CT5102: Programming for Data Analytics

Lecture 8: Processing Dates and Data Visualisation

Dr. Jim Duggan,

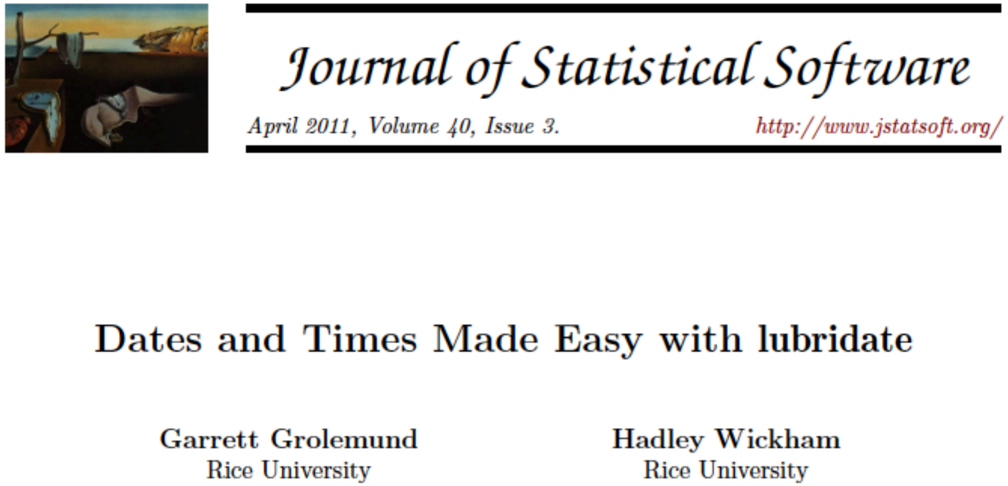
School of Engineering & Informatics

National University of Ireland Galway.

https://github.com/JimDuggan/PDAR

https://twitter.com/_jimduggan

Dates and Times



Journal of Statistical Software

April 2011, Volume 40, Issue 3. http://www.jstatsoft.org/

Dates and Times Made Easy with lubridate

Garrett Grolemund
Rice University

Hadley Wickham
Rice University

- Date-time data can be frustrating to work with
- Syntax in base R can be confusing and difficult to remember
- Can be complicated when doing arithmetic

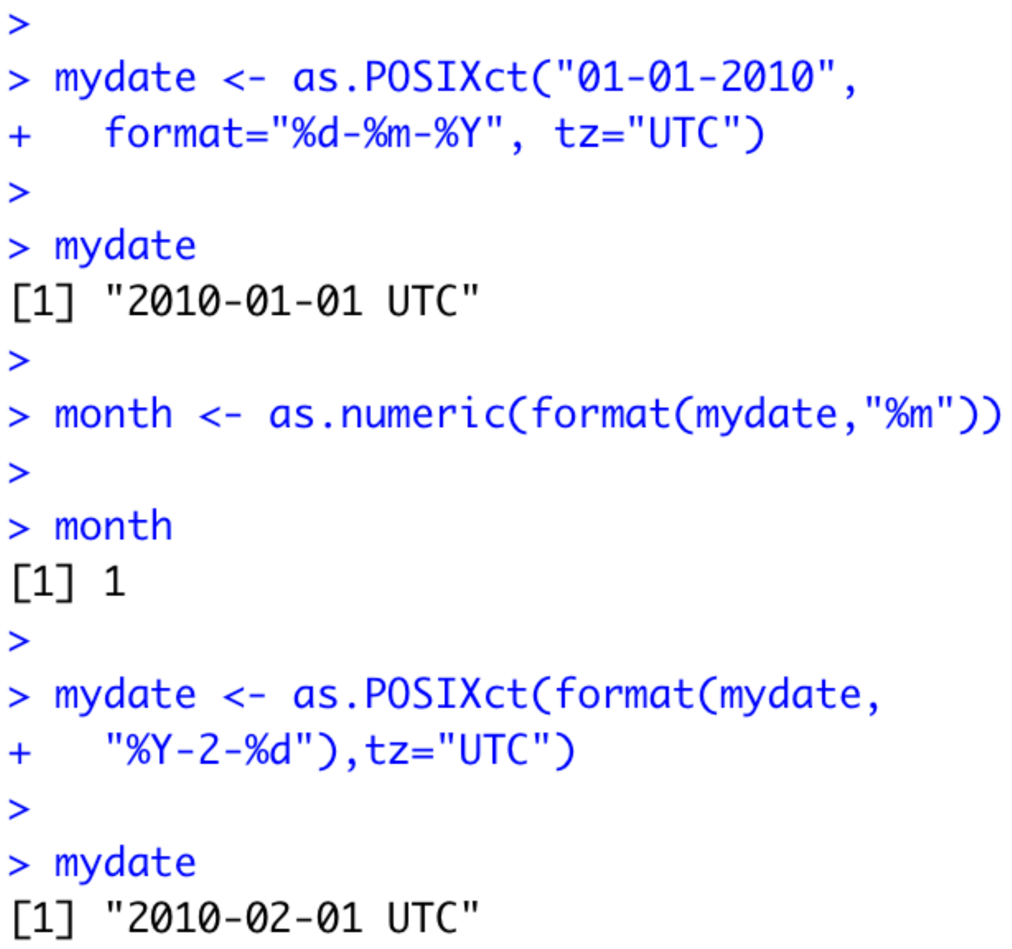
lubridate package

- Helps users to:
  - Identify and parse date-time data
  - Extract and modify components of date-time, such as years, months, days, hours, minutes and seconds
  - Perform accurate calculations with date-times and timespans
  - Handle time zones and daylight savings time

Example

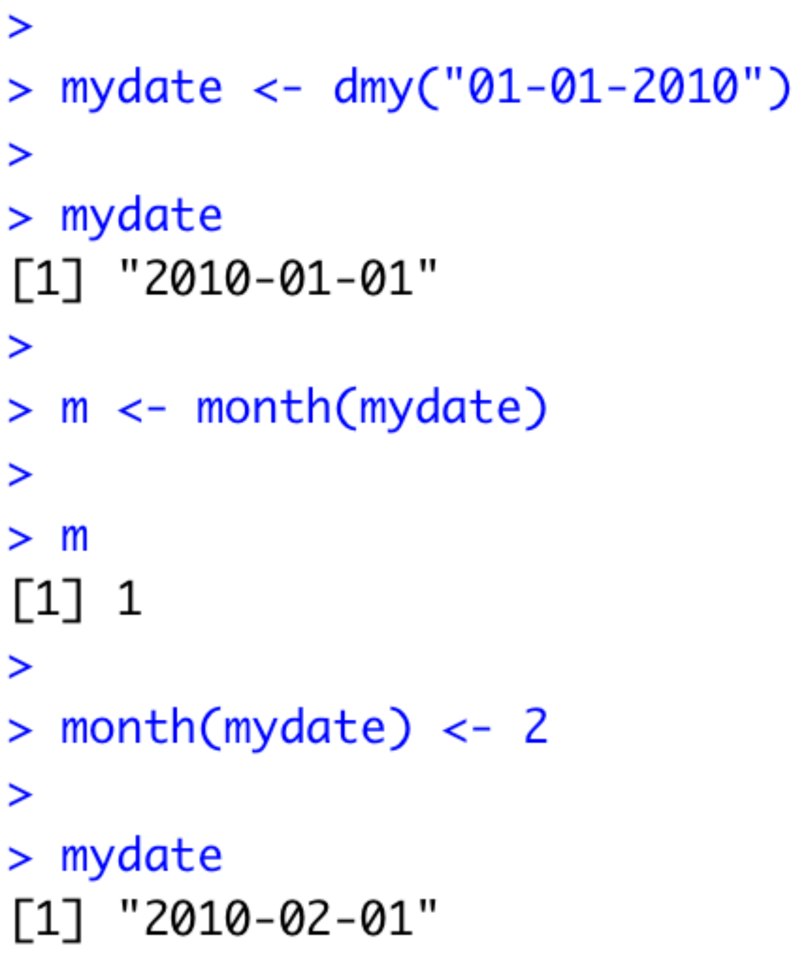
- Given a character string:
  - Read it in as a date-time object
  - Extract the month
  - Change the month to February
- Approaches
  - Base R method
  - lubridate method

Using Base R



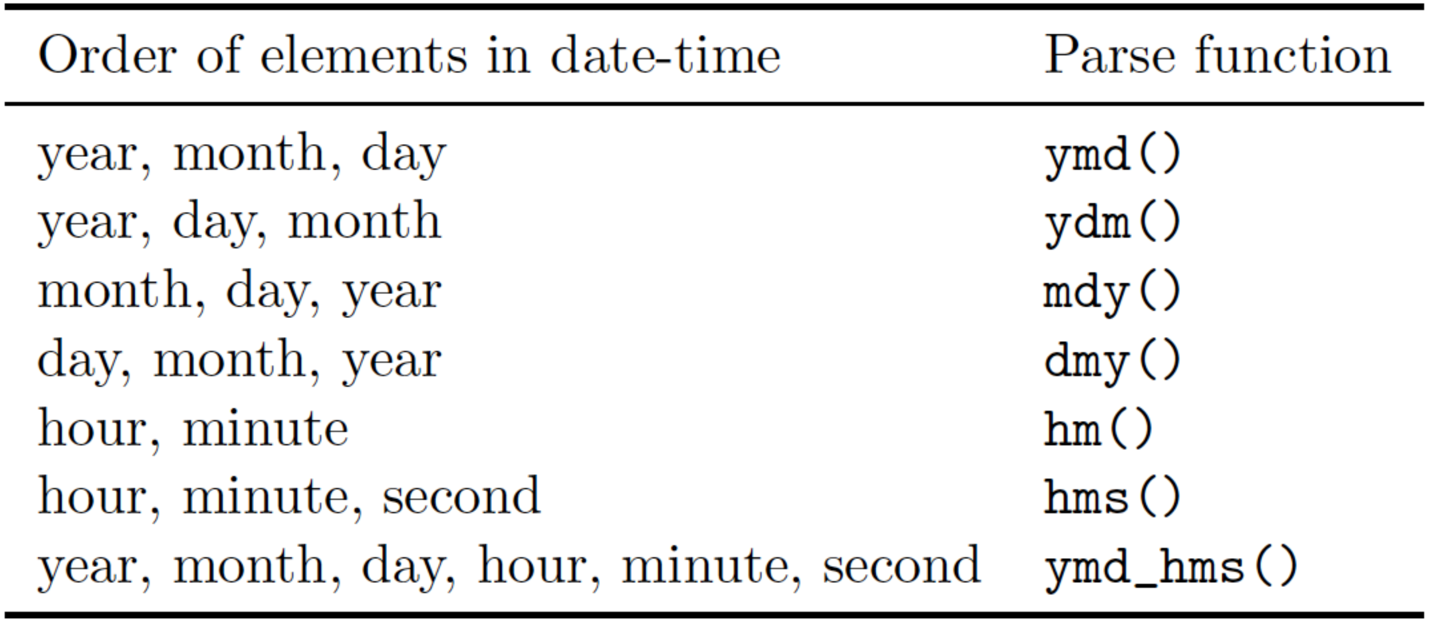
```
>
> mydate <- as.POSIXct("01-01-2010",
+   format="%d-%m-%Y", tz="UTC")
>
> mydate
[1] "2010-01-01 UTC"
>
> month <- as.numeric(format(mydate,"%m"))
>
> month
[1] 1
>
> mydate <- as.POSIXct(format(mydate,
+   "%Y-2-%d"),tz="UTC")
>
> mydate
[1] "2010-02-01 UTC"
```

With lubridate

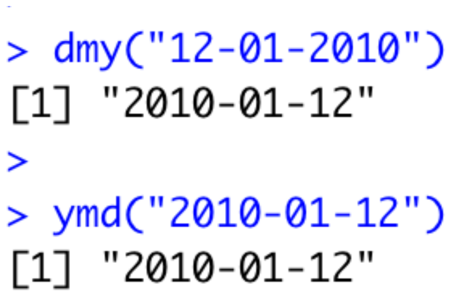


```
>
> mydate <- dmy("01-01-2010")
>
> mydate
[1] "2010-01-01"
>
> m <- month(mydate)
>
> m
[1] 1
>
> month(mydate) <- 2
>
> mydate
[1] "2010-02-01"
```

Parsing Functions…



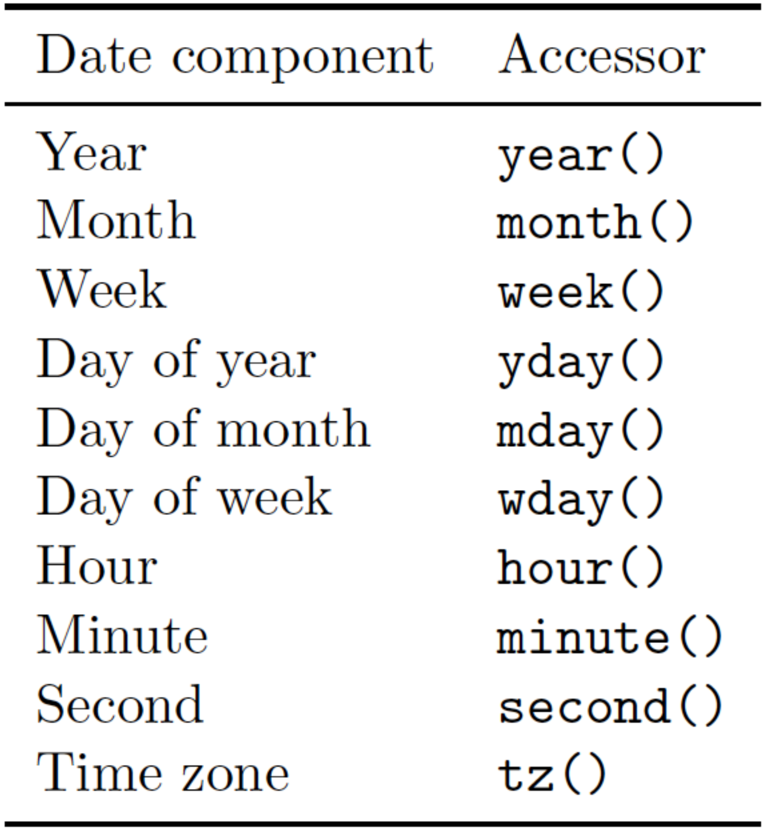
Order of elements in date-time	Parse function
year, month, day	ymd()
year, day, month	ydm()
month, day, year	mdy()
day, month, year	dmy()
hour, minute	hm()
hour, minute, second	hms()
year, month, day, hour, minute, second	ymd_hms()



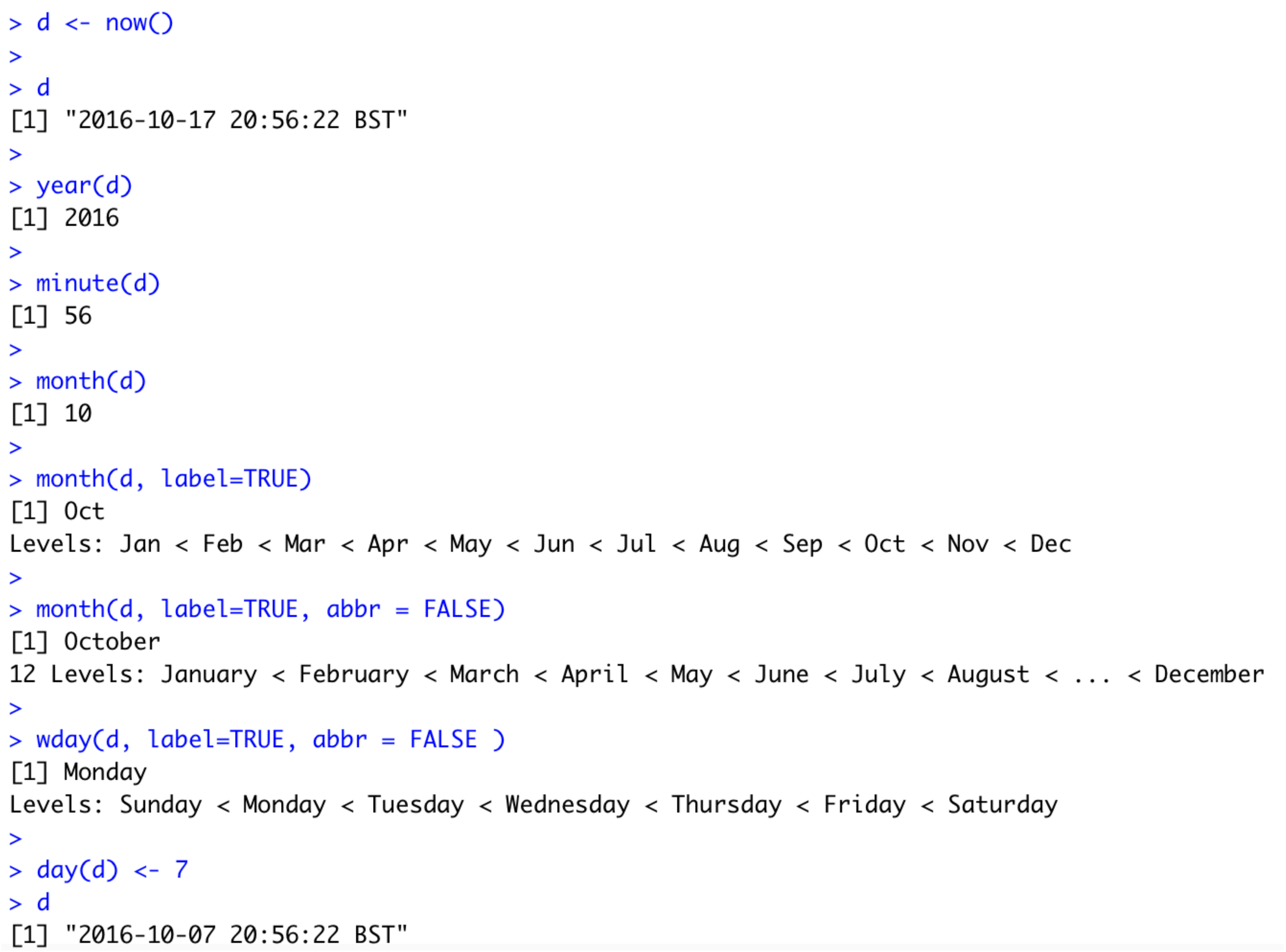
```
> dmy("12-01-2010")
[1] "2010-01-12"
>
> ymd("2010-01-12")
[1] "2010-01-12"
```

Manipulating date-times

- Each element of a date-time object can be extracted
- Accessor functions allow this

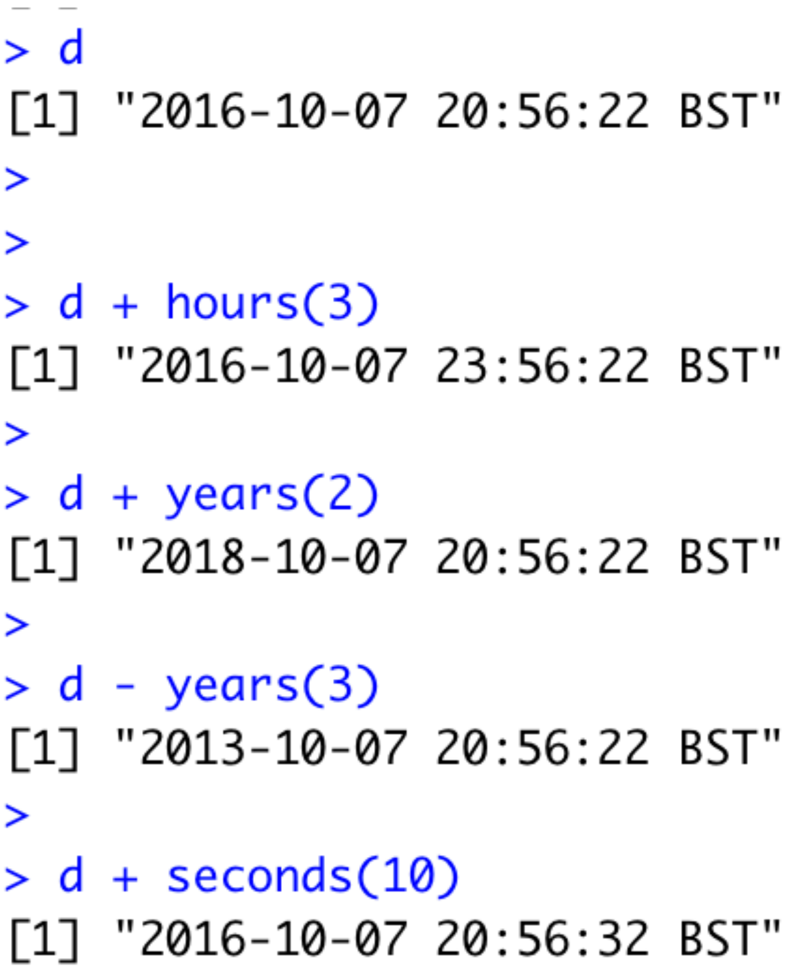


Date component	Accessor
Year	`year()`
Month	`month()`
Week	`week()`
Day of year	`yday()`
Day of month	`mday()`
Day of week	`wday()`
Hour	`hour()`
Minute	`minute()`
Second	`second()`
Time zone	`tz()`



```
> d <- now()
>
> d
[1] "2016-10-17 20:56:22 BST"
>
> year(d)
[1] 2016
>
> minute(d)
[1] 56
>
> month(d)
[1] 10
>
> month(d, label=TRUE)
[1] Oct
Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < Aug < Sep < Oct < Nov < Dec
>
> month(d, label=TRUE, abbr = FALSE)
[1] October
12 Levels: January < February < March < April < May < June < July < August < ... < December
>
> wday(d, label=TRUE, abbr = FALSE )
[1] Monday
Levels: Sunday < Monday < Tuesday < Wednesday < Thursday < Friday < Saturday
>
> day(d) <- 7
> d
[1] "2016-10-07 20:56:22 BST"
```

Manipulating time…



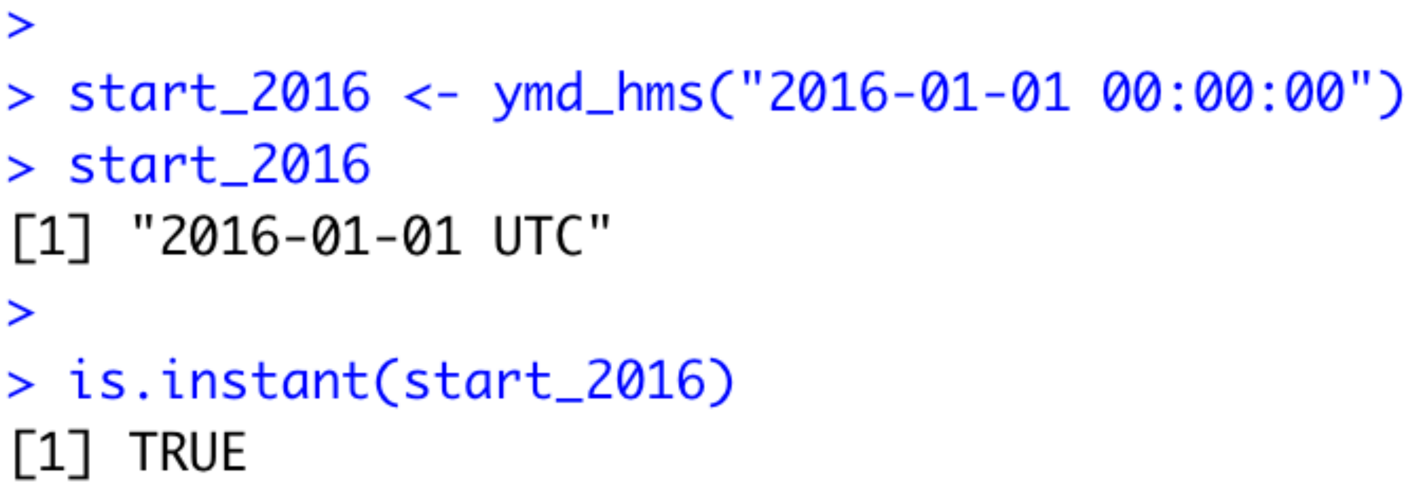
```
> d
[1] "2016-10-07 20:56:22 BST"
>
>
> d + hours(3)
[1] "2016-10-07 23:56:22 BST"
>
> d + years(2)
[1] "2018-10-07 20:56:22 BST"
>
> d - years(3)
[1] "2013-10-07 20:56:22 BST"
>
> d + seconds(10)
[1] "2016-10-07 20:56:32 BST"
```

Arithmetic with date-times

- lubridate allows arithmetic with both relative and exact units by introducing four new time-related objects:
  - Instants
  - Intervals
  - Durations
  - Periods
- Concepts borrowed from the ***Joda Time project***

(1) Instants

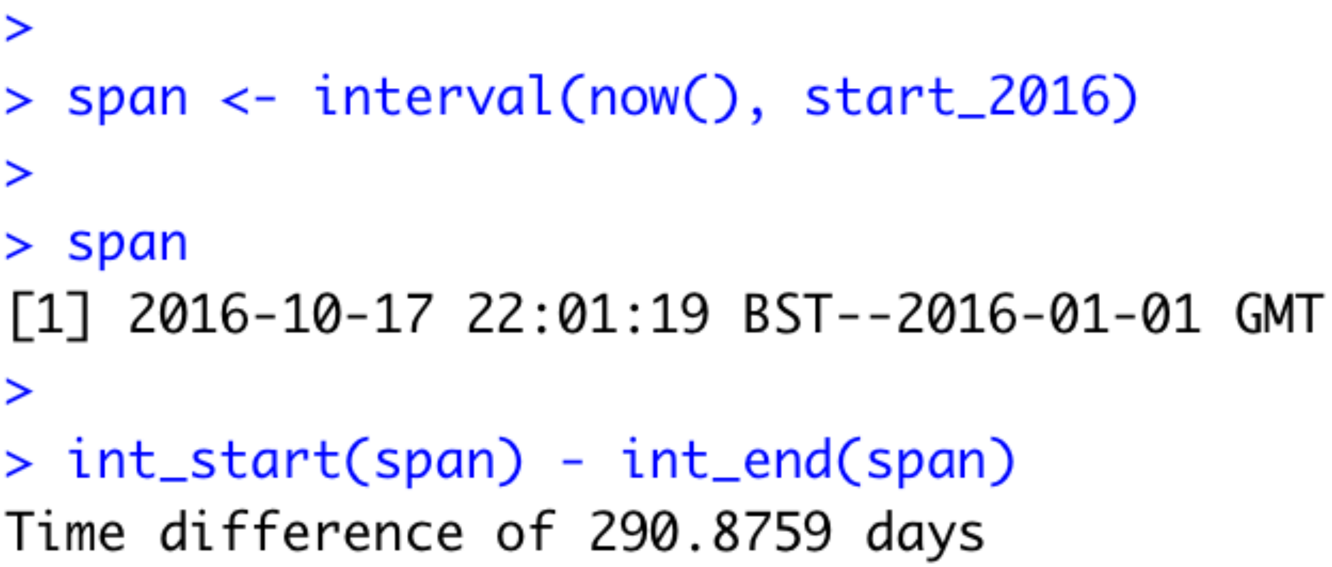
- An instant is a specific moment in time, for example January 1st 2016. We create an instant each time we parse a date in R.



```
>
> start_2016 <- ymd_hms("2016-01-01 00:00:00")
> start_2016
[1] "2016-01-01 UTC"
>
> is.instant(start_2016)
[1] TRUE
```

(2) Intervals

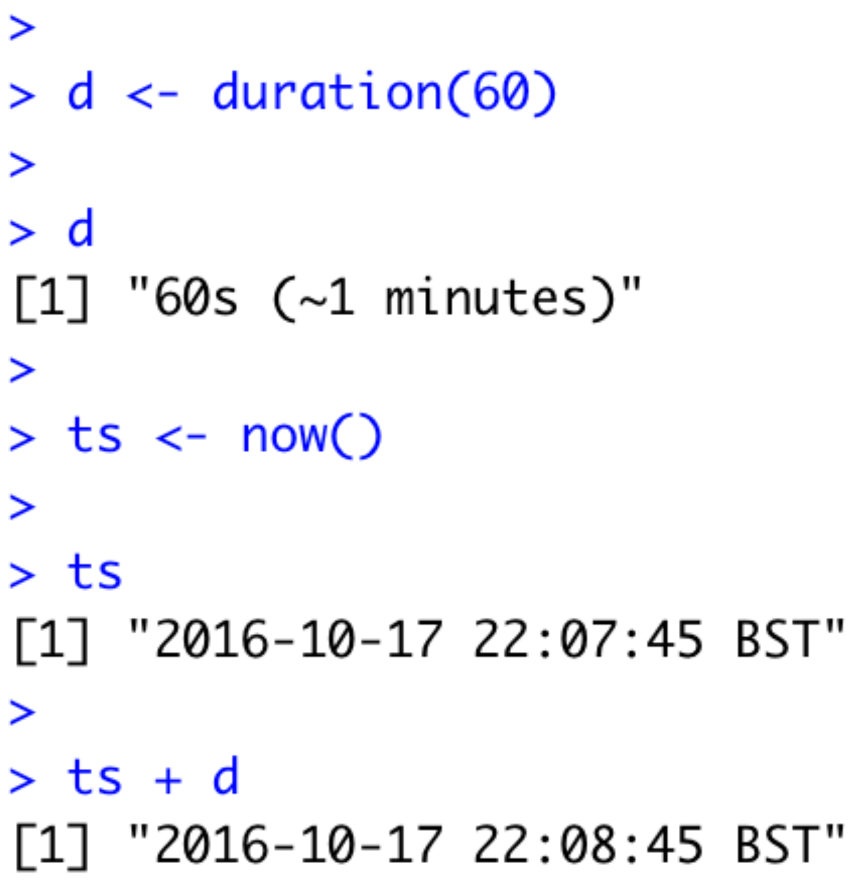
- An interval is a span of time between two specific instants
- The function interval() can be used



```
>
> span <- interval(now(), start_2016)
>
> span
[1] 2016-10-17 22:01:19 BST--2016-01-01 GMT
>
> int_start(span) - int_end(span)
Time difference of 290.8759 days
```

(3) Durations

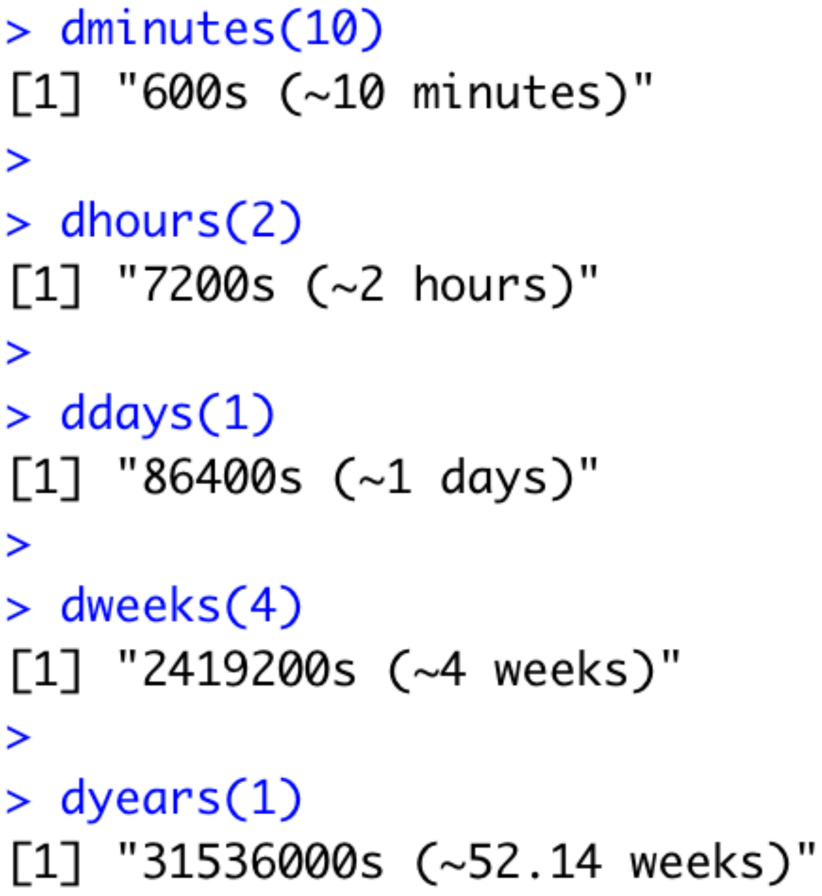
- If we remove the start and end dates from an interval, we will have a generic time span that we can add to any date. These time spans are called durations.
- Durations have consistent lengths



```
>
> d <- duration(60)
>
> d
[1] "60s (~1 minutes)"
>
> ts <- now()
>
> ts
[1] "2016-10-17 22:07:45 BST"
>
> ts + d
[1] "2016-10-17 22:08:45 BST"
```

Helper functions

- Can be used to create duration objects

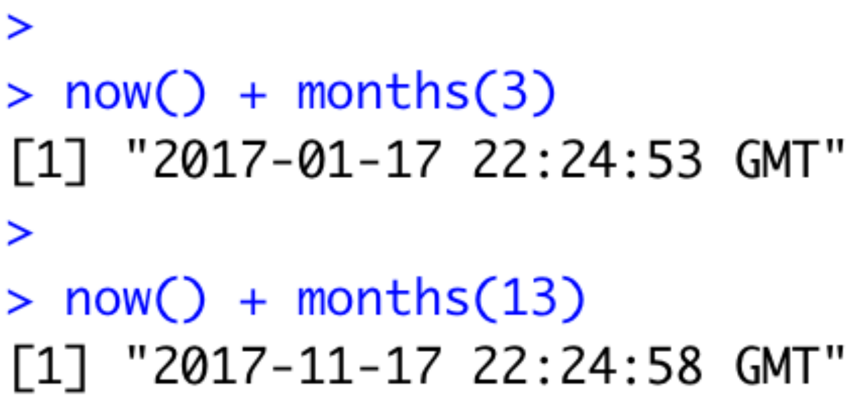


```
> dminutes(10)
[1] "600s (~10 minutes)"
>
> dhours(2)
[1] "7200s (~2 hours)"
>
> ddays(1)
[1] "86400s (~1 days)"
>
> dweeks(4)
[1] "2419200s (~4 weeks)"
>
> dyears(1)
[1] "31536000s (~52.14 weeks)"
```

(4) Periods

- Periods record a time span in units larger than seconds
- Period objects use the helper functions:
  - years()
  - months()
  - weeks()
  - days()
  - hours()
  - minutes()
  - seconds()

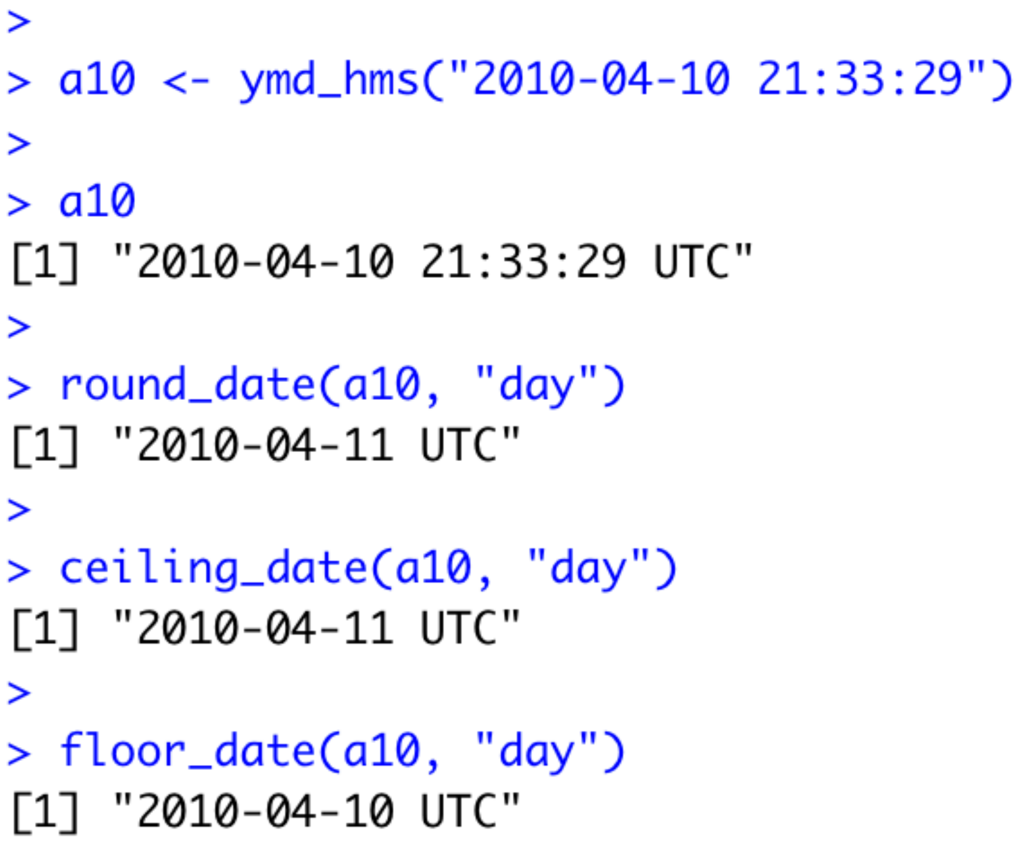
- Periods no longer have consistent lengths
- For example, months(2) always has the length of 2 months.



```
>
> now() + months(3)
[1] "2017-01-17 22:24:53 GMT"
>
> now() + months(13)
[1] "2017-11-17 22:24:58 GMT"
```

Rounding dates

- lubridate provides three methods that help perform rounding
  - round_date()
  - floor_date()
  - ceiling_date()



```
>
> a10 <- ymd_hms("2010-04-10 21:33:29")
>
> a10
[1] "2010-04-10 21:33:29 UTC"
>
> round_date(a10, "day")
[1] "2010-04-11 UTC"
>
> ceiling_date(a10, "day")
[1] "2010-04-11 UTC"
>
> floor_date(a10, "day")
[1] "2010-04-10 UTC"
```

Further Topics (see paper)

- Time zones
- Case Study 1 (holidays)
- Case Study 2 (LA Lakers Basketball)

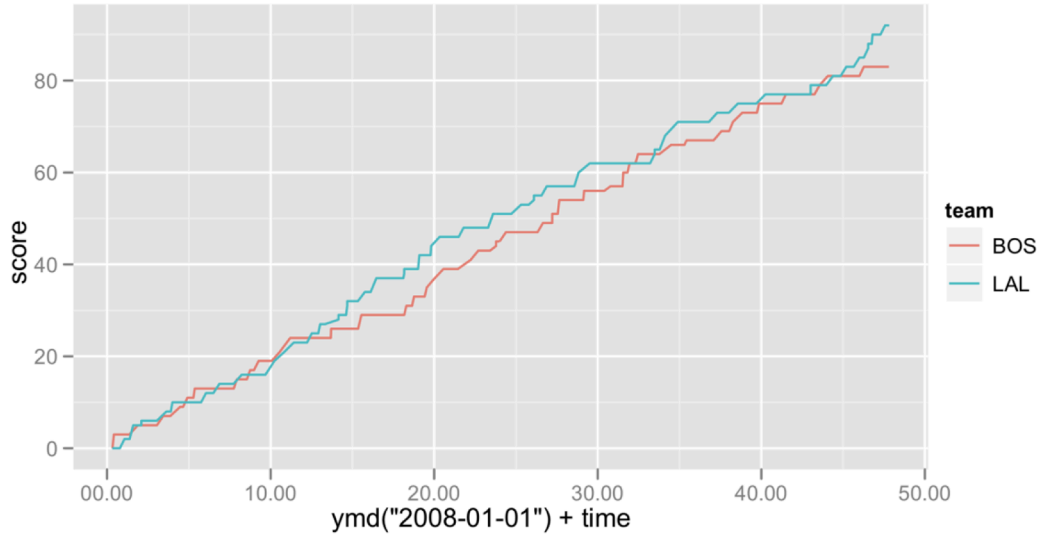
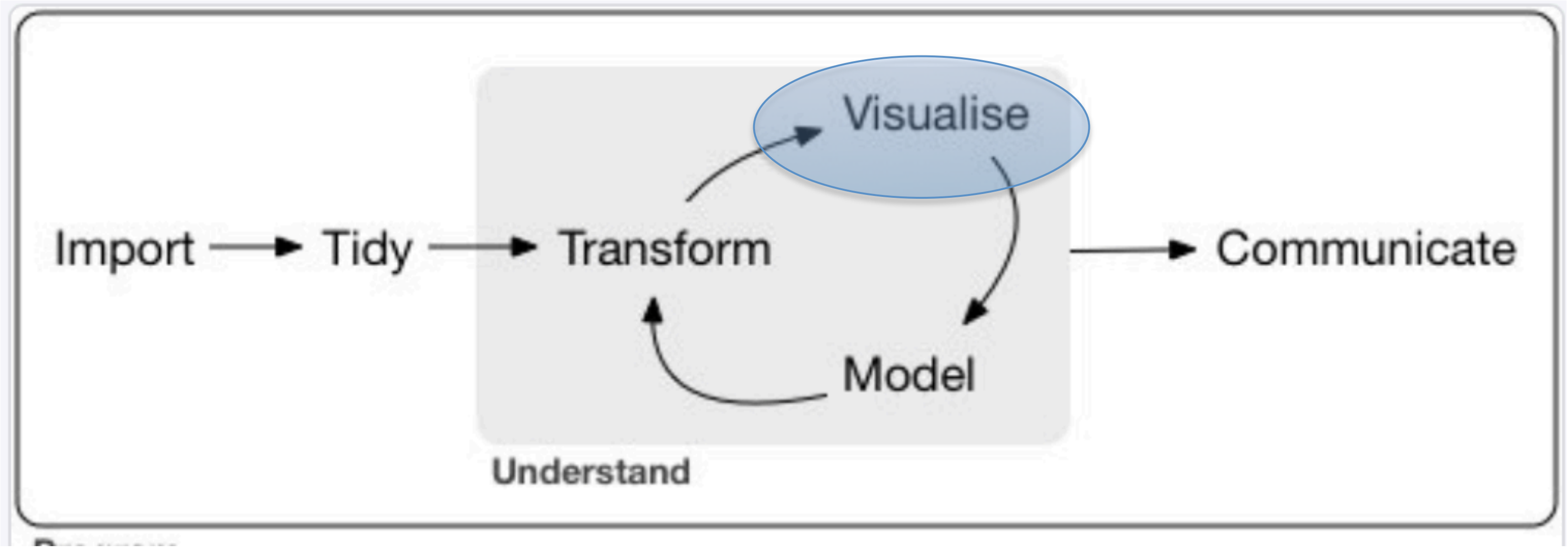


Figure 5: The lead changed between the Lakers and Celtics numerous times during the game.

Visualisation (library ggplot2)

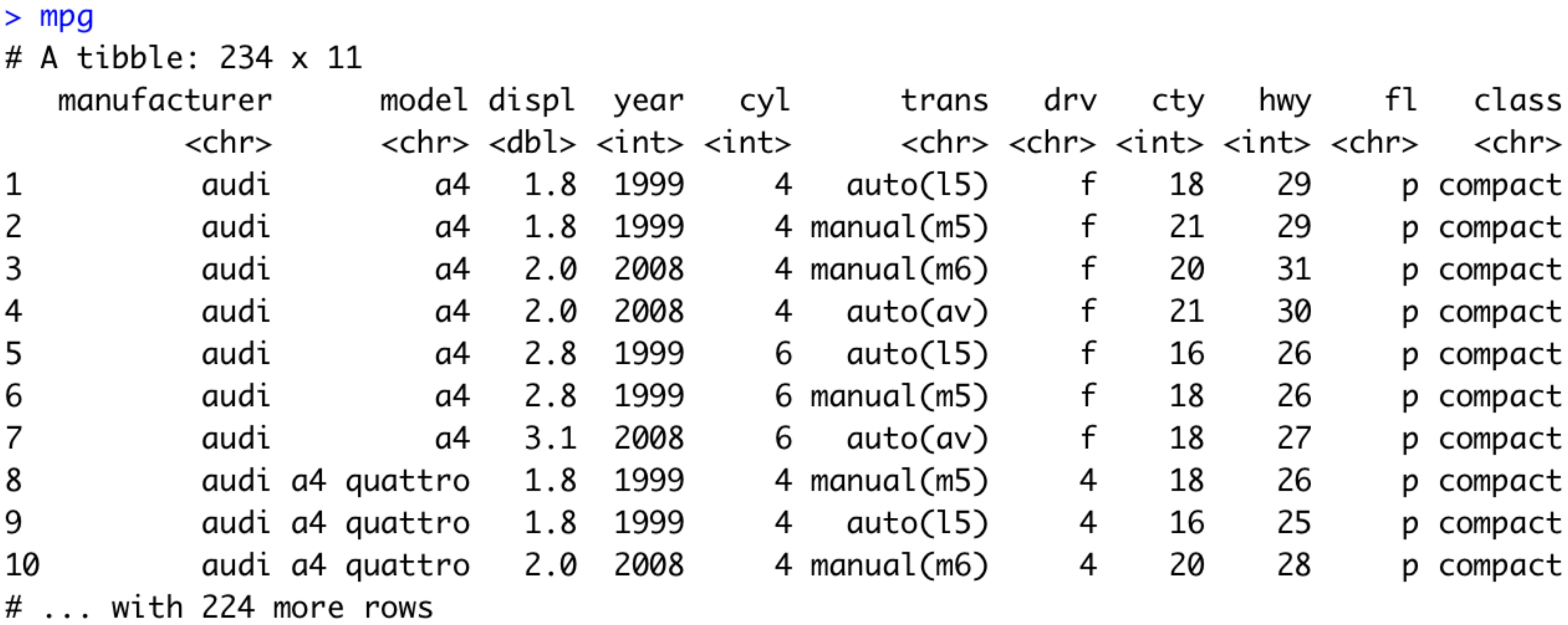
- Name based on Leland Wilkinson’s *grammar of graphics,* which provides a formal, structured perspective on how to describe data graphics
- **ggplot2** package developed by Hadley Wickham



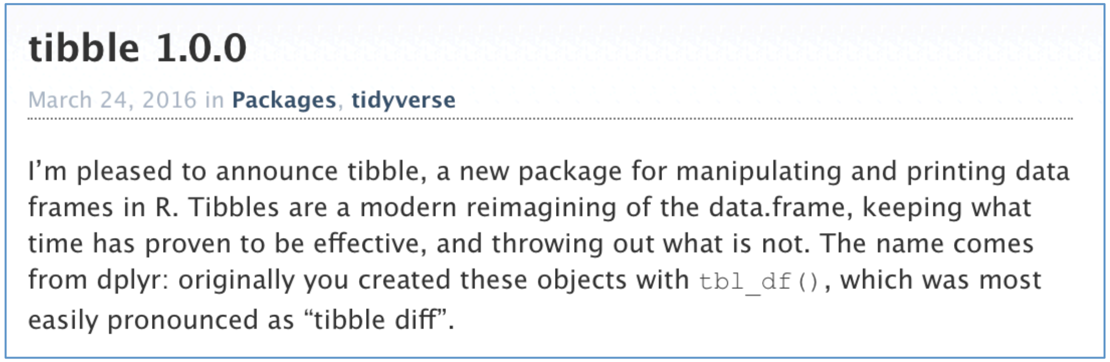
Plot components (Wickham 2016)

- Data
- A set of aesthetic mappings between variables in the data and visual properties
- At least one layer which describes how to render each observation. Layers are usually created with a **geom** function

Example (library ggplot2)



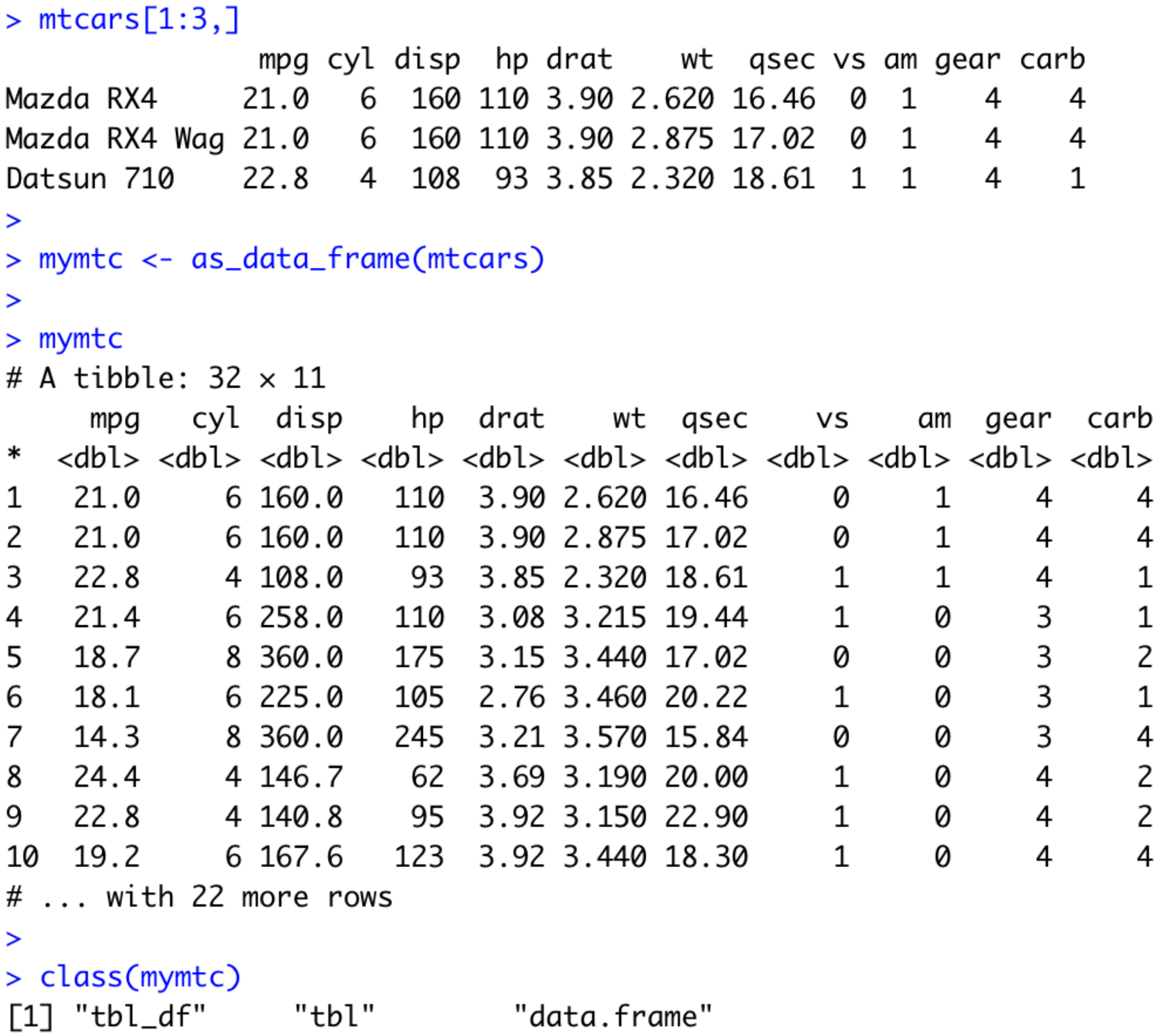
```
> mpg
# A tibble: 234 x 11
   manufacturer        model displ  year   cyl      trans   drv   cty   hwy    fl   class
          <chr>        <chr> <dbl> <int> <int>      <chr> <chr> <int> <int> <chr>   <chr>
1          audi           a4   1.8  1999     4   auto(l5)     f    18    29     p compact
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
```



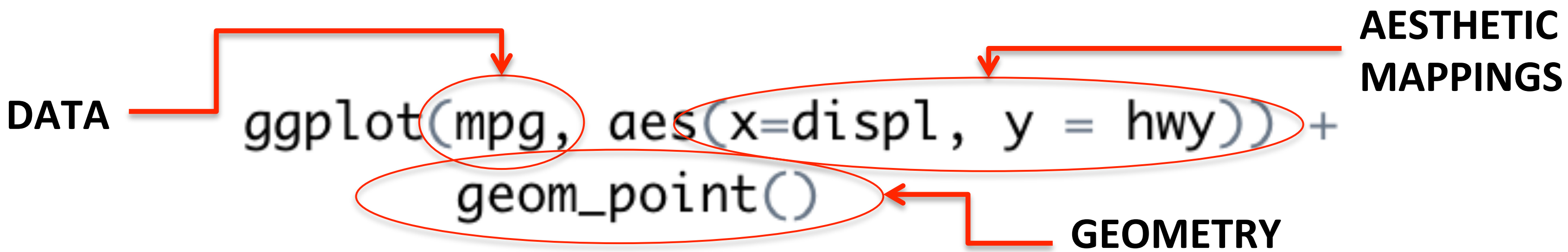
tibble 1.0.0

March 24, 2016 in **Packages**, **tidyverse**

I'm pleased to announce tibble, a new package for manipulating and printing data frames in R. Tibbles are a modern reimagining of the data.frame, keeping what time has proven to be effective, and throwing out what is not. The name comes from dplyr: originally you created these objects with `tbl_df()`, which was most easily pronounced as "tibble diff".



```
> mtcars[1:3,]
               mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4     21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710    22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
>
> mymtc <- as_data_frame(mtcars)
>
> mymtc
# A tibble: 32 × 11
     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
*  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1   21.0     6 160.0   110  3.90 2.620 16.46     0     1     4     4
2   21.0     6 160.0   110  3.90 2.875 17.02     0     1     4     4
3   22.8     4 108.0    93  3.85 2.320 18.61     1     1     4     1
4   21.4     6 258.0   110  3.08 3.215 19.44     1     0     3     1
5   18.7     8 360.0   175  3.15 3.440 17.02     0     0     3     2
6   18.1     6 225.0   105  2.76 3.460 20.22     1     0     3     1
7   14.3     8 360.0   245  3.21 3.570 15.84     0     0     3     4
8   24.4     4 146.7    62  3.69 3.190 20.00     1     0     4     2
9   22.8     4 140.8    95  3.92 3.150 22.90     1     0     4     2
10  19.2     6 167.6   123  3.92 3.440 18.30     1     0     4     4
# ... with 22 more rows
>
> class(mymtc)
[1] "tbl_df"     "tbl"        "data.frame"
```

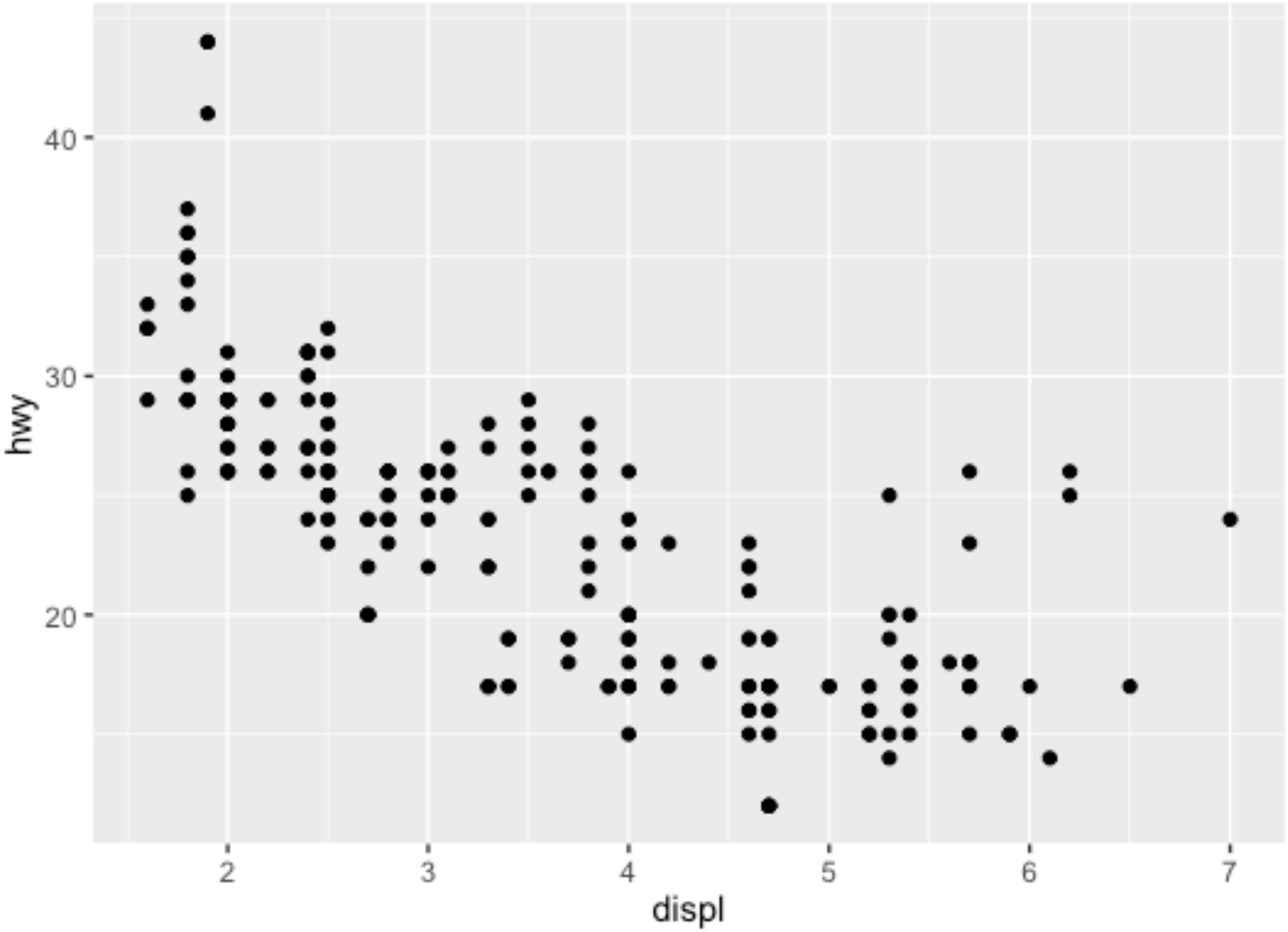


DATA

AESTHETIC MAPPINGS

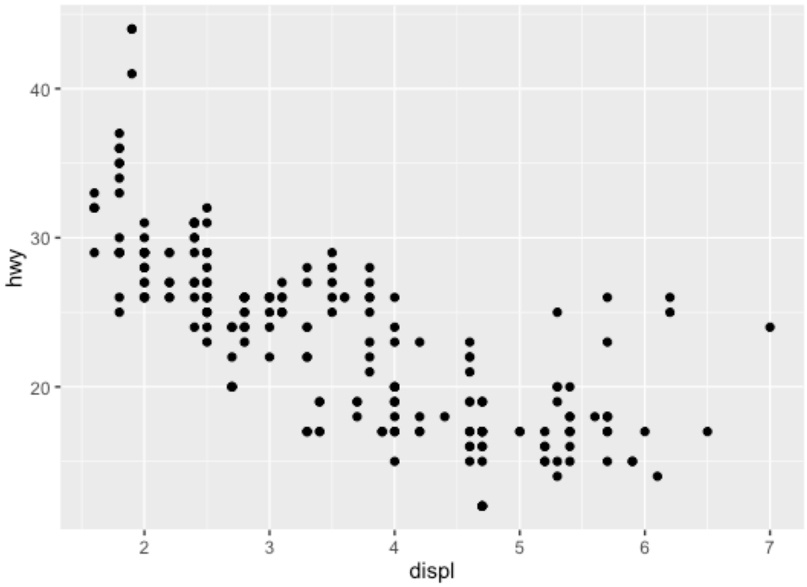
GEOMETRY

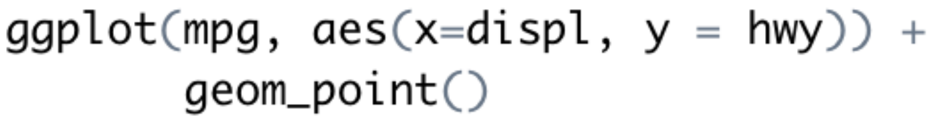
```
ggplot(mpg, aes(x=displ, y = hwy)) +
    geom_point()
```



Scatterplot structure

- Data: mpg
- Aesthetic Mapping
  - Engine size mapped to x position
  - Fuel economy to y position
- Layer: points
- Note
  - Data and aesthetics supplied in ggplot()
  - Layers added with +

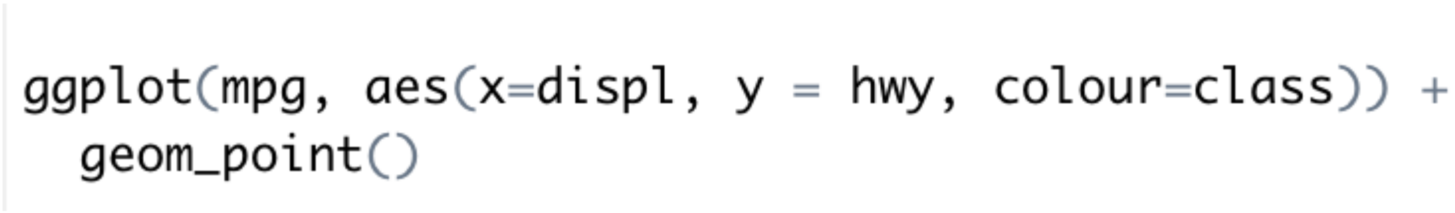




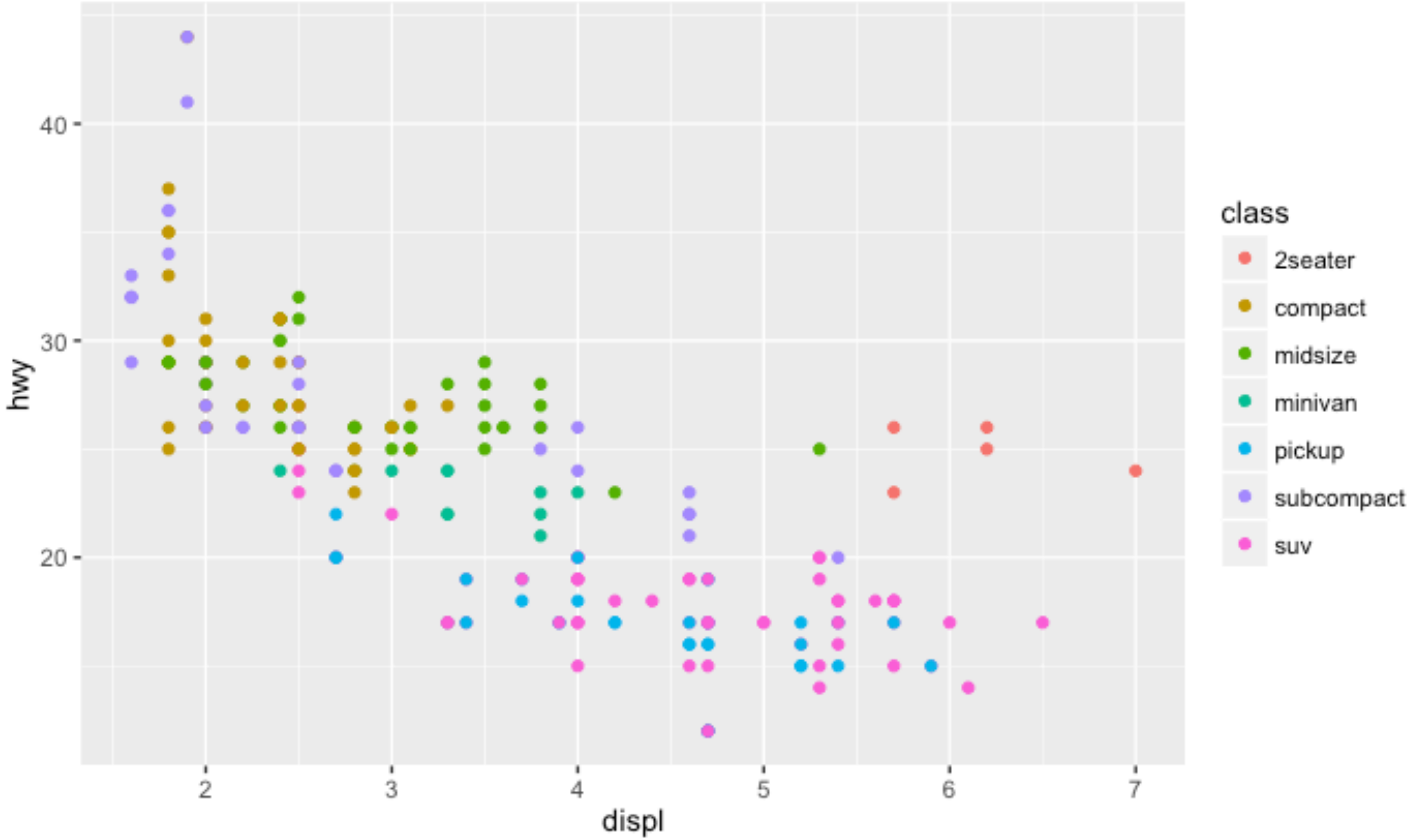
```
ggplot(mpg, aes(x=displ, y = hwy)) +
        geom_point()
```

Colour, Size and Shape

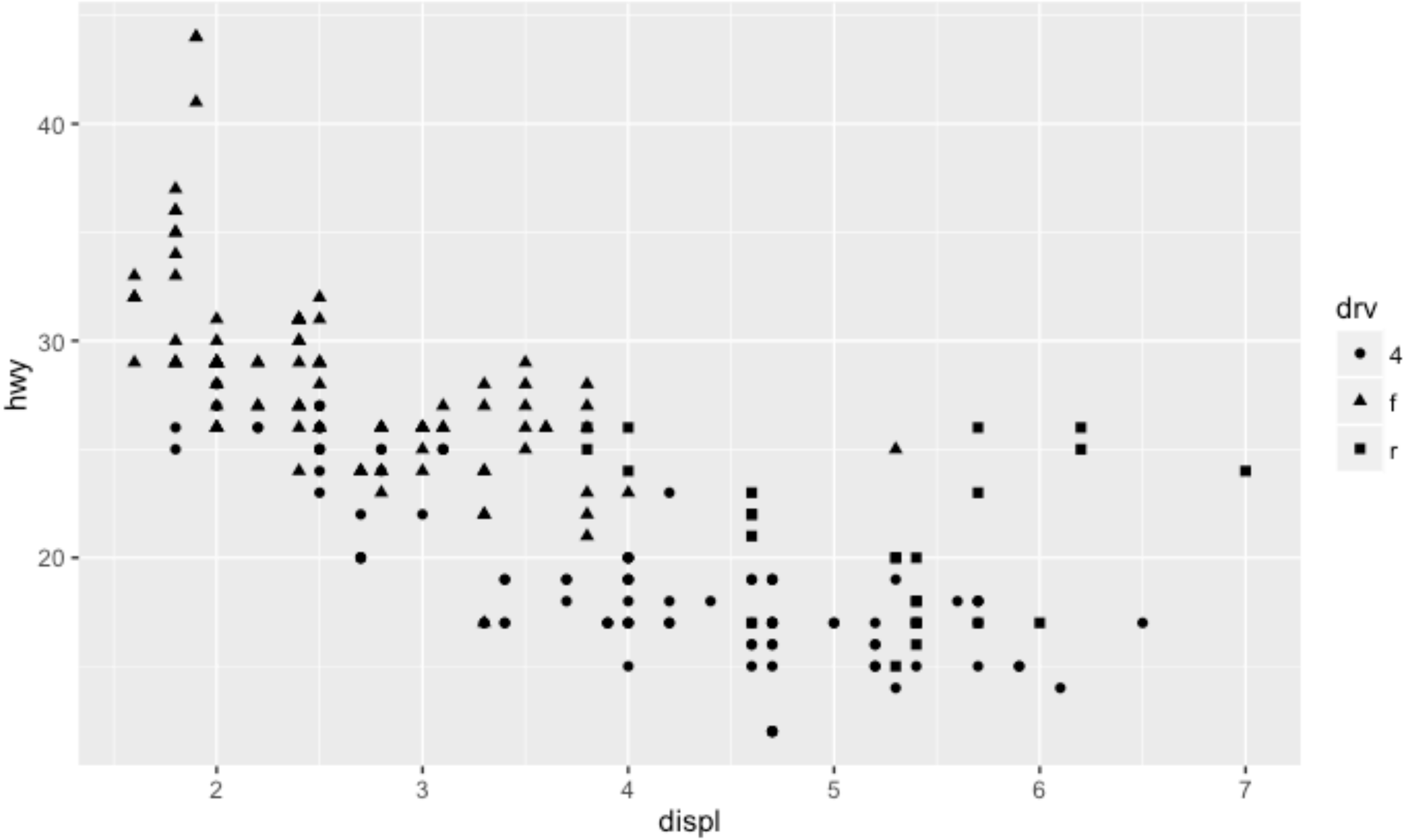
- To add additional variables to a plot, other aesthetics can be used
  - Colour
  - Shape
  - Size
- Work the same way as x and y aesthetics, and are added into the call to aes



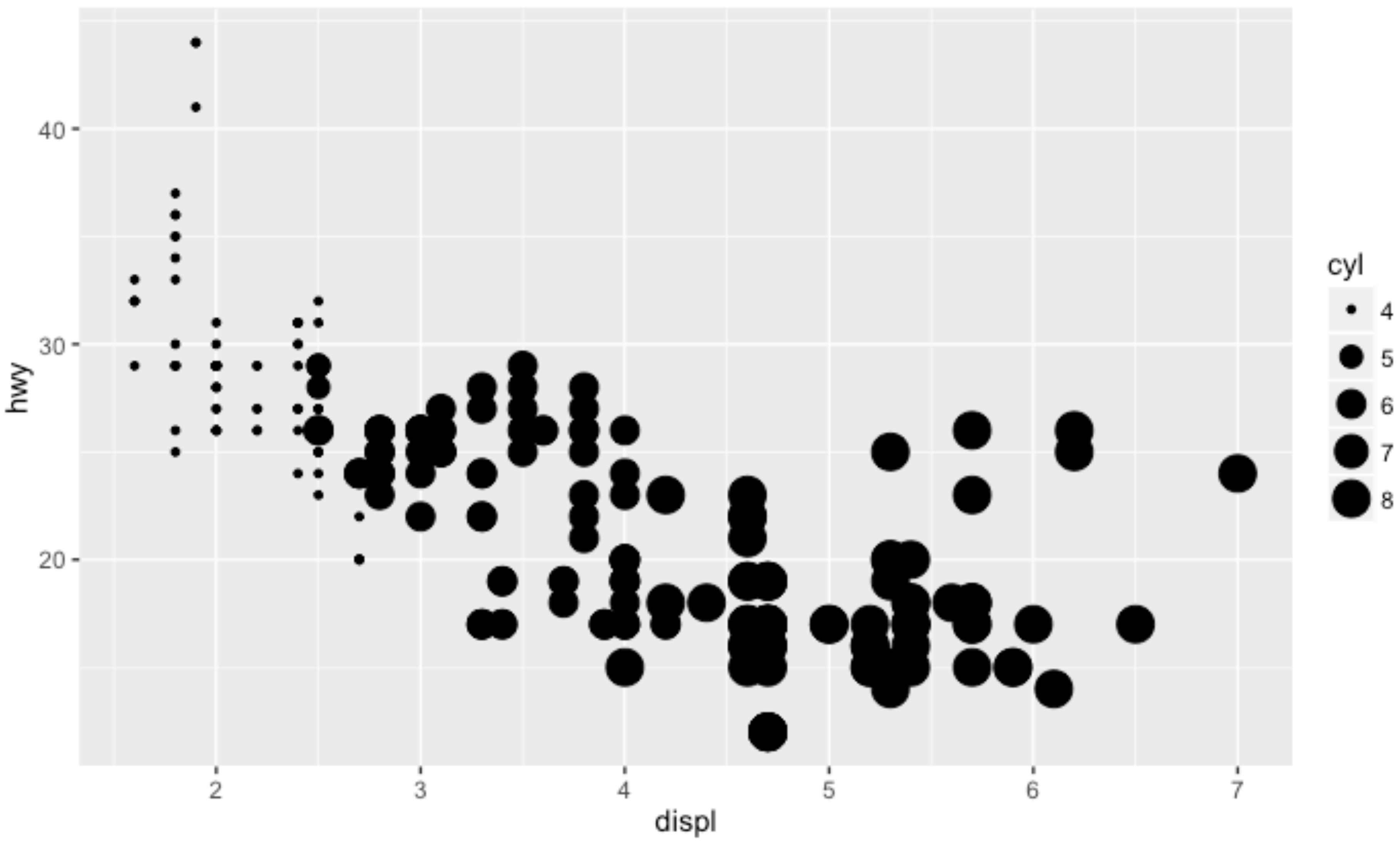
```
ggplot(mpg, aes(x=displ, y = hwy, colour=class)) +
  geom_point()
```



```
ggplot(mpg, aes(x=displ, y = hwy, shape=drv)) +
  geom_point()
```



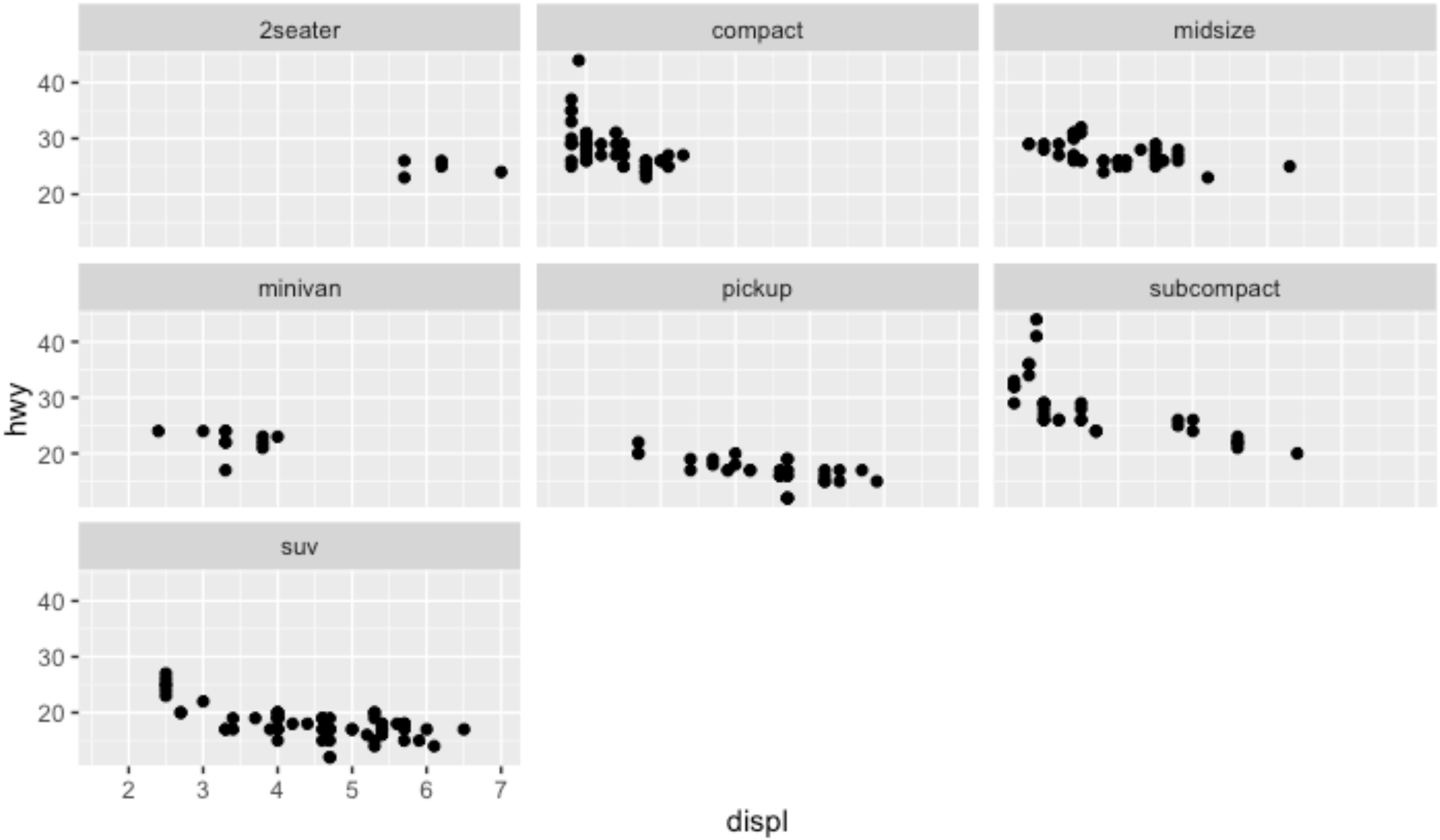
```
ggplot(mpg, aes(x=displ, y = hwy, size= cyl)) +
  geom_point()
```



Facetting

- An additional technique for displaying categorical variables on a plot
- Splits the data into subsets and displays the same graph for each subset
- facet_wrap() function with the name preceded by ~

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



Other plot geoms

Geom	Purpose
geom_smooth()	Fits a smoother to data and displays the smooth and its standard error
geom_boxplot()	Produces a box-and-whisker plot to summarise the distribution of a set of points

geom_histogram()
geom_freqpoly()
	Shows the distribution of continuous variables

geom_bar()	Shows the distribution of categorical variables

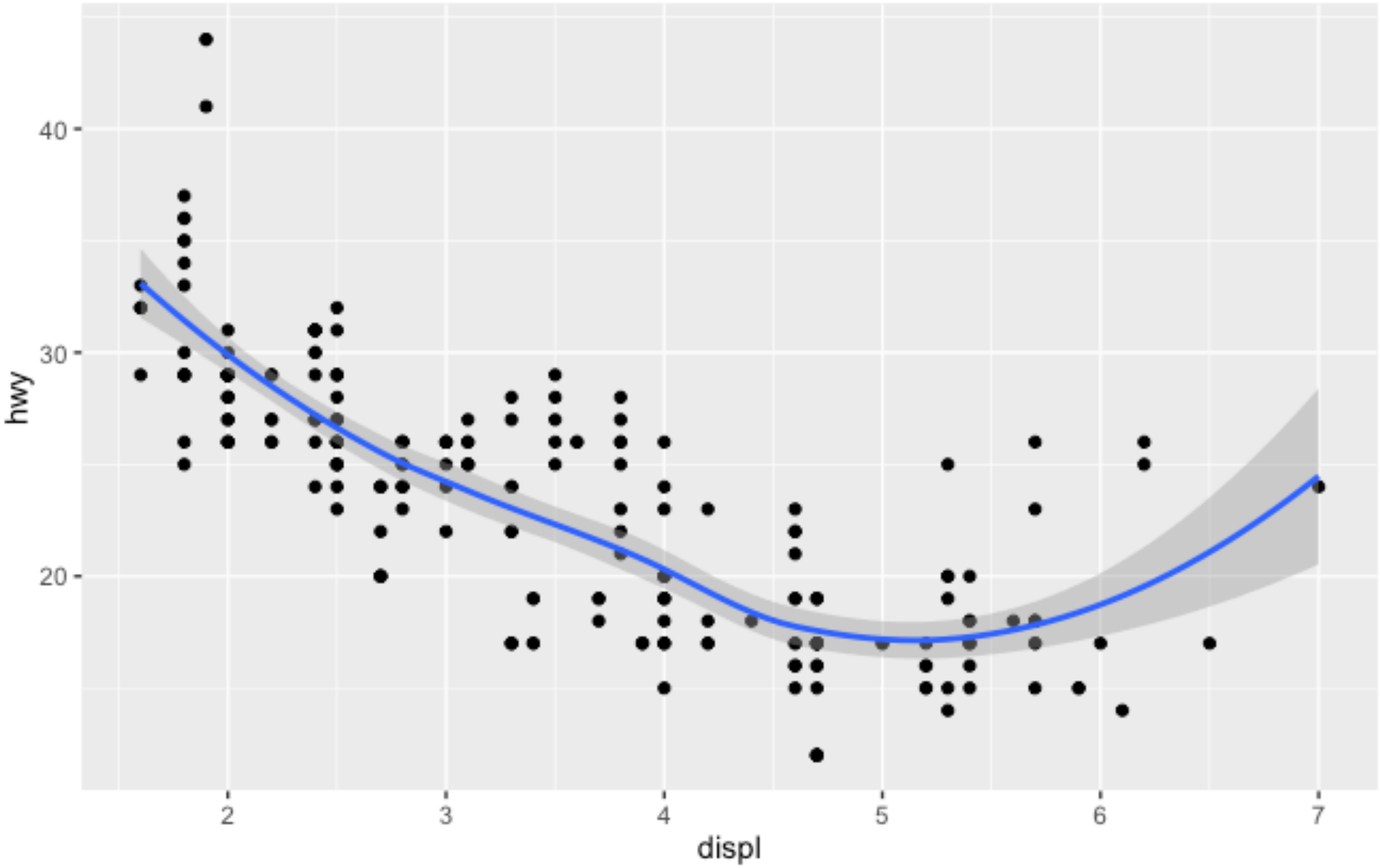
geom_path()
geom_line()
	Draws lines between data points

geom_area()	Draws an area plot, which is a line plot filled to the y-axis. Multiple groups will be stacked upon each other

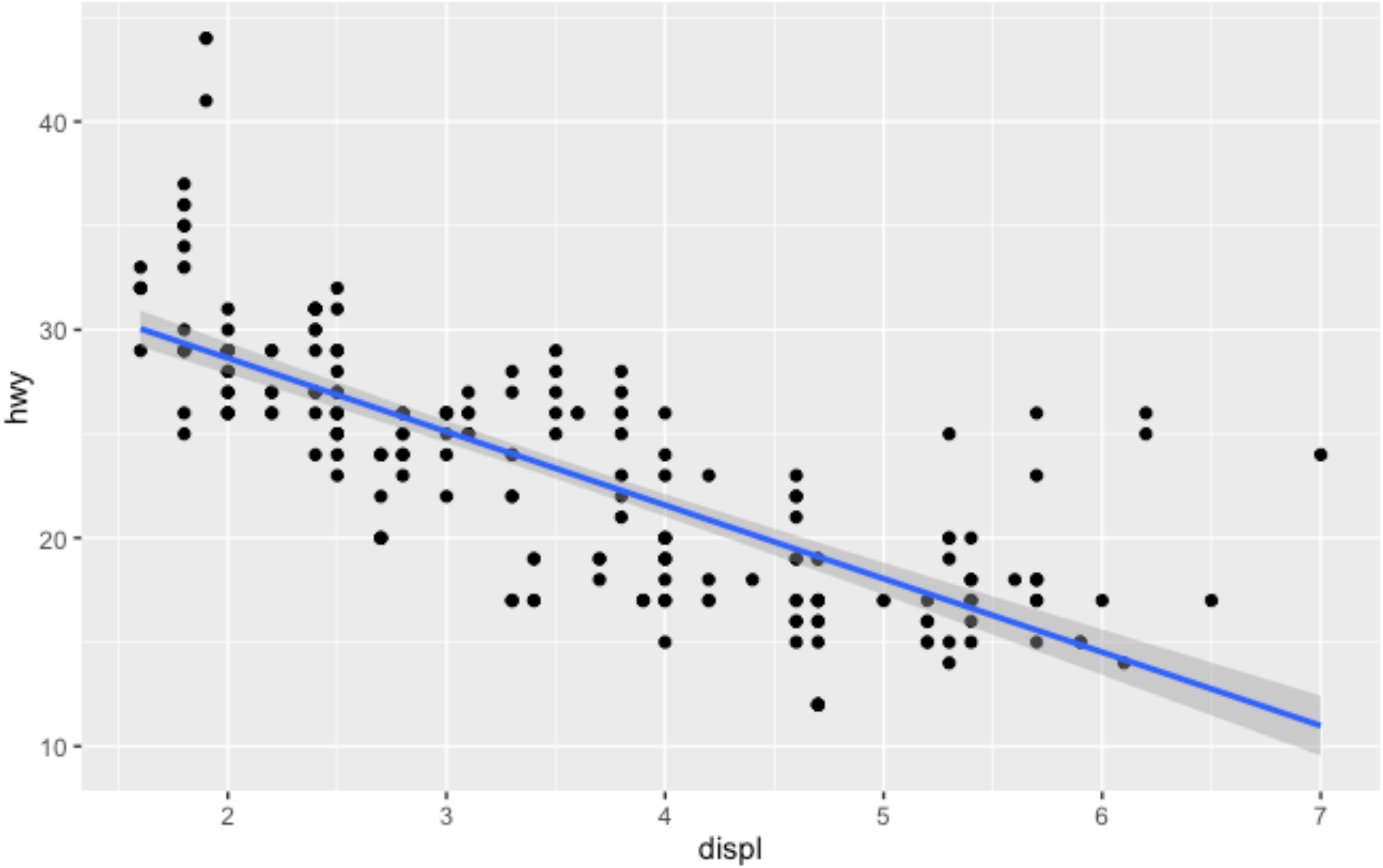
geom_rect()
geom_tile()
geom_raster()
	Draw rectangles

geom_polygon()	Draws polygons, which are filled paths.

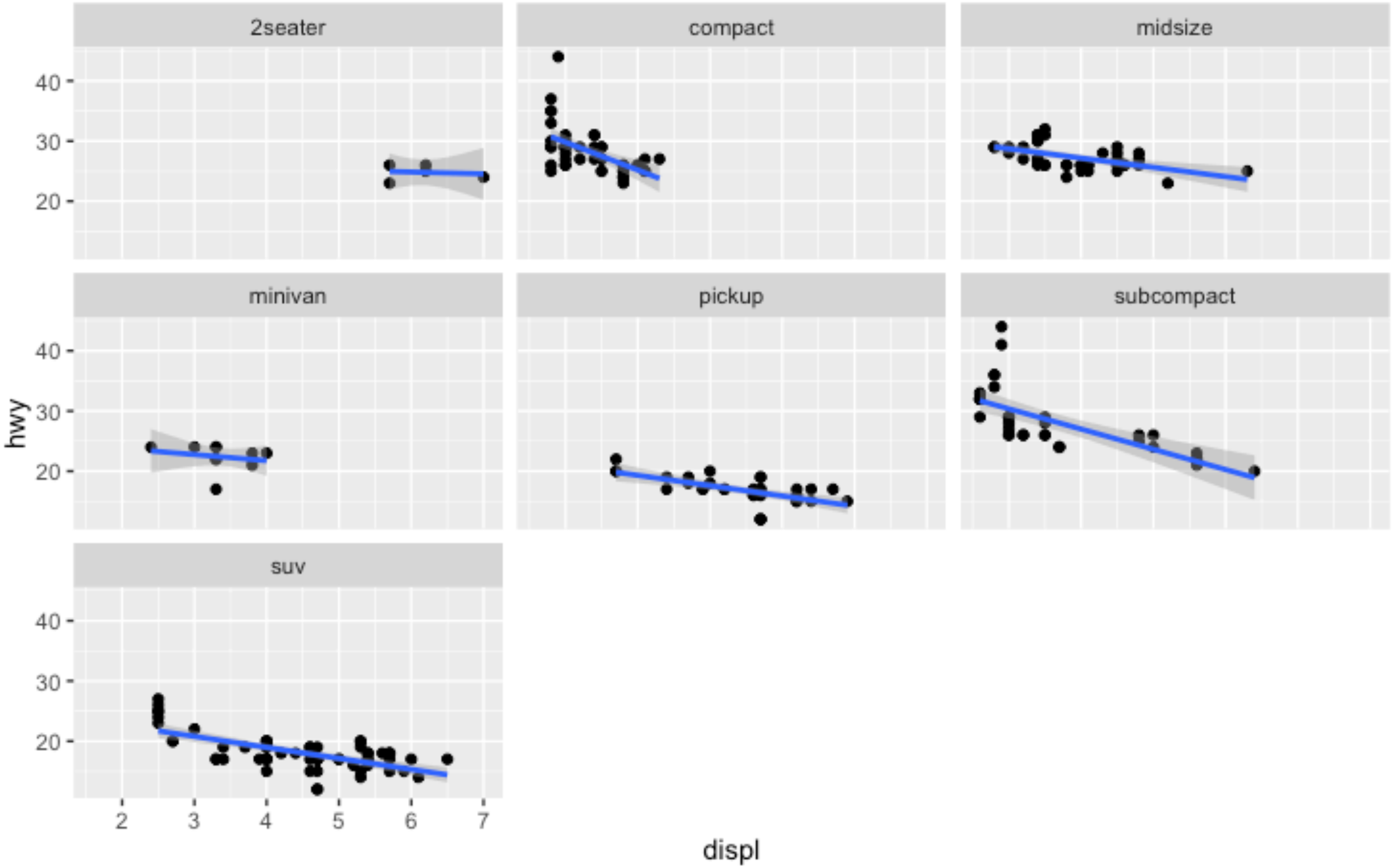
```
ggplot(mpg, aes(displ, hwy)) +
  geom_point() + geom_smooth()
```



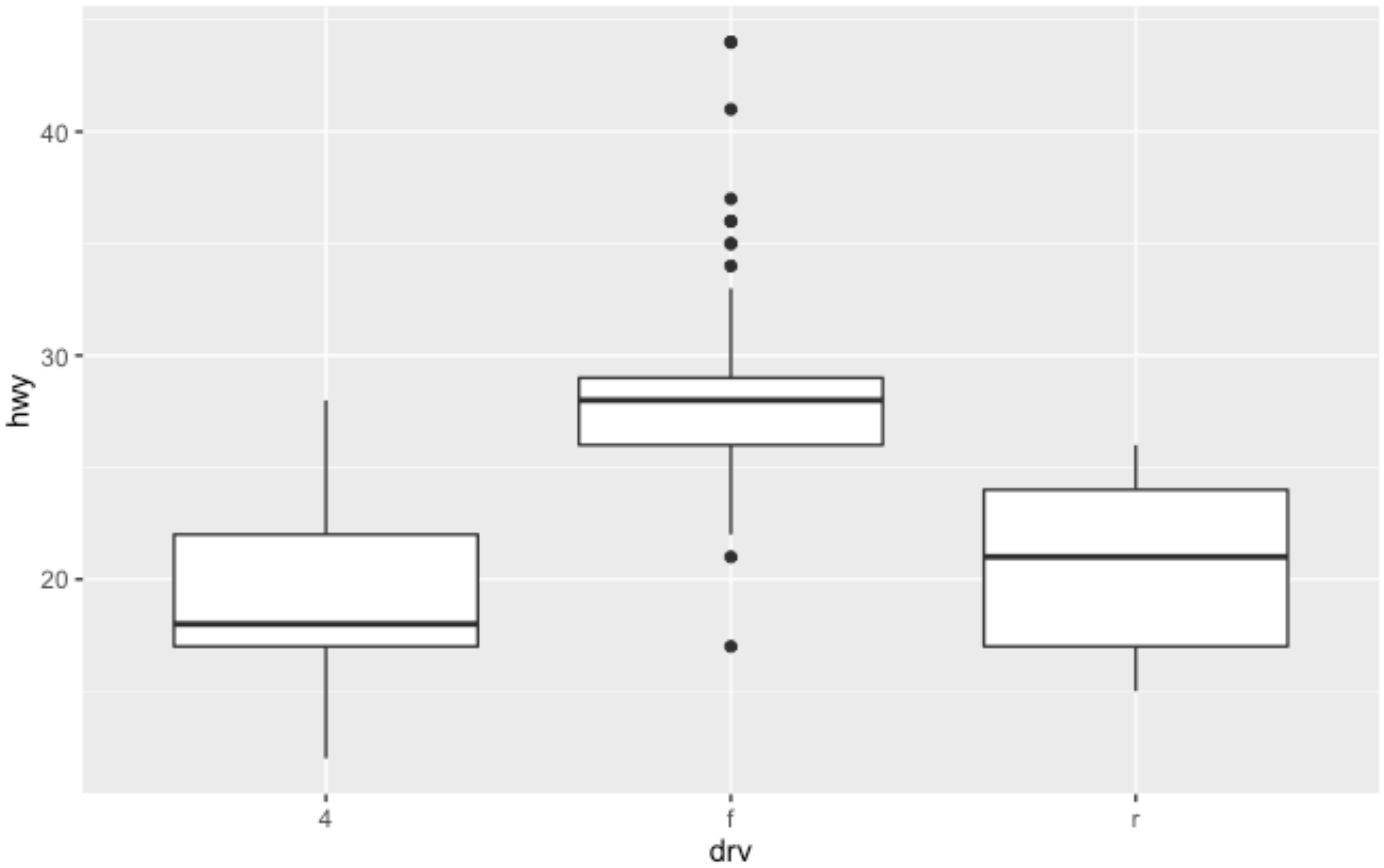
```
ggplot(mpg, aes(displ, hwy)) +
  geom_point() + geom_smooth(method="lm")
```



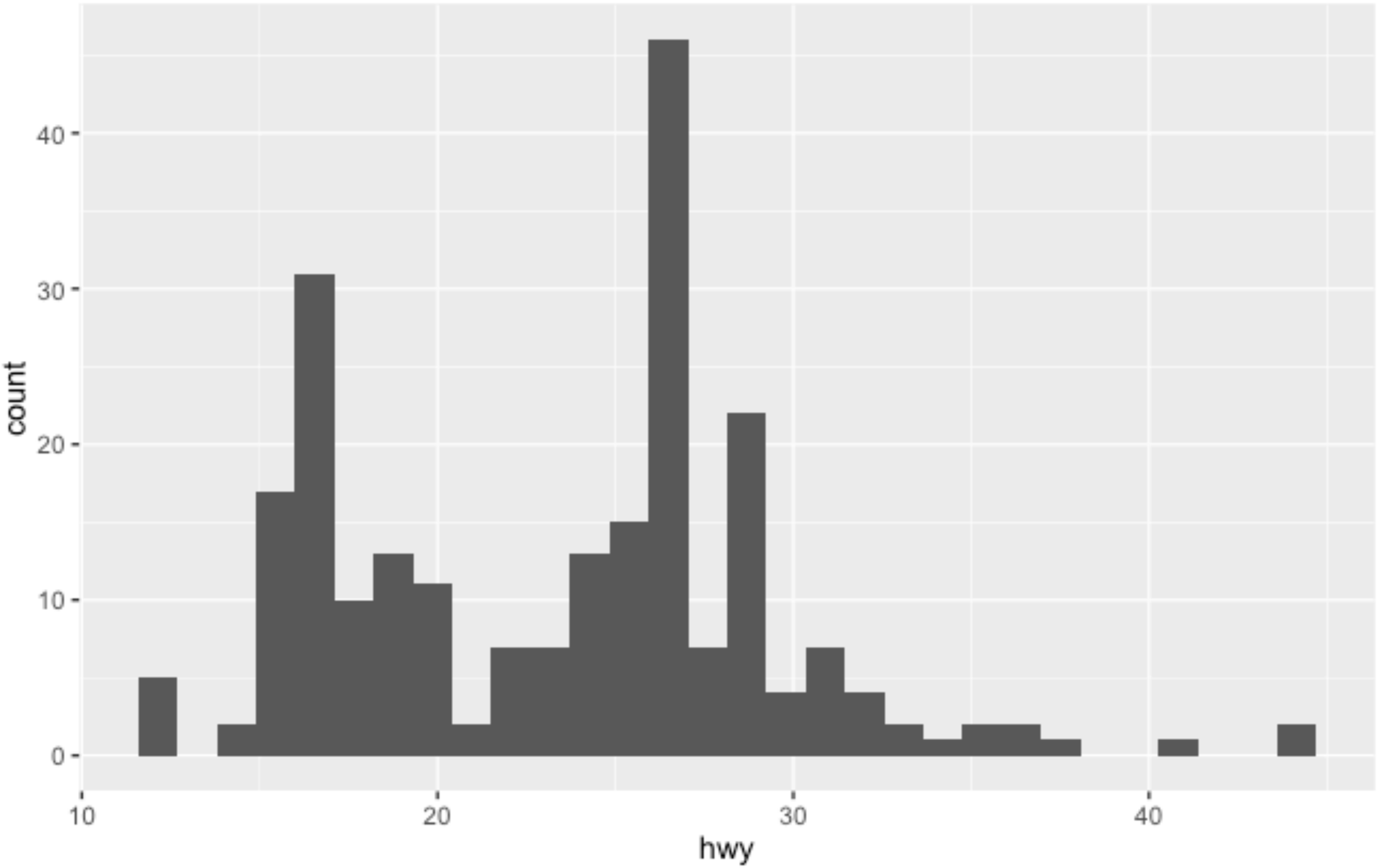
```
ggplot(mpg,aes(x=displ, y=hwy)) +
  geom_point() +facet_wrap(~class) + geom_smooth(method="lm")
```



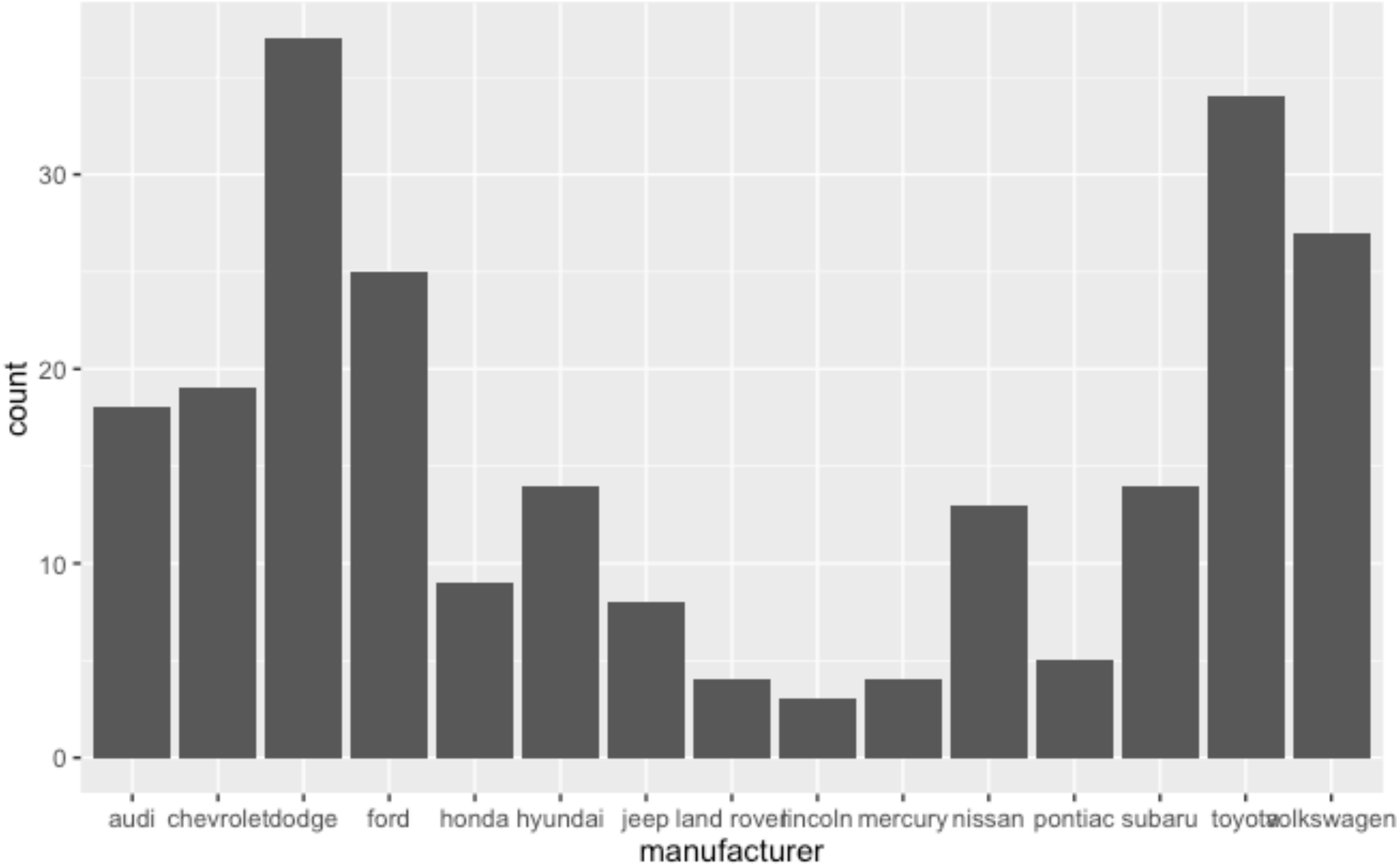
```
ggplot(mpg, aes(drv, hwy)) +
  geom_boxplot()
```



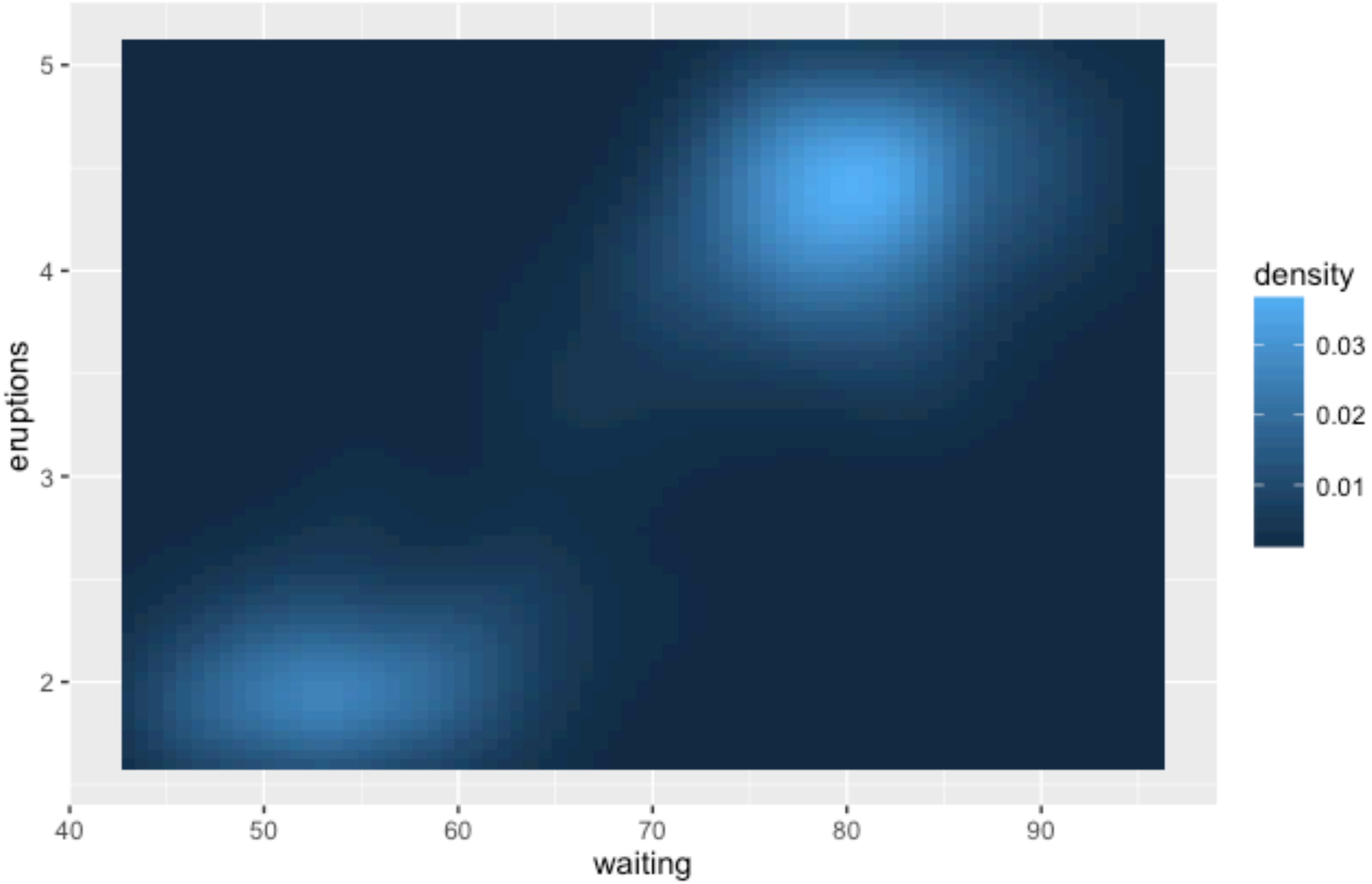
```
ggplot(mpg, aes(hwy)) +
  geom_histogram()
```



```
ggplot(mpg, aes(manufacturer)) +
  geom_bar()
```



```
ggplot(faithfuld,aes(waiting, eruptions))+
  geom_tile(aes(fill=density))
```

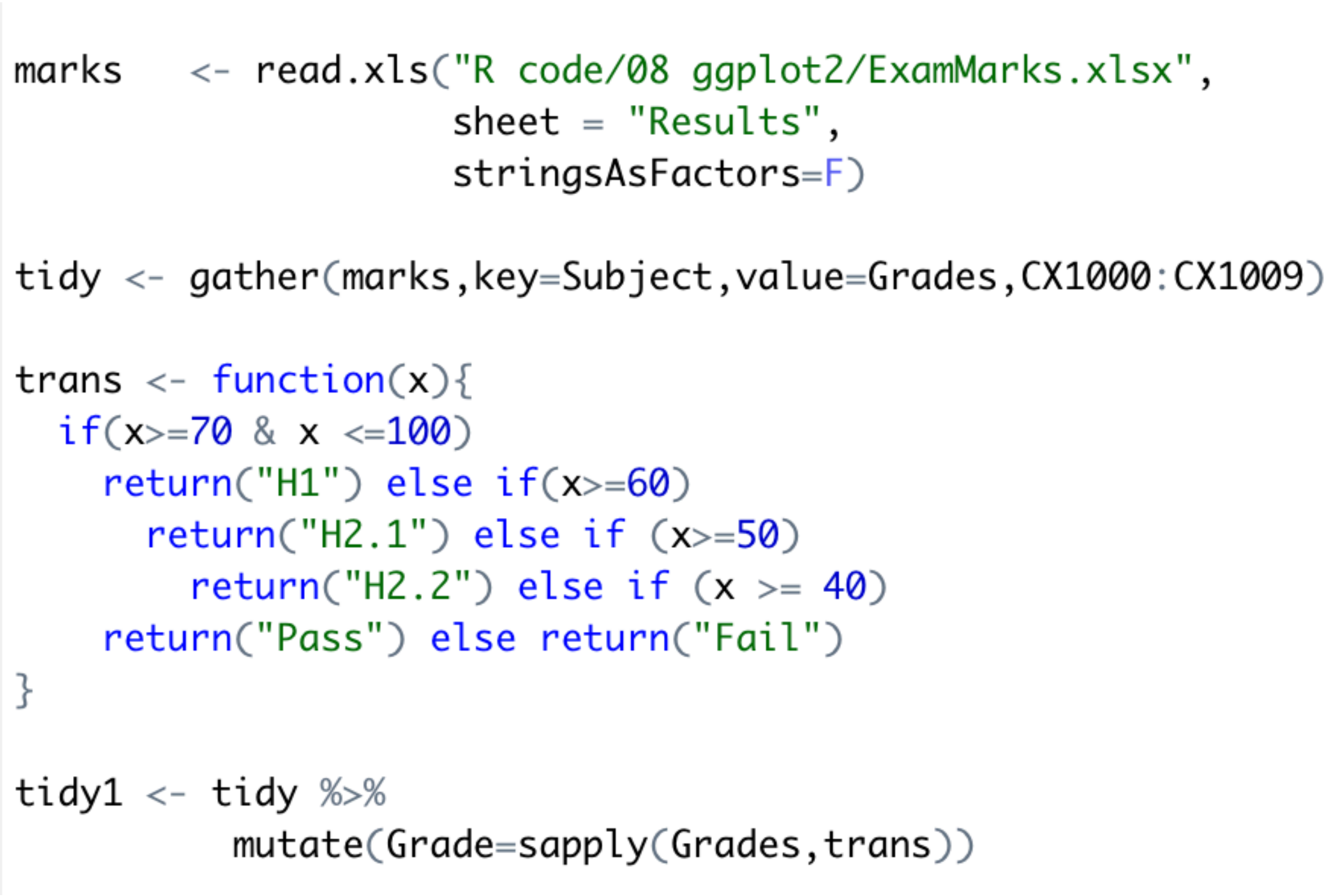


Example 1: Examination Grades

- Simulated data
- Ten subjects
- 50 Students
- Data in “untidy” format
- Process:
  - Read from Excel
  - Tidy using gather()
  - Visualise with ggplot

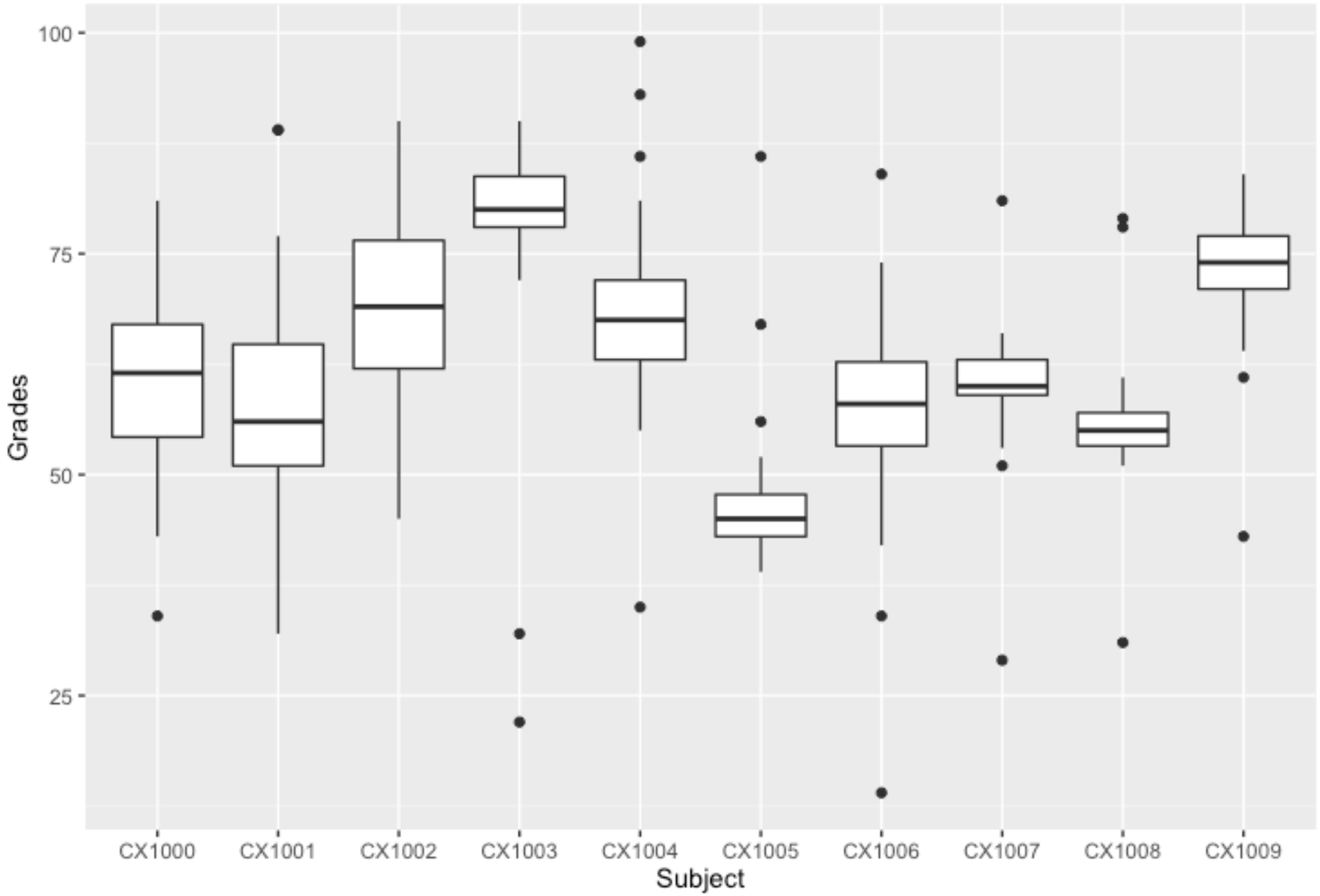
Student ID	CX1000	CX1001	CX1002	CX1003	CX1004	CX1005	CX1006	CX1007	CX1008	CX1009
1111111	56	51	78	85	63	45	55	59	52	76
1111112	56	64	68	80	70	39	46	60	55	74
1111113	52	61	63	81	71	49	54	61	54	76
1111114	50	42	72	81	63	44	62	59	56	68
1111115	67	53	77	84	65	52	63	62	52	71
1111116	45	57	62	32	61	56	62	51	55	79
1111117	67	58	54	77	75	44	58	62	57	77
1111118	69	50	66	78	72	39	60	58	57	84
1111119	70	56	62	80	71	52	60	63	54	70
1111120	51	52	46	82	74	42	66	63	55	73
1111121	71	89	90	72	99	86	67	81	79	79
1111122	66	62	80	85	67	49	60	59	54	77
1111123	62	56	75	88	70	46	54	57	57	72
1111124	61	77	62	79	70	43	71	59	61	79
1111125	72	56	48	78	57	45	56	63	53	75
1111126	67	56	68	79	63	41	42	64	56	70
1111127	64	67	74	84	69	44	48	61	55	70
1111128	77	56	66	82	59	44	61	61	54	64
1111129	64	52	66	72	64	45	84	60	51	61
1111130	67	65	70	79	67	44	54	56	55	73
1111131	55	38	79	84	66	44	58	63	51	74
1111132	73	41	52	82	55	42	65	59	55	79
1111133	59	60	85	76	62	47	65	64	55	72
1111134	65	67	79	90	69	43	53	63	57	78
1111135	46	54	46	78	58	41	54	59	58	77
1111136	56	47	75	81	62	47	55	53	53	76
1111137	79	69	53	88	74	44	69	60	56	71
1111138	76	41	75	80	93	47	55	56	53	79
1111139	55	66	78	80	57	42	56	60	55	82
1111140	64	67	70	78	65	45	59	59	59	66
1111141	54	56	80	90	74	47	34	63	55	70
1111142	58	45	66	81	80	49	46	58	51	78
1111143	65	44	60	76	63	39	74	62	56	76
1111144	43	32	45	22	35	67	14	29	31	43
1111145	62	51	74	80	64	52	45	57	53	79
1111146	71	52	70	88	72	43	70	63	55	72

Preparing the data

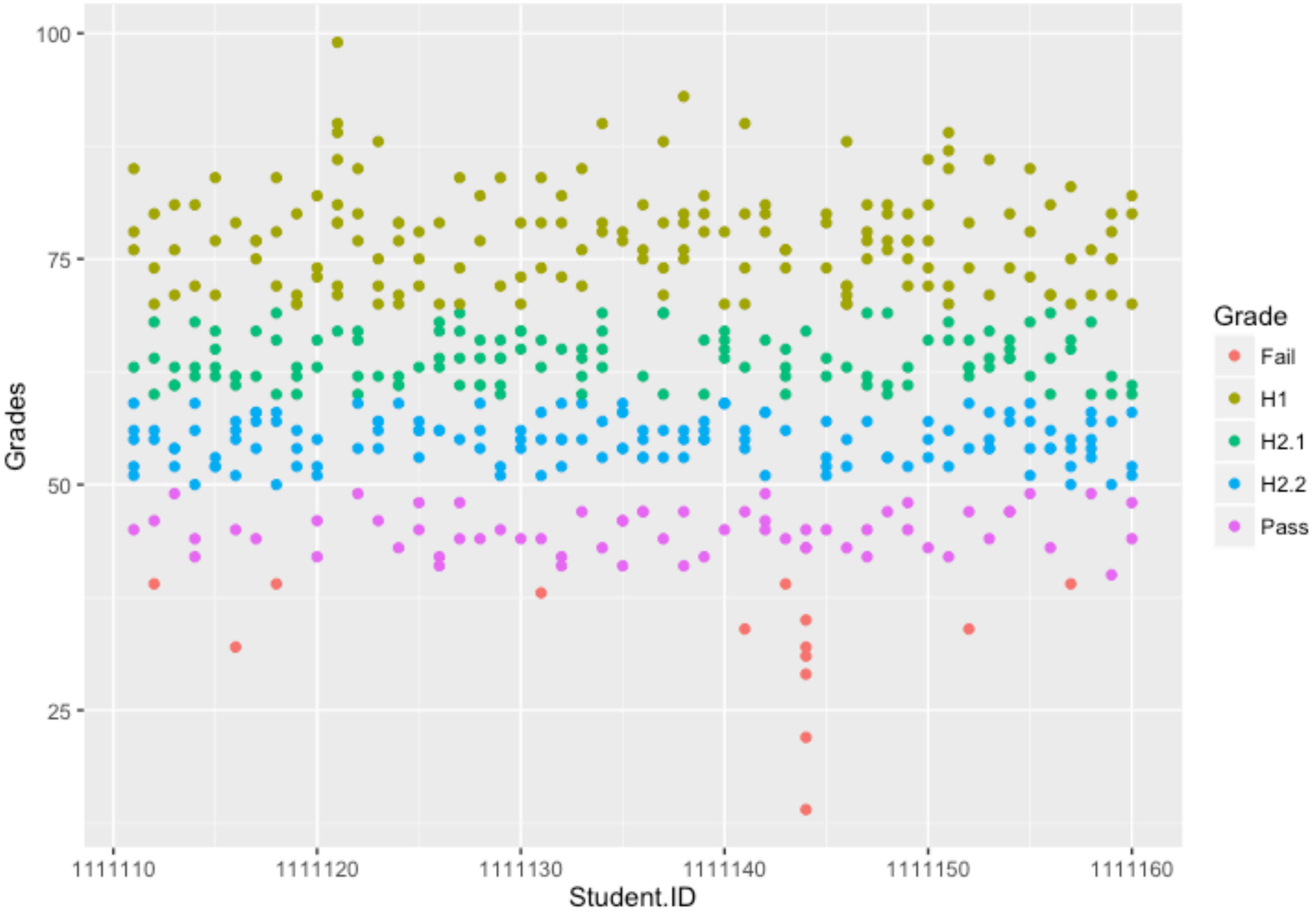


```
marks   <- read.xls("R code/08 ggplot2/ExamMarks.xlsx",
                    sheet = "Results",
                    stringsAsFactors=F)

tidy <- gather(marks,key=Subject,value=Grades,CX1000:CX1009)

trans <- function(x){
  if(x>=70 & x <=100)
    return("H1") else if(x>=60)
      return("H2.1") else if (x>=50)
        return("H2.2") else if (x >= 40)
    return("Pass") else return("Fail")
}

tidy1 <- tidy %>%
          mutate(Grade=sapply(Grades,trans))
```

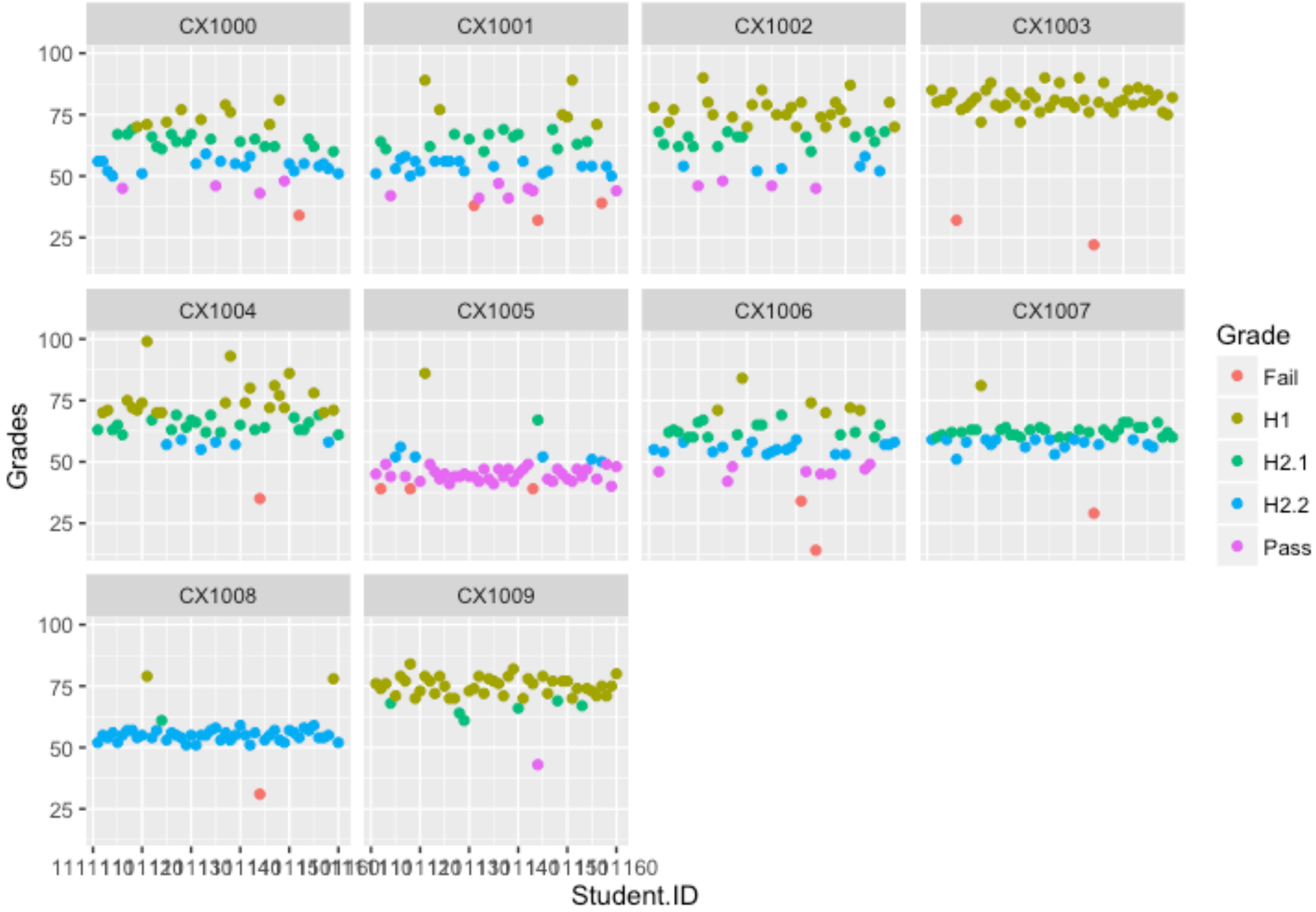
```
ggplot(tidy1, aes(x=Subject,y=Grades))+geom_boxplot()
```



```
ggplot(tidy1, aes(x=Student.ID,y=Grades,colour=Grade))+
  geom_point()
```



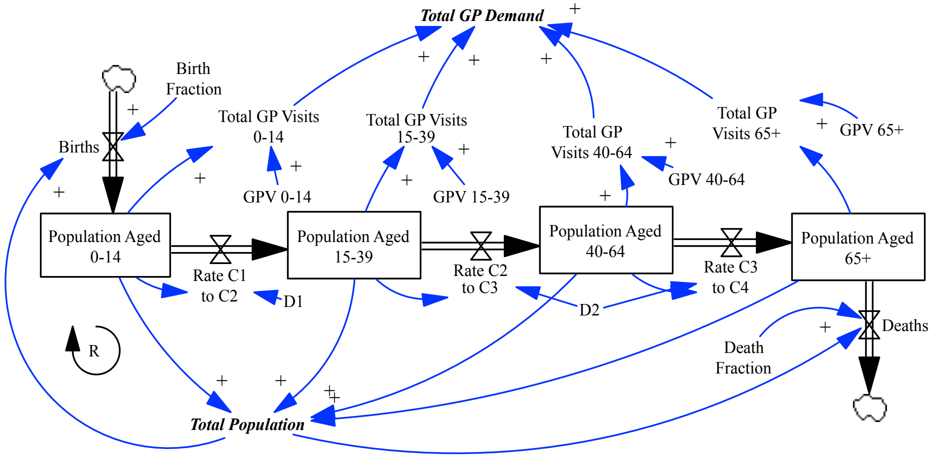
```
ggplot(tidy1,aes(x=Student.ID, y=Grades,colour=Grade)) +
  geom_point() +facet_wrap(~Subject)
```



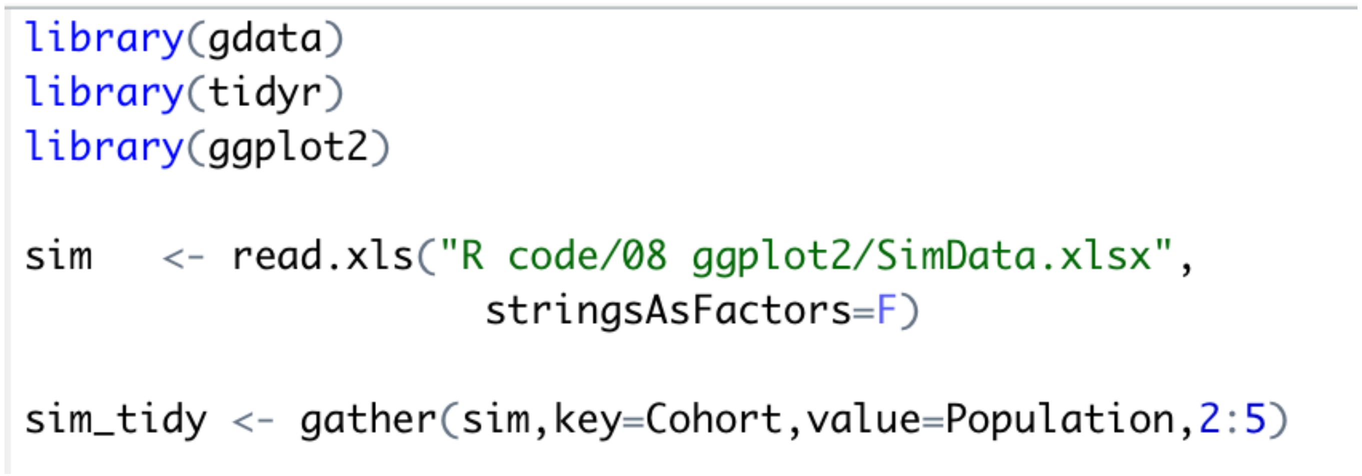
Example 2: Population Simulation

- Results from a simulation model
- Aging chain structure
  - 0-14
  - 15-39
  - 40-64
  - 64+
- Health system planning

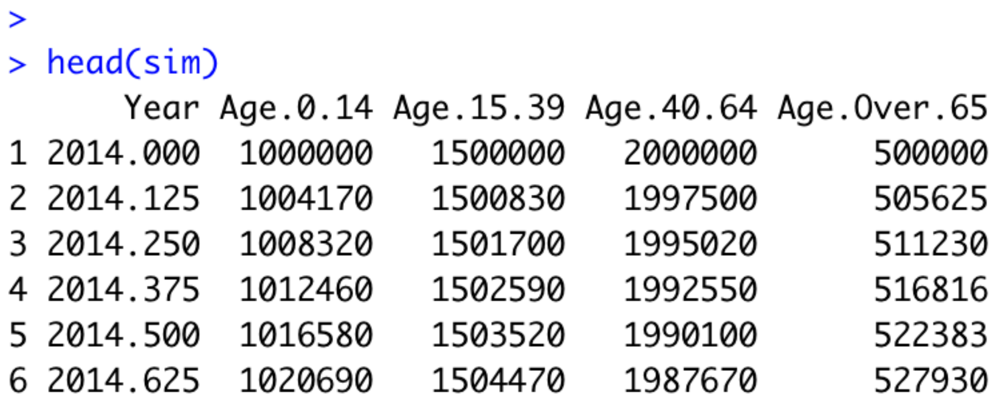
Year	Age 0-14	Age 15-39	Age 40-64	Age Over 65
2014.00	1000000	1500000	2000000	500000
2014.13	1004170	1500830	1997500	505625
2014.25	1008320	1501700	1995020	511230
2014.38	1012460	1502590	1992550	516816
2014.50	1016580	1503520	1990100	522383
2014.63	1020690	1504470	1987670	527930
2014.75	1024790	1505450	1985250	533457
2014.88	1028870	1506470	1982850	538966
2015.00	1032940	1507510	1980470	544455
2015.13	1036990	1508580	1978110	549925
2015.25	1041040	1509680	1975760	555376
2015.38	1045070	1510800	1973430	560808
2015.50	1049080	1511960	1971110	566222
2015.63	1053090	1513140	1968820	571616
2015.75	1057080	1514350	1966540	576992
2015.88	1061060	1515590	1964280	582349
2016.00	1065020	1516850	1962040	587688
2016.13	1068970	1518140	1959810	593008
2016.25	1072920	1519460	1957600	598309
2016.38	1076850	1520800	1955410	603592
2016.50	1080760	1522170	1953240	608858
2016.63	1084670	1523570	1951080	614104
2016.75	1088570	1524990	1948940	619333
2016.88	1092450	1526440	1946830	624544
2017.00	1096320	1527910	1944720	629736



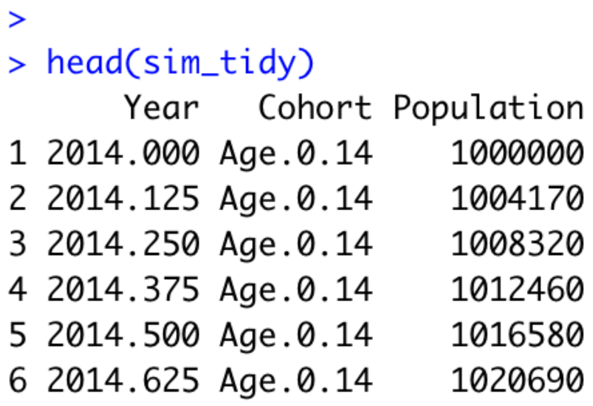
Convert to tidy data format



```r
library(gdata)
library(tidyr)
library(ggplot2)

sim    <- read.xls("R code/08 ggplot2/SimData.xlsx",
                           stringsAsFactors=F)

sim_tidy <- gather(sim,key=Cohort,value=Population,2:5)
```

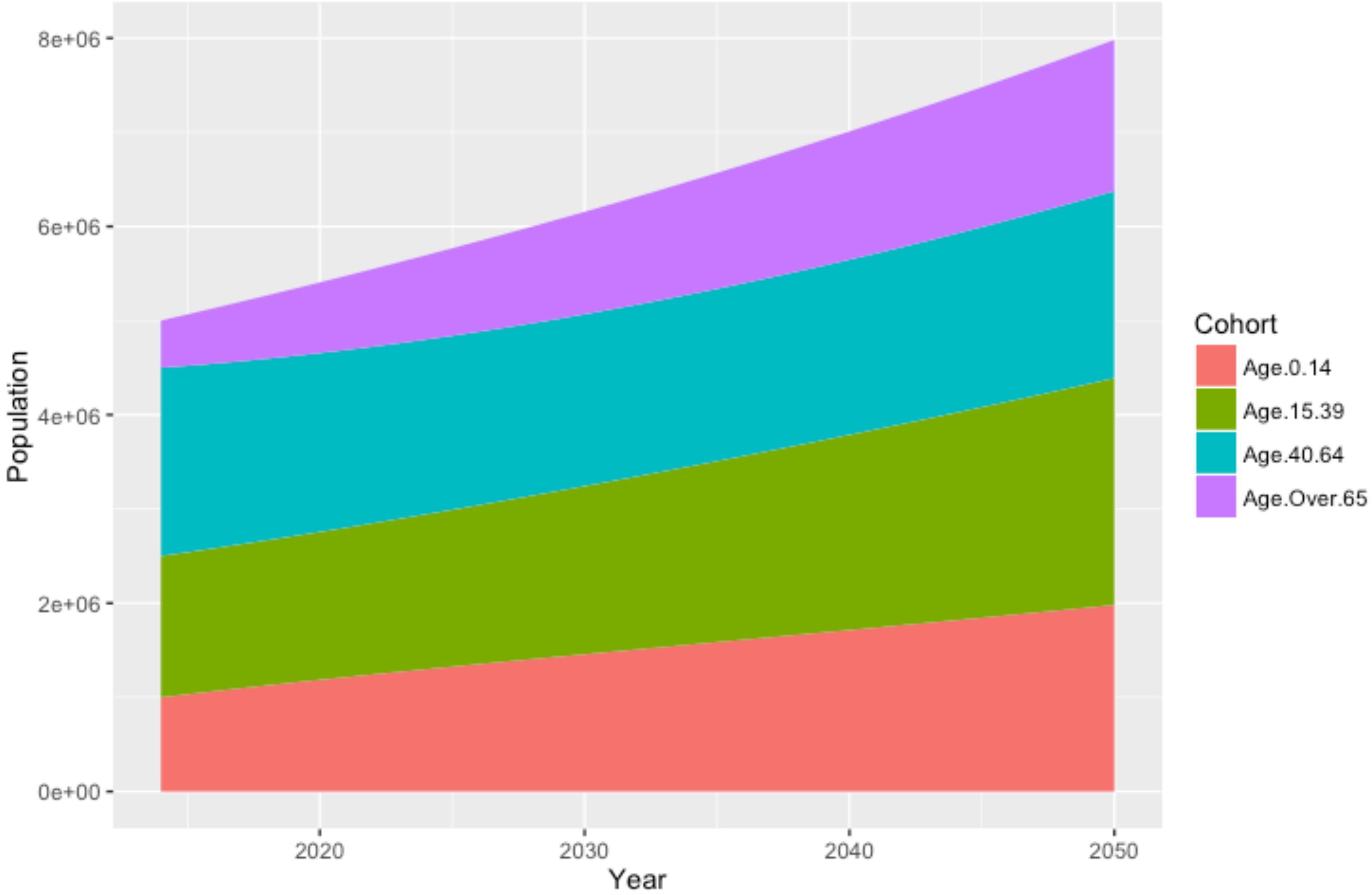


```
>
> head(sim)
      Year Age.0.14 Age.15.39 Age.40.64 Age.Over.65
1 2014.000  1000000   1500000   2000000      500000
2 2014.125  1004170   1500830   1997500      505625
3 2014.250  1008320   1501700   1995020      511230
4 2014.375  1012460   1502590   1992550      516816
5 2014.500  1016580   1503520   1990100      522383
6 2014.625  1020690   1504470   1987670      527930
```



```
>
> head(sim_tidy)
      Year   Cohort Population
1 2014.000 Age.0.14    1000000
2 2014.125 Age.0.14    1004170
3 2014.250 Age.0.14    1008320
4 2014.375 Age.0.14    1012460
5 2014.500 Age.0.14    1016580
6 2014.625 Age.0.14    1020690
```

```
ggplot(data=sim_tidy,aes(x=Year,y=Population,fill=Cohort)) +
  geom_area()
```

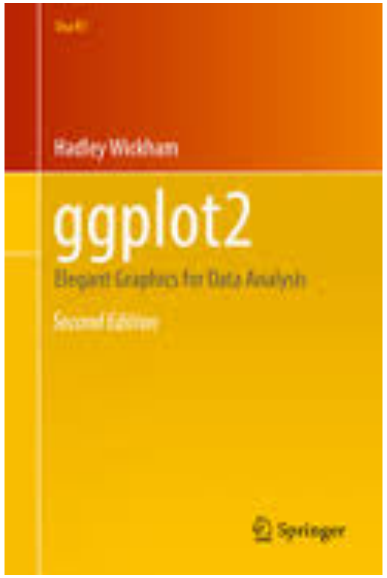


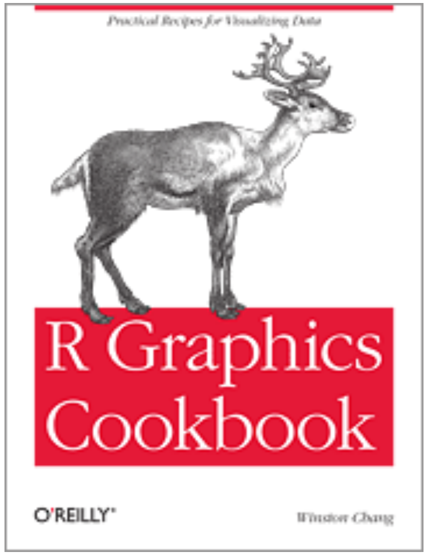
Additional Topics (Wickham 2016)

- Labels
- Scales, Axes and Legends
- Positioning
- Themes
- Data Analysis
- Data Transformation
- Modelling for Visualisation
- Programming with ggplot2

References

- Wickham, H. 2016. ggplot2: Elegant Graphics for Data Analysis. Springer





http://www.cookbook-r.com/Graphs/