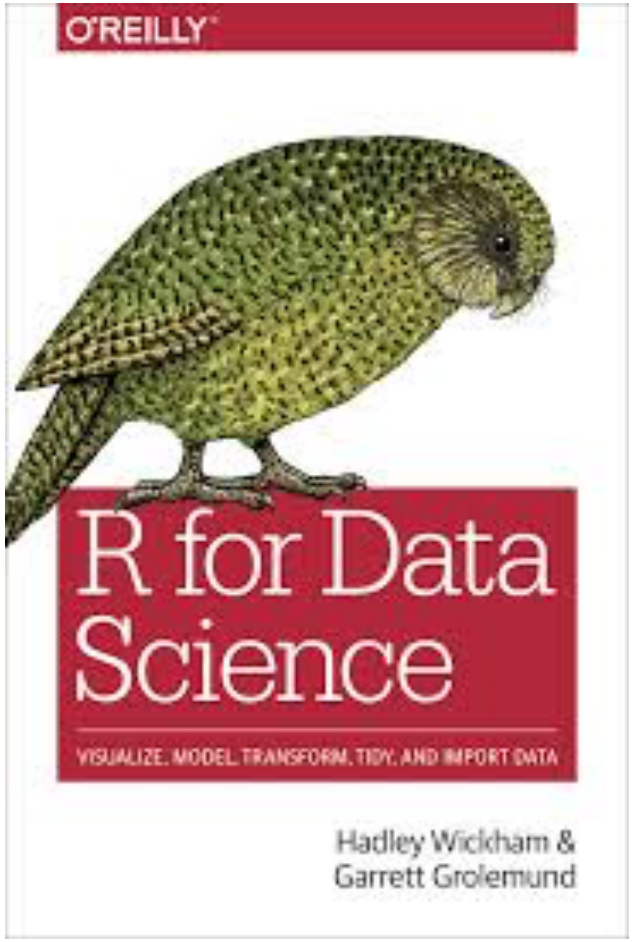
CT474: Smart Grid

Lecture 1: Introduction to Data Science with R

Dr. Jim Duggan,
School of Engineering & Informatics
National University of Ireland Galway.
https://github.com/JimDuggan/PDAR
https://twitter.com/_jimduggan

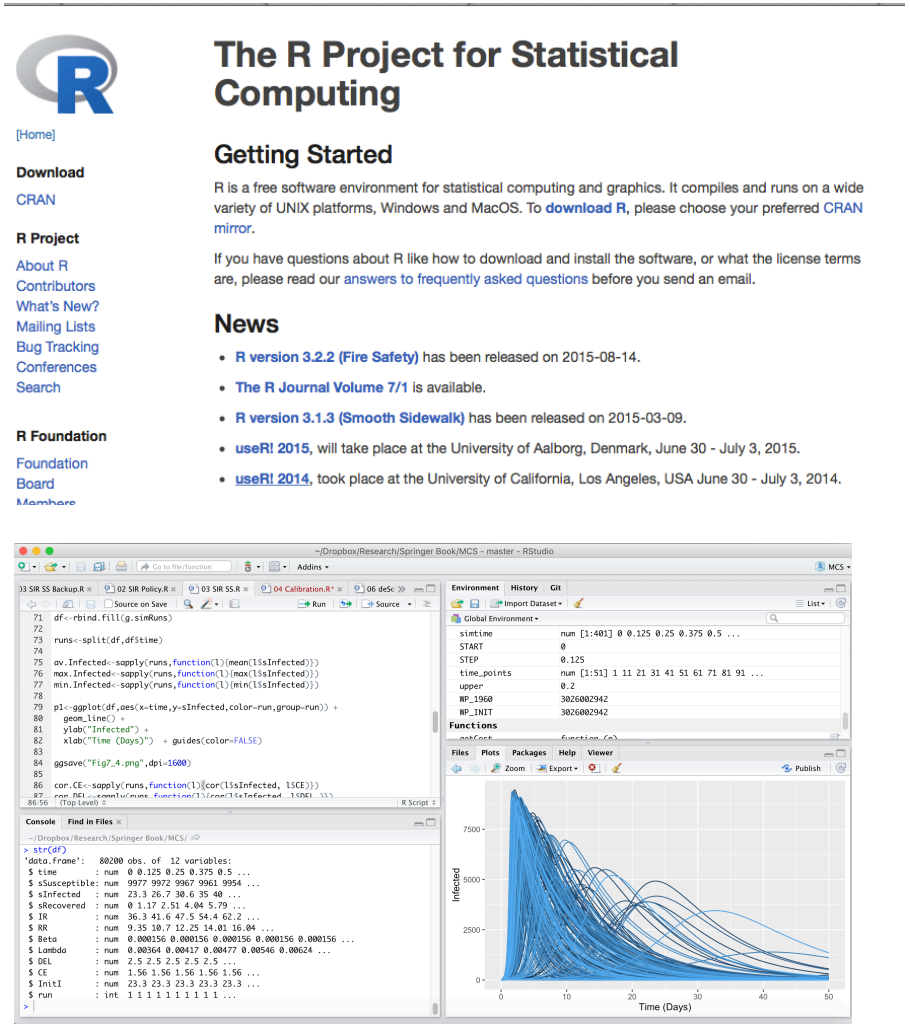
Topic Structure

- Introduction to Data Science and R
- Data Visualisation
- Data Transformation
- Data Modeling
- R Aspects
  - ggplot2
  - Vectors & Functions
  - Data Frames / Tibbles
  - dplyr
  - lm
- Energy examples



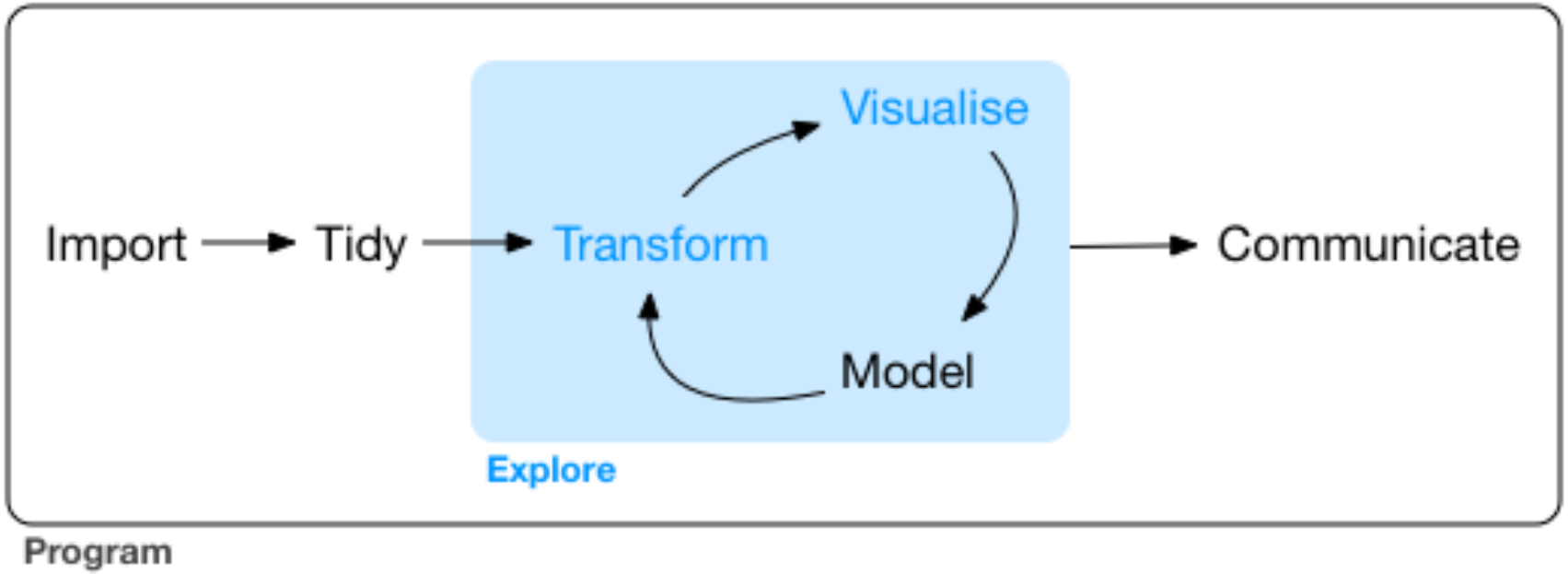
The R Project for Statistical Computing

- R's *mission* is to enable the best and most thorough exploration of data possible (Chambers 2008).
- It is a dialect of the S language, developed at Bell Laboratories
- ACM noted that *S "will forever alter the way people analyze, visualize, and manipulate data"*



Data Exploration

"Data exploration is the art of looking at your data, rapidly generating hypotheses, quickly testing them, then repeating again and again and again." (Wickham and Grolemund 2017).



Data Visualisation with **ggplot2**

"The simple graph has brought more information to the data analyst's mind that any other device." – John Tukey

```
> dt <- ggplot2::mpg
>
> dt
# A tibble: 234 × 11
   manufacturer      model displ  year   cyl      trans   drv   cty   hwy    fl   class
          <chr>      <chr> <dbl> <int> <int>      <chr> <chr> <int> <int> <chr>   <chr>
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
# ... with 224 more rows
```

Fuel Economy Data Set (ggplot2::mpg)

This dataset contains a subset of the fuel economy data that the EPA makes available on http://fueleconomy.gov. It contains only models which had a new release every year between 1999 and 2008 - this was used as a proxy for the popularity of the car.

manufacturer	manufacturer	**drv**	f = front-wheel drive, r = rear wheel drive, 4 = 4wd
model	model name	**cty**	city miles per gallon
displ	engine displacement, in litres	**hwy**	highway miles per gallon
year	year of manufacture	**fl**	fuel type
cyl	number of cylinders	**class**	"type" of car
trans	type of transmission		

First Steps

- Generate a first graph to help answer the following question:
  - *Do cars with big engines use more fuel than cars with small engines*
- What might the relationship between engine size and fuel efficiency look like?
  - Positive or negative?
  - Linear or non-linear?

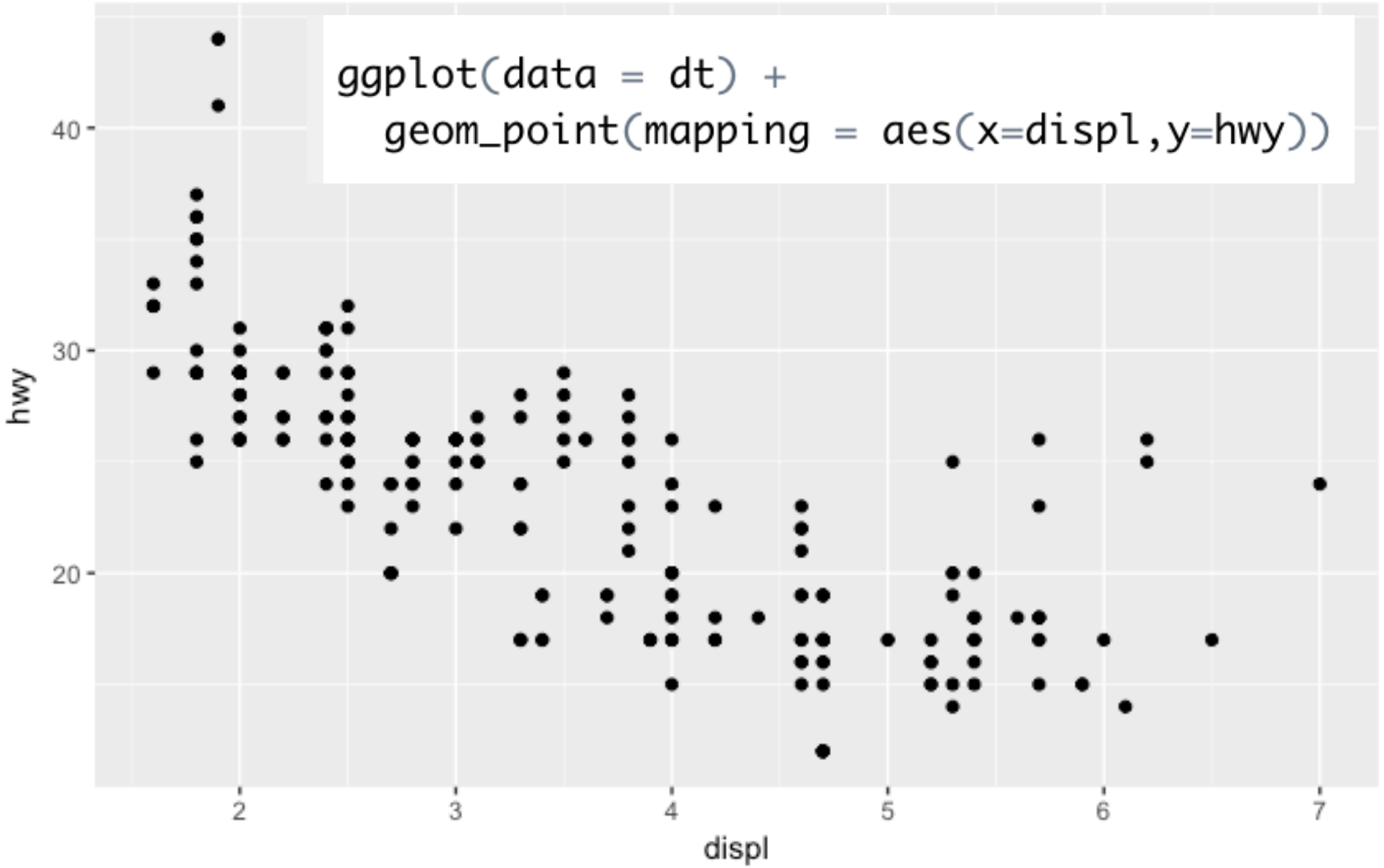
Selecting data

```
> dt
# A tibble: 234 × 11
   manufacturer      model displ  year   cyl      trans   drv   cty   hwy    fl   class
          <chr>      <chr> <dbl> <int> <int>      <chr> <chr> <int> <int> <chr>   <chr>
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
```

- Among the variables are:
  - **displ**, a car's engine size in litres
  - **hwy**, a car's fuel efficiency on the highway in miles per gallon

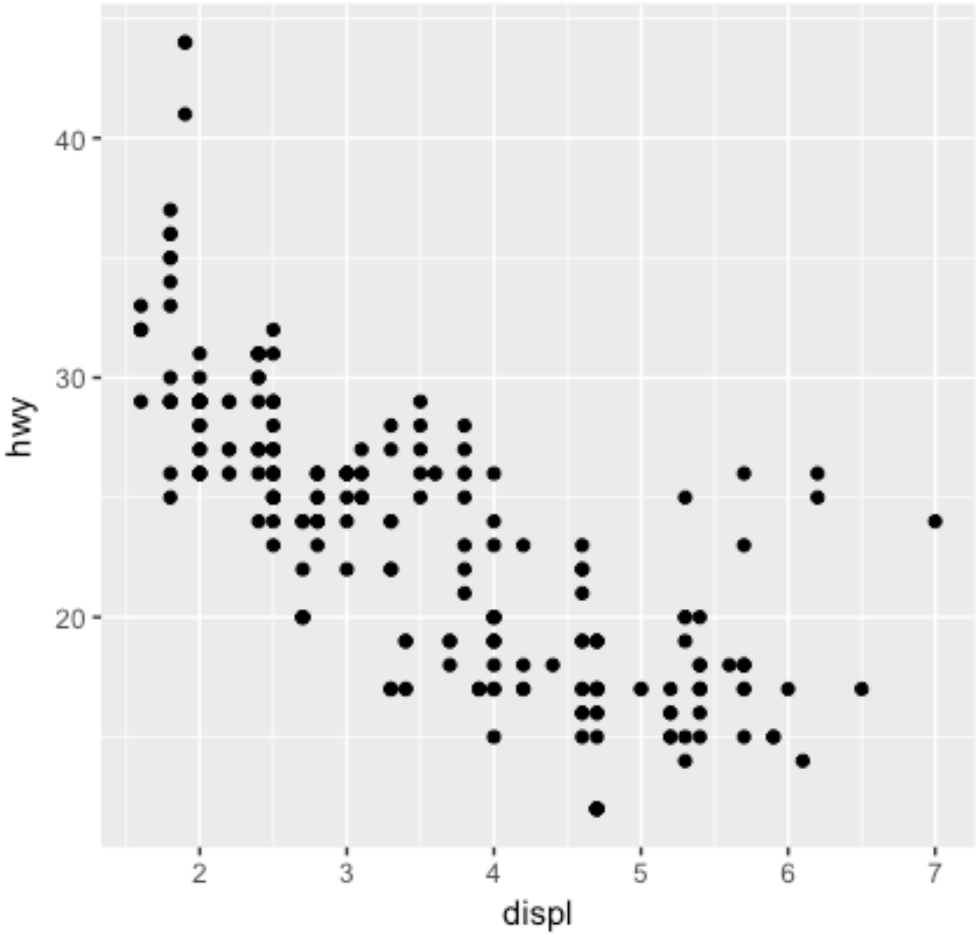
Creating a ggplot

```
ggplot(data = dt) +
  geom_point(mapping = aes(x=displ,y=hwy))
```



Interpreting the plot

- The plot shows a negative relationship between engine size (displ) and fuel efficiency (hwy)
- Cars with big engines use more fuel
- Does this confirm or refute your hypothesis about fuel efficiency and engine size?



A Graphing Template in R

```
ggplot(data = dt) +
  geom_point(mapping = aes(x=displ,y=hwy))
```

- Turn the code into a reusable template for making graphs with ggplot2

ggplot(data = <DATA>) +

<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))

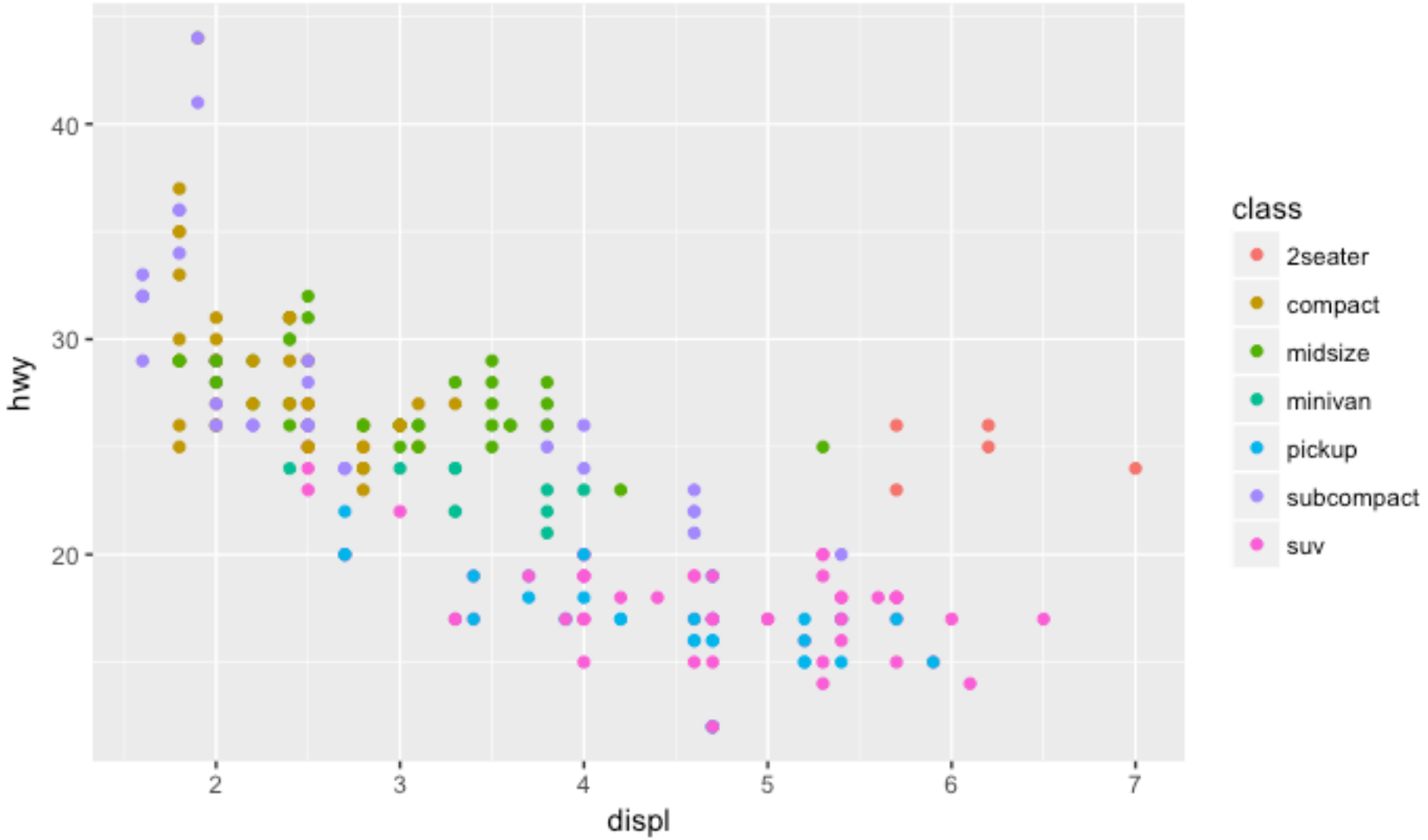
Aesthetic Mappings

"The greatest value of a picture is when it forces us to notice what we never expected to see" – John Tukey

```
> unique(dt$class)
[1] "compact"    "midsize"    "suv"        "2seater"    "minivan"
[6] "pickup"     "subcompact"
```

- A third variable can be added to a 2-D plot by mapping it to an aesthetic.
- An aesthetic is a visual property of the plot's objects.
- An aesthetic's *level* could be colour, size or shape.

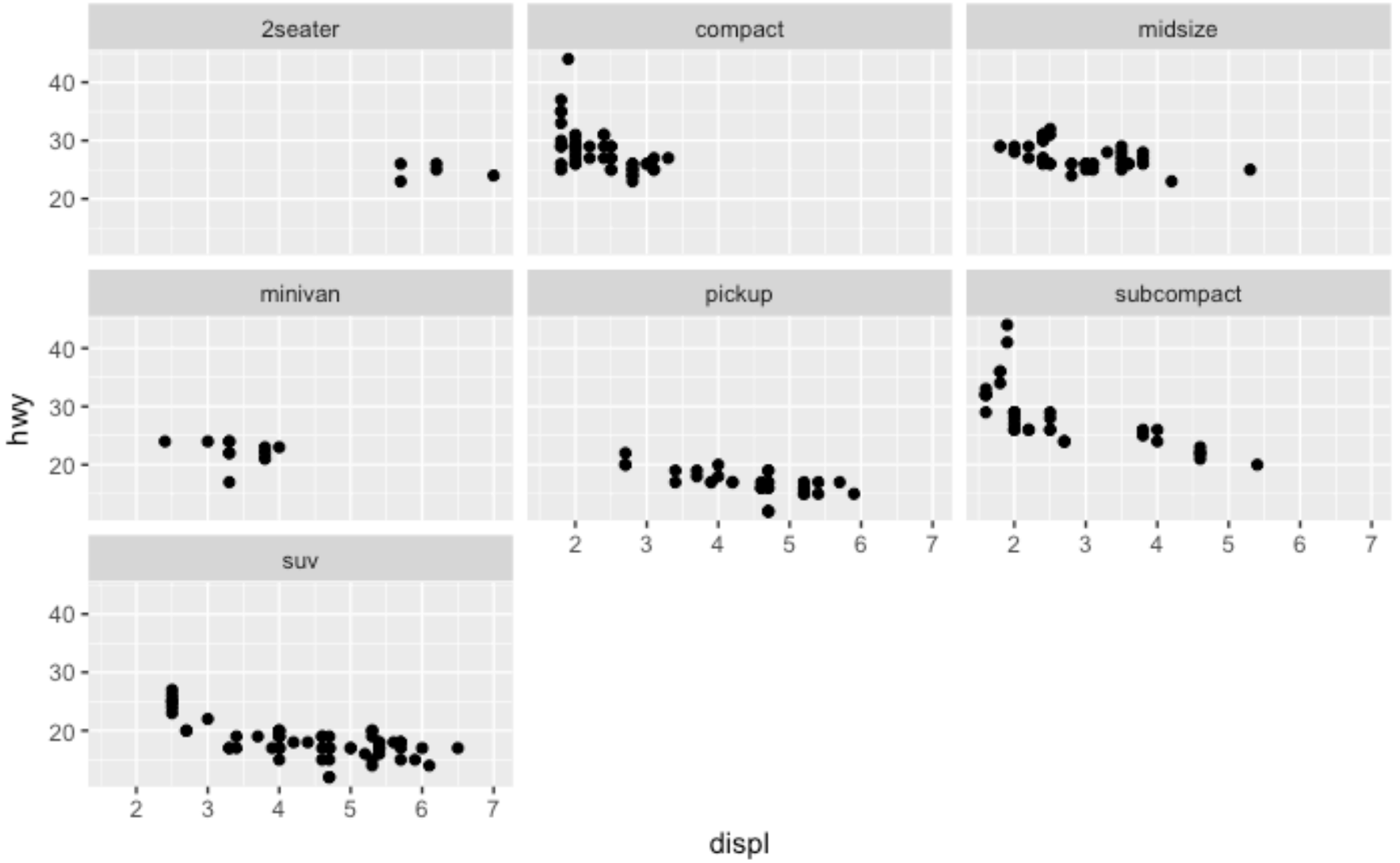
```
ggplot(data = dt) +
  geom_point(mapping = aes(x=displ,y=hwy,colour=class))
```



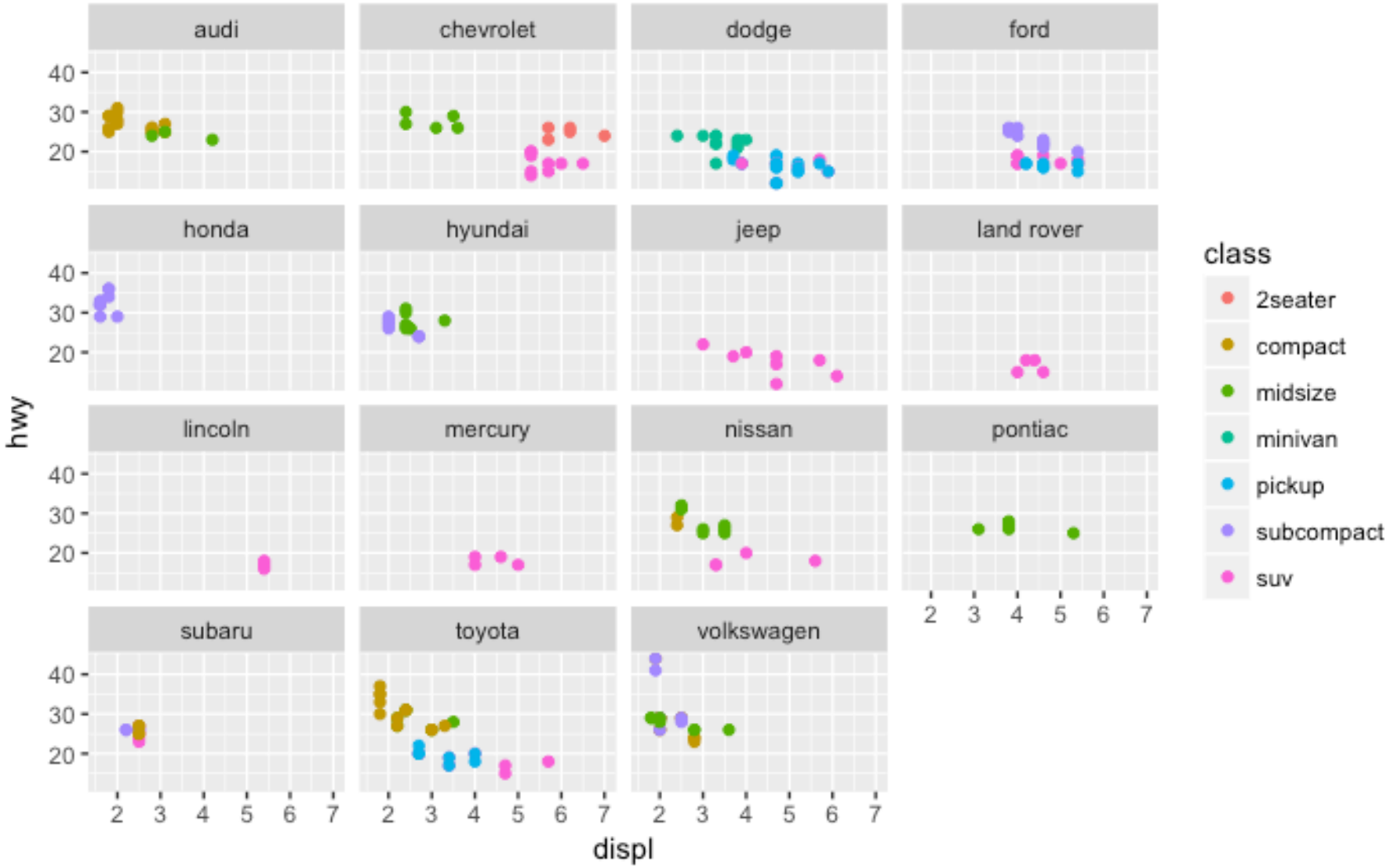
Facets

- Another way to add categorical variables is to split a plot into facets, subplots that display one subset of the data.
- To facet your plot by a single variable, use facet_wrap()

```
ggplot(data = dt) +
  geom_point(mapping = aes(x=displ,y=hwy)) + facet_wrap(~class)
```



```
ggplot(data = dt) +
  geom_point(mapping = aes(x=displ,y=hwy,colour=class)) +
  facet_wrap(~manufacturer)
```



Running R Code

R as a calculator

```
> 1 / 200 * 30
[1] 0.15
>
>
> 10 + 40 / 5
[1] 18
```

Creating objects in R

```
> x <- 10
>
> x
[1] 10
>
>
> y <- 1:10
>
> y
 [1]  1  2  3  4  5  6  7  8  9 10
>
> z <- y * x
>
> z
 [1]  10  20  30  40  50  60  70  80  90 100
>
```

Object names

- Must start with a letter, and can only contain letters, numbers, _ and .
- Object names should be descriptive.
- *snake_case* often recommended

```
> room_temperature <- 20
>
>
> room_temperature
[1] 20
```

Calling functions

- R has a large collection of built-in functions that are called like this

  function_name(arg1=val1, arg2=val2, …)

```
> seq(1,10)
 [1]  1  2  3  4  5  6  7  8  9 10
>
> sqrt(1:10)
 [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427
 [9] 3.000000 3.162278
>
> paste("Hello",1:10,sep="-")
 [1] "Hello-1"  "Hello-2"  "Hello-3"  "Hello-4"  "Hello-5"  "Hello-6"  "Hello-7"
 [8] "Hello-8"  "Hello-9"  "Hello-10"
```

R – Data Types

	Homogenous	Heterogenous
1d	Atomic Vector	List
2d	Matrix	Data Frame
nd	Array	

- To understand a data structure, use the str() function

```
> x
[1] 1 2 3 4 5
> str(x)
 int [1:5] 1 2 3 4 5
```

Vectors

- The basic data structure in R is the Vector
- Vectors come in two flavours
  - Atomic vectors
  - Lists
- They have 3 common properties
  - Type, typeof(), what it is
  - Length, length(), how many elements
  - *Attributes, attributes, additional metadata*

Atomic Vectors

- A sequence of elements that have the same data type (Matloff 2009)
- Four common types
  - logical
  - integer
  - double (or numeric)
  - character
- Usually created with c() – short for combine

```
> dbl_var <- c(2.2, 2.5, 2.9)
> str(dbl_var)
 num [1:3] 2.2 2.5 2.9
>
> int_var <- c(0L, 1L, 2L)
> str(int_var)
 int [1:3] 0 1 2
>
> log_var<- c(TRUE, TRUE, F, FALSE)
> str(log_var)
 logi [1:4] TRUE TRUE FALSE FALSE
>
> chr_var<- c("CT5102","CT561")
> str(chr_var)
 chr [1:2] "CT5102" "CT561"
```

Atomic vector types

```
> dbl_var
[1] 2.2 2.5 2.9
> typeof(dbl_var)
[1] "double"
>
> int_var
[1] 0 1 2
> typeof(int_var)
[1] "integer"
```

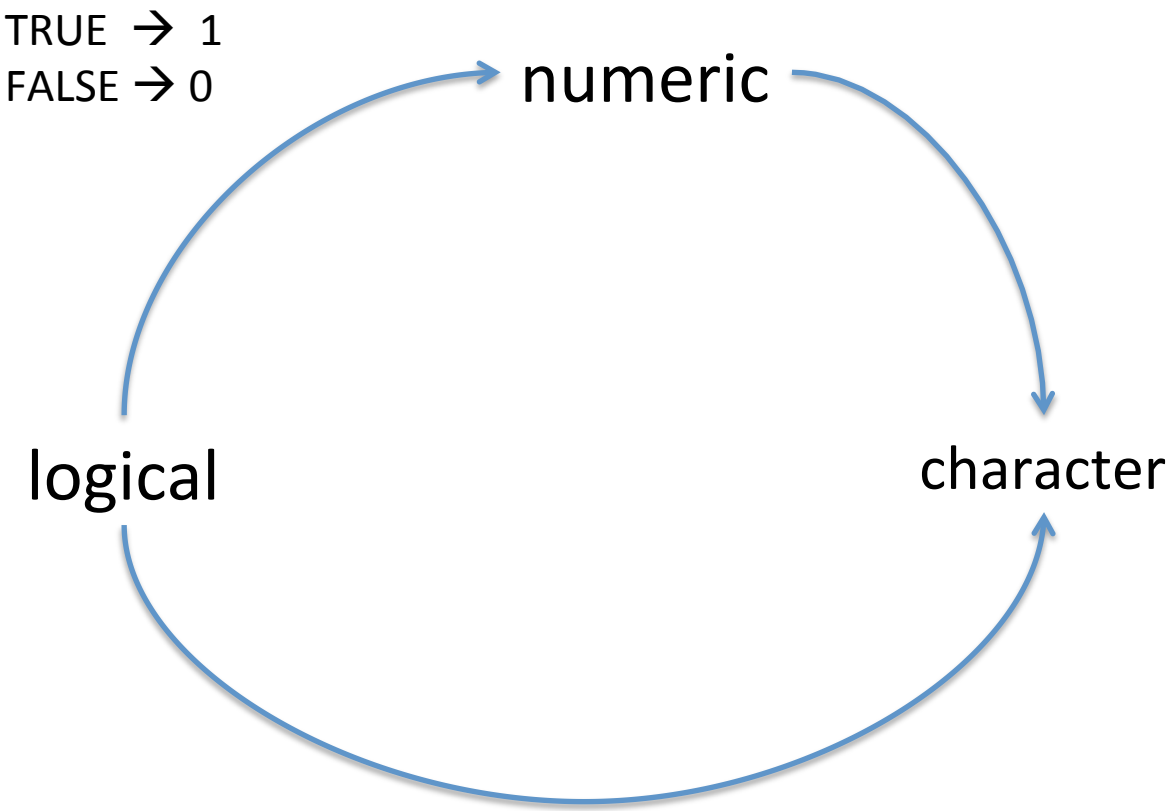
```
> log_var
[1]  TRUE  TRUE FALSE FALSE
> typeof(log_var)
[1] "logical"
>
> chr_var
[1] "CT5102" "CT561"
> typeof(chr_var)
[1] "character"
```

Coercion of atomic vectors

- All elements of an atomic vector MUST be of the same type
- When different type are combined, they will be coerced into the most flexible types
- What will be the type and values of
  - c(1L, T, F)
  - c(1,T,F)

Coercion Rules
Least to most flexible

- logical
- integer
- double
- character



Grolemund (2014) p 52

Challenge 1.1

- Determine the types for each of the following (coerced) vectors

```
v1<- c(1L, T, FALSE)

v2<- c(1L, T, FALSE, 2)

v3<- c(T, FALSE, 2, "FALSE")

v4<- c(2L, "FALSE")

v5<- c(0L, 1L, 2.11)
```

Creating atomic vectors using sequences

The colon operator (:) generates regular sequences (atomic vectors) within a specified range.

```
> v1<-1:10
> v1
 [1]  1  2  3  4  5  6  7  8  9 10

> v2<-3:13
> v2
 [1]  3  4  5  6  7  8  9 10 11 12 13
```

Subsetting Atomic Vectors

- R's subsetting operators are powerful and fast
- For atomic vectors, the operator [is used
- There are six ways to subset an atomic vector in R
- In R, the index for a vector starts at 1

```
> x<-c(2.1, 4.2, 3.3, 5.4)
>
> x
[1] 2.1 4.2 3.3 5.4
>
>
> x[1]
[1] 2.1
> x[c(1,3)]
[1] 2.1 3.3
```

(1) Positive integers

Positive integers return elements at the specified position


```
> x<-1:5
>
> x
[1] 1 2 3 4 5
>
> x[1:2]
[1] 1 2
>
> x[5]
[1] 5
>
> x[5:1]
[1] 5 4 3 2 1
```

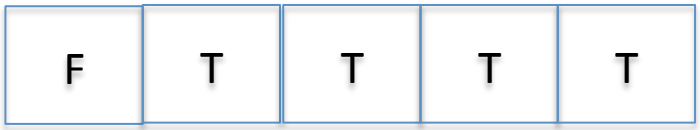
(2) Negative integers

Negative integers omit elements at specified positions

```
> x
[1] 1 2 3 4 5
>
> x[-1]
[1] 2 3 4 5
>
> x[-(3:4)]
[1] 1 2 5
```

(3) Logical Vectors

Select elements where the corresponding logical value is TRUE. This approach supports ***recycling***




```
> x
[1] 1 2 3 4 5
> x[c(F,T,T,T,T)]
[1] 2 3 4 5
```

```
> x
[1] 1 2 3 4 5
>
> x[c(T,F)]
[1] 1 3 5
```

Logical Vectors - Advantages

Expressions can be used to create a logical vector

```
> x
[1] 1 2 3 4 5
> b <- x < median(x)
>
> b
[1]  TRUE  TRUE FALSE FALSE FALSE
> x[b]
[1] 1 2
> x[x<median(x)]
[1] 1 2
```

Logical Expressions in R

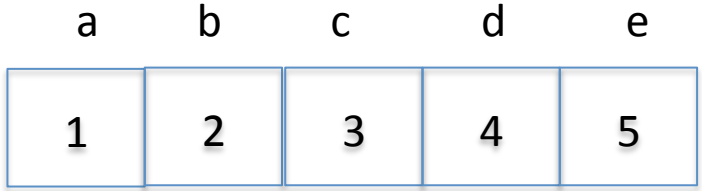
Operators	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	not x
x | y	x OR y
x & y	x AND y

```
> x
[1] 1 2 3 4 5
>
> b<- x<median(x) | x > median(x)
>
> b
[1]  TRUE  TRUE FALSE  TRUE  TRUE
>
> x[b]
[1] 1 2 4 5
```

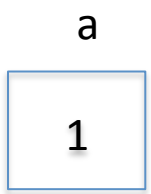
Challenge 1.2

- Create an R vector of squares of 1 to 10
- Find the minimum
- Find the maximum
- Find the average
- Subset all those values greater than the average

(4) Character vectors



Return elements with matching names




```
> x<-1:5
> x
[1] 1 2 3 4 5
> letters[x]
[1] "a" "b" "c" "d" "e"
>
> names(x)<-letters[x]
>
> x
a b c d e
1 2 3 4 5
>
> x["a"]
a
1
```

Vectorization

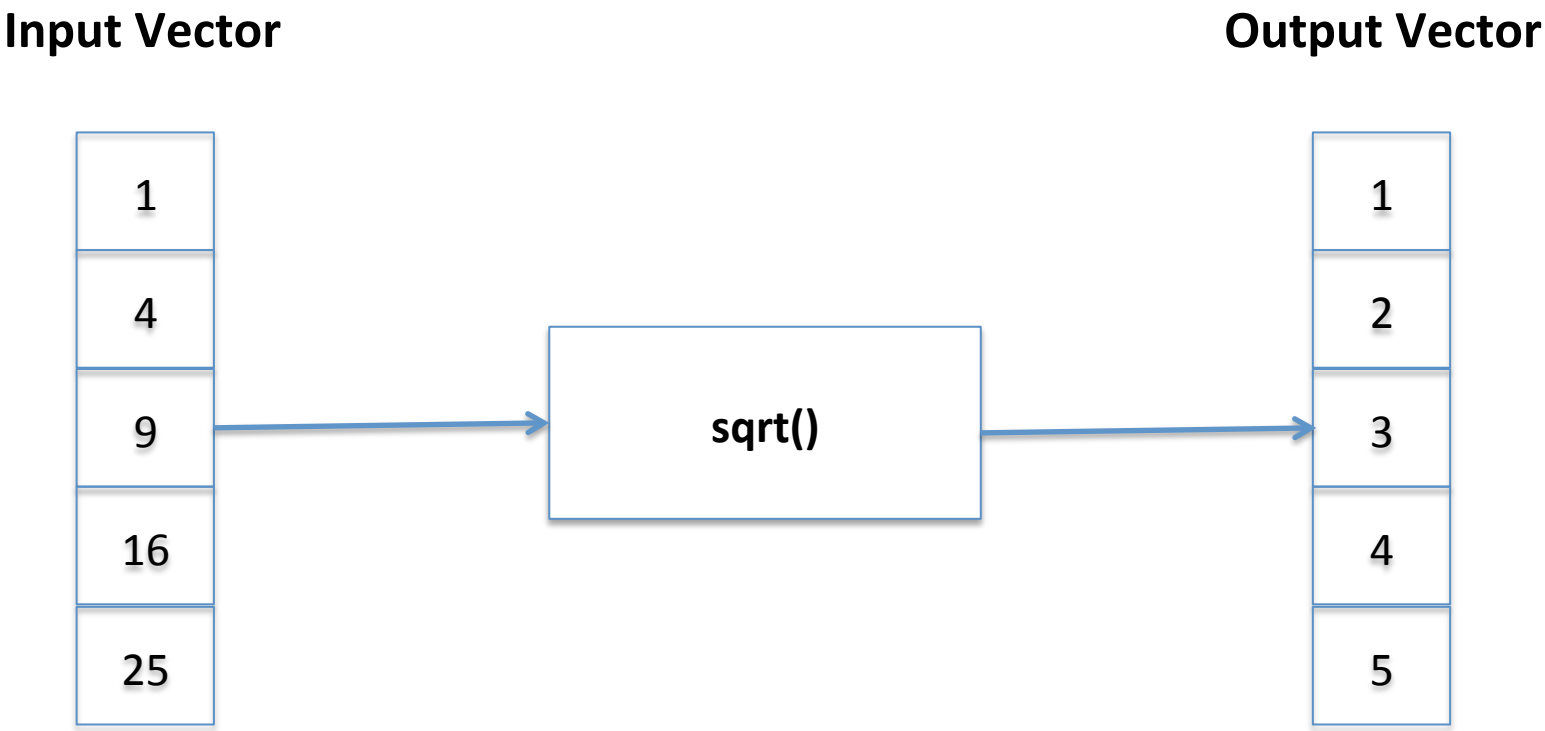
- A powerful feature of R is that it supports *vectorization*
- Functions can operate on every element of a vector, and return the results of each individual operation in a new vector.

```
> v1
[1] 1 2 3 4 5

> r<-sqrt(v1)

> r
[1] 1.000000 1.414214 1.732051 2.000000 2.236068
```

Key Idea



Arithmetic Operators

- Arithmetic operations can also be applied to vectors in an element-wise manner

```
> v1
[1] 1 2 3 4 5
> v2<-3*v1

> v2
[1]  3  6  9 12 15

> v3<-v1+v2
> v3
[1]  4  8 12 16 20
```

Vectorized if/else

- Vectors can also be processed using the vectorized **ifelse(b,u,v)** function, which accepts a boolean vector **b** and allocates the element-wise results to be either **u** or **v**.

```
> v1
[1]  1  4  9 16 25

> c1<-ifelse(v1%%2==0,"Even","Odd")

> c1
[1] "Odd"  "Even" "Odd"  "Even" "Odd"
```