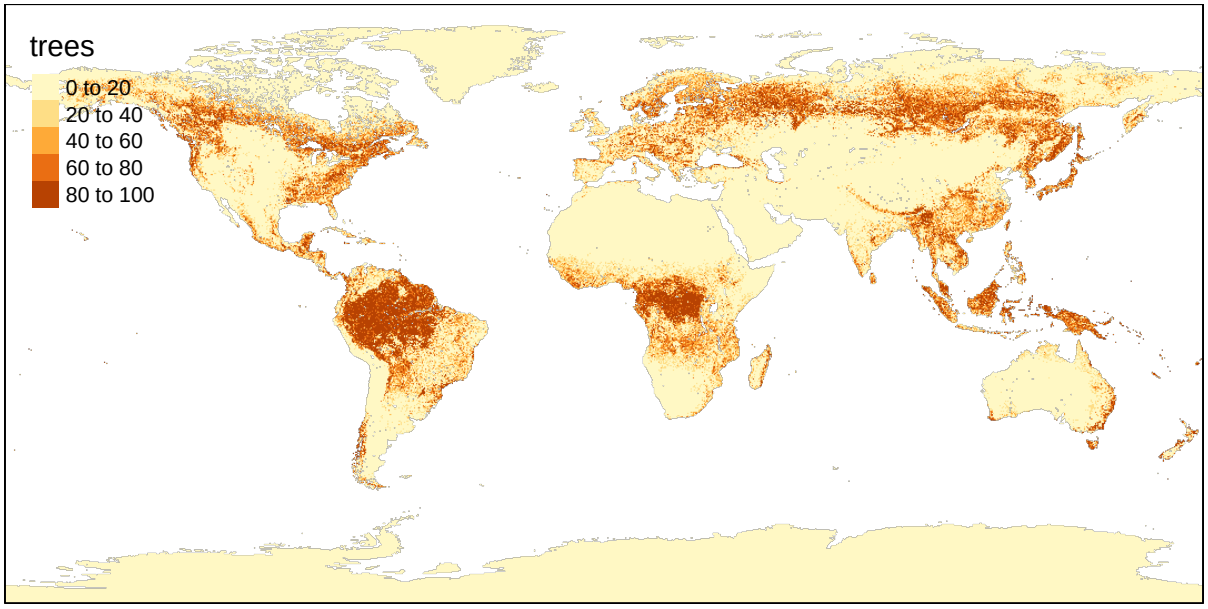
```
# load some data from tmap
data(land)

class(land)
```

```
## [1] "SpatialGridDataFrame"
## attr(,"package")
## [1] "sp"
```

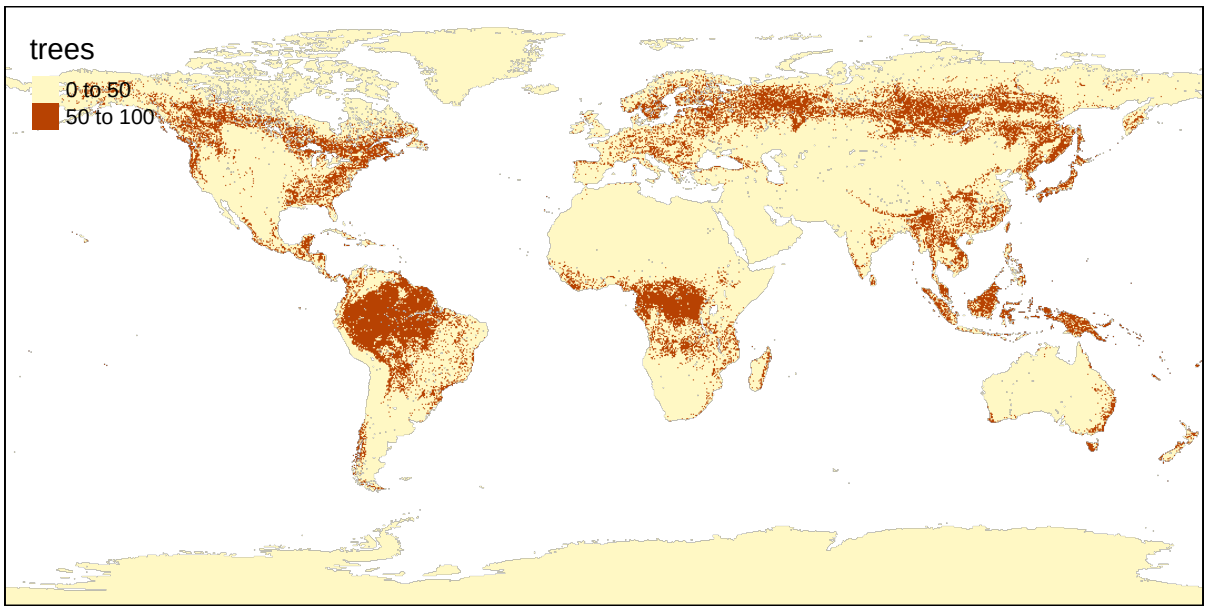
```
str(land@data)
```

```
## 'data.frame':    583200 obs. of  4 variables:
##  $ cover    : Factor w/ 20 levels "Broadleaf Evergreen Forest",..: 20 20 20 20 20 20 20 20 20 20 ...
##  $ cover_cls: Factor w/ 8 levels "Forest","Other natural vegetation",..: 8 8 8 8 8 8 8 8 8 8 ...
##  $ trees    : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ elevation: int  NA NA NA NA NA NA NA NA NA NA ...
```

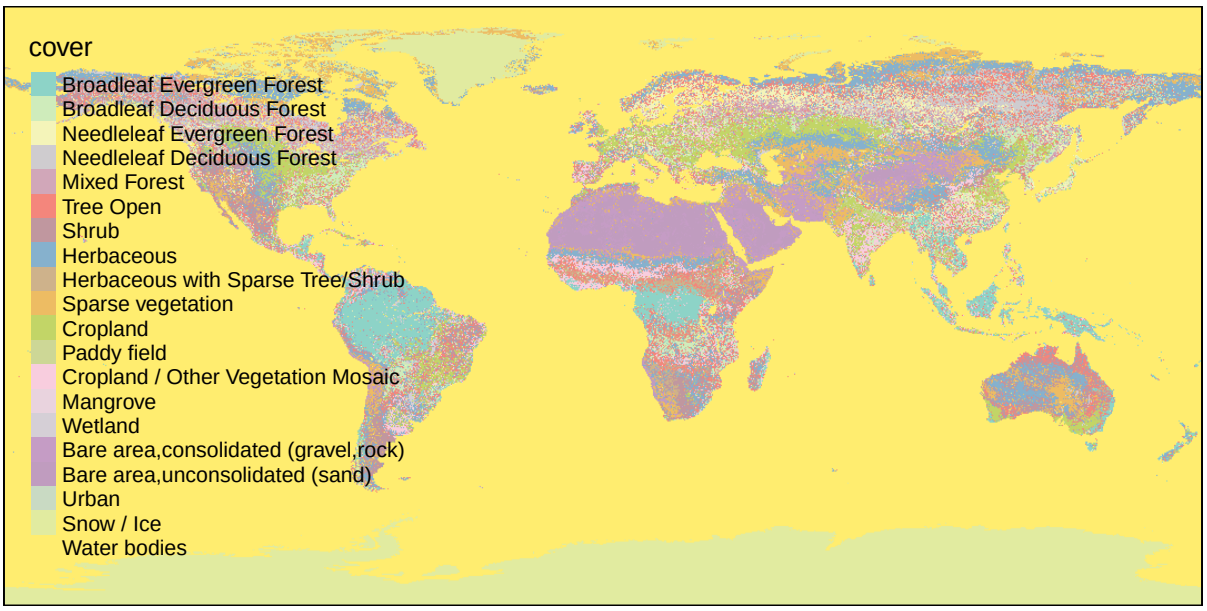
```
tm_shape(land) +
    tm_raster("trees", breaks=seq(0, 100, by=20) )
```



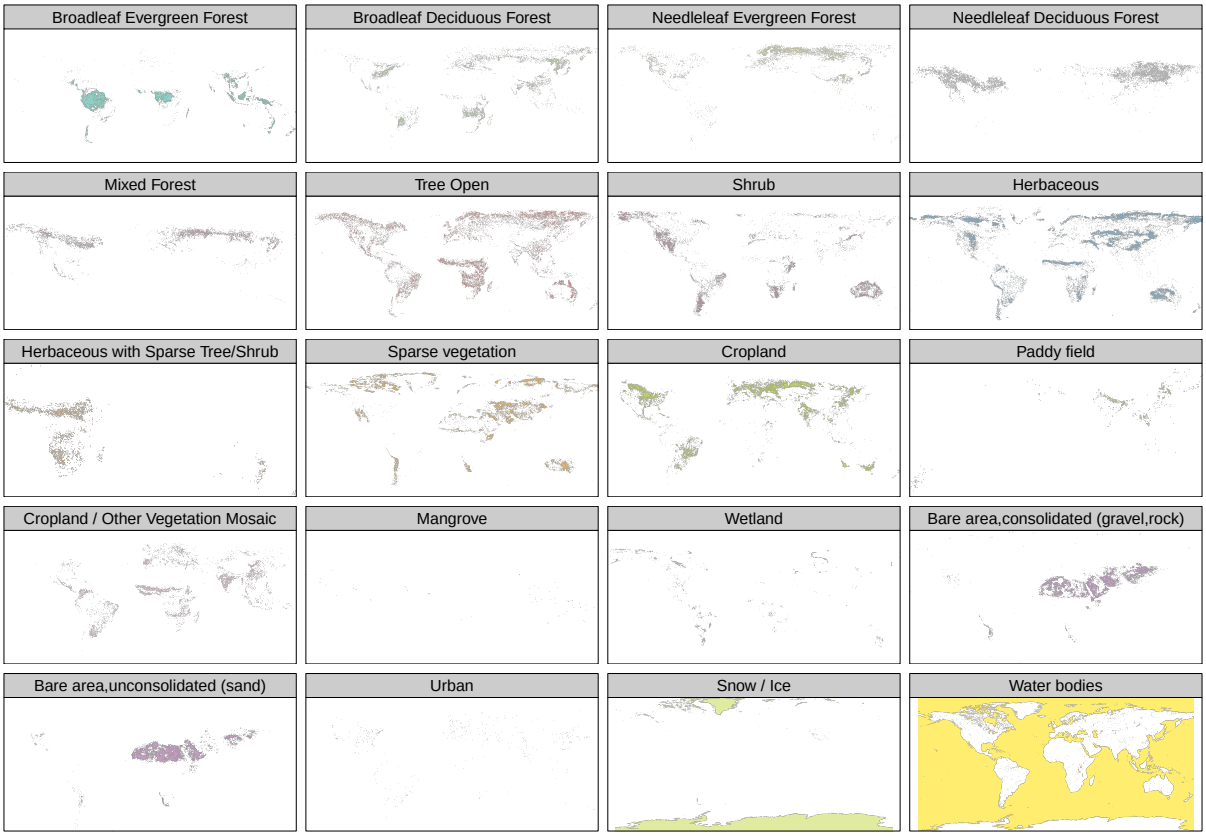

```
tm_shape(land) +
    tm_raster("trees", breaks=seq(0, 100, by=50) )
```



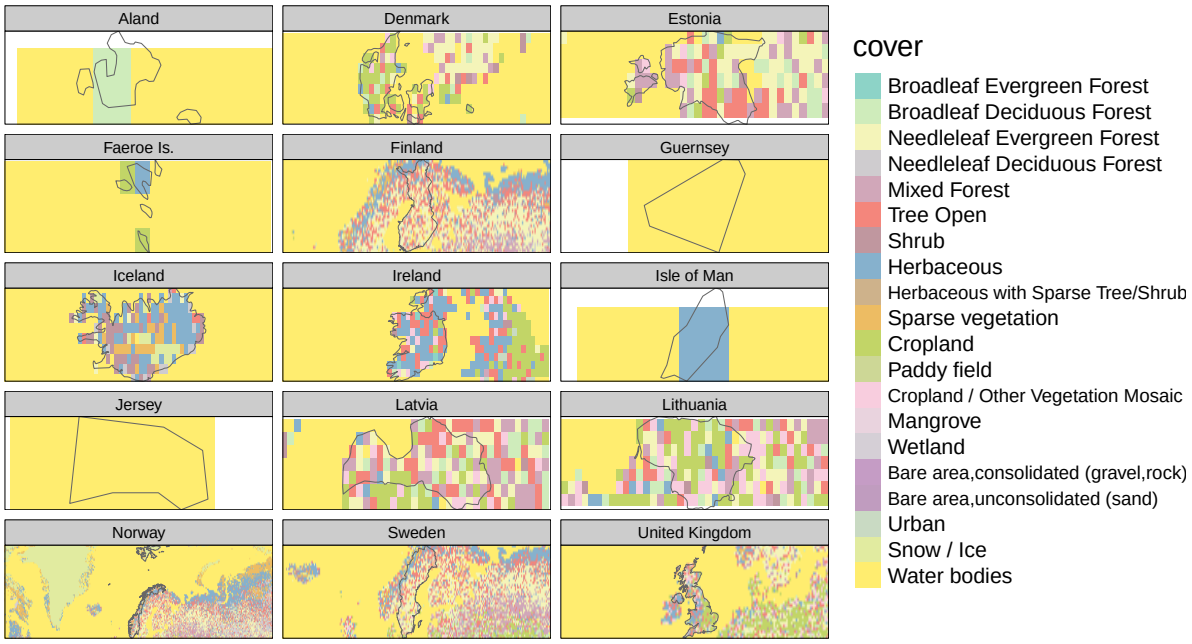

```
# categorical land cover
tm_shape(land) +
    tm_raster("cover")
```




```
# using faceting

# facet by cover to get one map for each cover type
tm_shape(land) +
    tm_raster("cover", legend.show = FALSE) +
    tm_facets("cover", free.coords=TRUE, drop.units=TRUE)
```




```
# facet by country to get one cover map per country
tm_shape(land) +
    tm_raster("cover") + # , legend.show = FALSE)
tm_shape(Europe[which(Europe$part=="Northern Europe"),]) +
    tm_borders() +
    tm_facets("name", free.coords = TRUE)
```



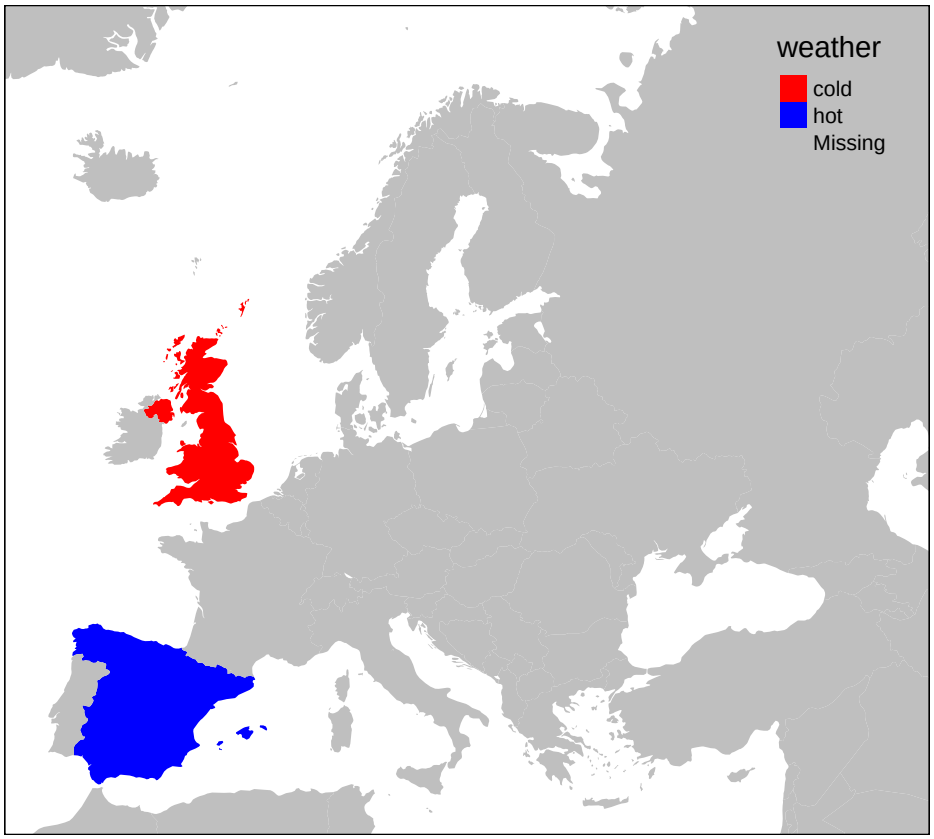
4 Using your own data

You can use both your own geometry and/or attribute data in tmap and other mapping options.

```
# create a very simple data frame as an example
dF <- data.frame( country=c("Spain", "United Kingdom"),
                  weather=c("hot", "cold") )

# tmap can join this onto a map (other options are available too)
Eu_and_dat <- append_data(Europe, dF, key.shp="name", key.data="country")
```

```
## Under coverage. No data for 66 out of 68 polygons: Albania, Aland, Andorra, Armenia, Austria, ...
```

```
tm_shape(Eu_and_dat) +
  tm_fill("weather", palette=c("red","blue"))
```



4 Exercise

```
# Can you plot some other data by country ?
# remember the country names in the tmap map are in Europe$name
install.packages("gapminder")
library(gapminder)
?gapminder
str(gapminder)

Eu_and_gap <- append_data(Europe, gapminder[gapminder$year==2007,], key.shp="name", key.data="country")

tm_shape(Eu_and_gap) +
   tm_fill("lifeExp")
```

5 Interactivity (with leaflet &/or tmap)

Going further

```
# other useful packages, resources

# packages for creating the circular migration plots I showed this morning
install.packages("migest")
```

```r
install.packages("circlize")


# raster package, great for mapping satellite data etc.
install.packages("raster")
library(raster)

# rmapshaper for simplifying polygon boundaries
install.packages("rmapshaper")
library(rmapshaper)
```

Acknowledgements :

Thankyou to all the package developers on whose work this tutorial is based.