Programming for Data Analytics

Lecture 5: Introduction to dplyr

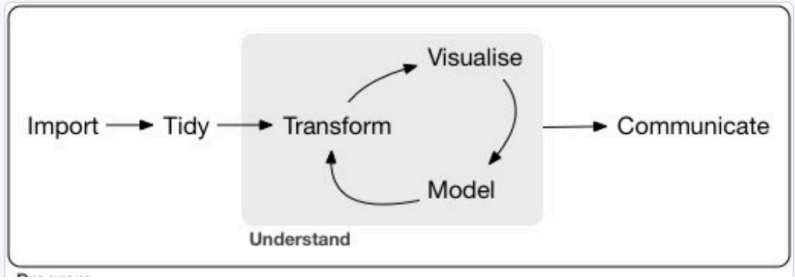
Dr. Jim Duggan,
School of Engineering & Informatics
National University of Ireland Galway.

https://twitter.com/_jimduggan

Course Overview

Lectures I-3	**R Fundamentals** *Atomic Vectors – Functions – Lists – Matrices – Data Frames*
Lectures 4-9	**Data Science with R** *ggplot2 – **dplyr** – tidyr – stringr – lubridate - purrr*
Lectures 10-11	**Advanced Programming with R** *Environments – Closures – S3 Object System*
Lectures 12	**Machine Learning with R – Case Studies** *Electricity Generation, Health*

Lecture Overview

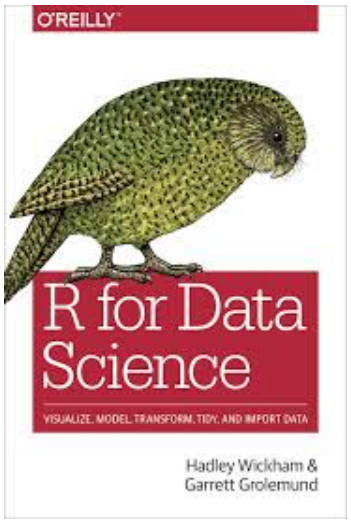


- Tibble
- dplyr
  - filter()
  - arrange()
  - select()
  - mutate()
  - summarise()

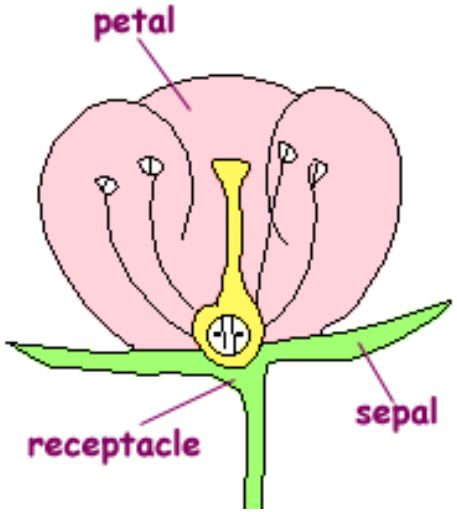
Lectures I-3	**R Fundamentals** *Atomic Vectors – Functions – Lists – Matrices – Data Frames*
Lectures 4-9	**Data Science with R** *ggplot2 – **dplyr** – tidyr – stringr – lubridate - purrr*
Lectures 10-11	**Advanced Programming with R** *Environments – Closures – S3 Object System*
Lectures 12	**Machine Learning with R – Case Studies** *Electricity Generation, Health*

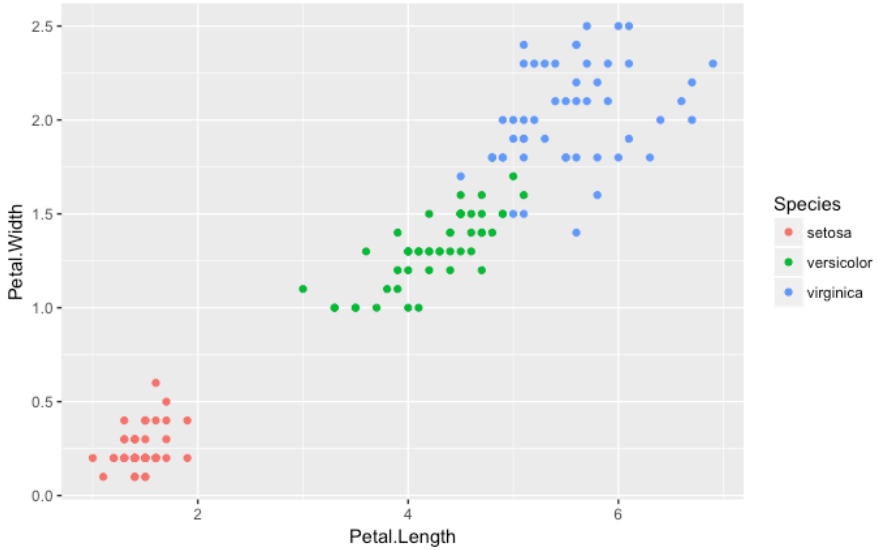
Overview

- Visualisation is an important tool for insight generation, but it's rare that you get the data in exactly the right form you need" (Wickham and Grolemund 2017)
  - Create new variables
  - Create summaries
  - Order data
- **dplyr** package is designed for data transformation



Data Set: Iris Data Set





Iris flower data set

From Wikipedia, the free encyclopedia

The ***Iris* flower data set** or **Fisher's *Iris* data set** is a multivariate data set introduced by the British statistician and biologist Ronald Fisher in his 1936 paper *The use of multiple measurements in taxonomic problems* as an example of linear discriminant analysis.[1] It is sometimes called **Anderson's *Iris* data set** because Edgar Anderson collected the data to quantify the morphologic variation of *Iris* flowers of three related species.[2] Two of the three species were collected in the Gaspé Peninsula "all from the same pasture, and picked on the same day and measured at the same time by the same person with the same apparatus".[3]

The data set consists of 50 samples from each of three species of *Iris* (*Iris setosa*, *Iris virginica* and *Iris versicolor*). Four features were measured from each sample: the length and the width of the sepals and petals, in centimetres. Based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other.

tibble

- "Tibbles are data frames, but they tweak some older behaviours to make life a little easier"
- One of the unifying features of the **tidyverse**
- To coerce a data frame to a tibble, use as_tibble()
- A tibble can be created from individual vectors using tibble()

```
> as_tibble(iris)
# A tibble: 150 × 5
   Sepal.Length Sepal.Width Petal.Length Petal.Width
          <dbl>       <dbl>        <dbl>       <dbl>
1           5.1         3.5          1.4         0.2
2           4.9         3.0          1.4         0.2
3           4.7         3.2          1.3         0.2
4           4.6         3.1          1.5         0.2
5           5.0         3.6          1.4         0.2
6           5.4         3.9          1.7         0.4
7           4.6         3.4          1.4         0.3
8           5.0         3.4          1.5         0.2
9           4.4         2.9          1.4         0.2
10          4.9         3.1          1.5         0.1
# ... with 140 more rows, and 1 more variables:
#   Species <fctr>
```

tibble v data.frame - Printing

- Printing: tibbles have a refined print method that only shows the first 10 row, and all columns that fit the screen
- Each column also reports its type

```
> as_tibble(iris)
# A tibble: 150 × 5
   Sepal.Length Sepal.Width
          <dbl>       <dbl>
1           5.1         3.5
2           4.9         3.0
3           4.7         3.2
4           4.6         3.1
5           5.0         3.6
6           5.4         3.9
7           4.6         3.4
8           5.0         3.4
9           4.4         2.9
10          4.9         3.1
# ... with 140 more rows, and 3 more
#   variables: Petal.Length <dbl>,
#   Petal.Width <dbl>, Species <fctr>
```

tibble v data.frame - Subsetting

- [], $ and [[work similar to a data frame
- Partial matching not supported

```
> t <- slice(as_tibble(iris),1:5)
>
> t
# A tibble: 5 × 5
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
         <dbl>       <dbl>        <dbl>       <dbl>  <fctr>
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
>
> t$Sepal.Length
[1] 5.1 4.9 4.7 4.6 5.0
>
> t[["Sepal.Length"]]
[1] 5.1 4.9 4.7 4.6 5.0
>
> t$Sepal.Lengt
NULL
Warning message:
Unknown column 'Sepal.Lengt'
```

tibble abbreviations

Abbreviation	Data Type
int	integers
dbl	doubles (real numbers)
chr	character vectors (strings
dttm	date-times
lgl	logical
fctr	factor (categorical variables with fixed possible values)
date	dates

Challenge 5.1
(p124 Wickham & Grolemund)

- Compare and contrast the following operations on a data.frame and equivalent tibble. What is different? Why might the default data frame behaviour cause you frustration?

```
df <- data.frame(abc=1, xyz="a")
df$x
df[, "xyz"]
df[, c("abc", "xyz")]
```

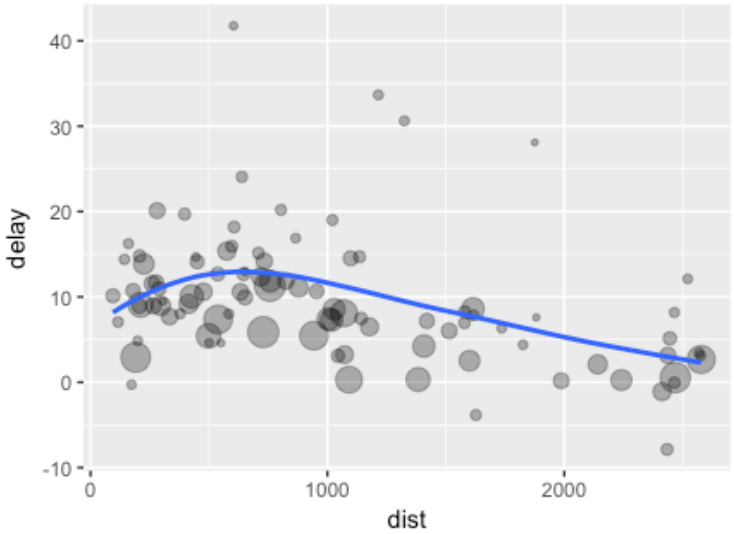
dplyr Basics: 5 key functions

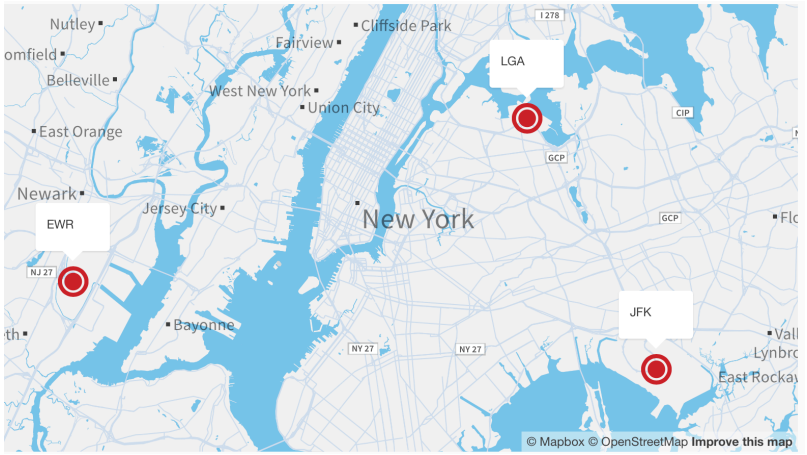
Function	Purpose
filter()	Pick observations by their values
arrange()	Reorder the rows
select()	Pick variables by their names
mutate()	Create new variables with functions of existing variables
summarise()	Collapse many values down to a single summary

- All verbs (functions) work similarly
  - The first argument is a data frame
  - The subsequent arguments decide what to do with the data frame
  - The result is a data frame (supports chaining of steps)

Data Set: nycflights13

```
> flights
# A tibble: 336,776 × 19
    year month   day dep_time sched_dep_time dep_delay
   <int> <int> <int>    <int>          <int>     <dbl>
1   2013     1     1      517            515         2
2   2013     1     1      533            529         4
3   2013     1     1      542            540         2
4   2013     1     1      544            545        -1
5   2013     1     1      554            600        -6
6   2013     1     1      554            558        -4
7   2013     1     1      555            600        -5
8   2013     1     1      557            600        -3
9   2013     1     1      557            600        -3
10  2013     1     1      558            600        -2
# ... with 336,766 more rows, and 13 more variables:
#   arr_time <int>, sched_arr_time <int>, arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
#   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
#   minute <dbl>, time_hour <dttm>
```





1. filter()

- Subset observations based on their values.
- First argument the name of the data frame
- Subsequent arguments are expressions that filter the data frame
- Only includes rows that have no missing values

```
> filter(flights,month==1,day==1)
# A tibble: 842 × 19
    year month   day dep_time sched_dep_time
   <int> <int> <int>    <int>          <int>
1   2013     1     1      517            515
2   2013     1     1      533            529
3   2013     1     1      542            540
4   2013     1     1      544            545
5   2013     1     1      554            600
6   2013     1     1      554            558
7   2013     1     1      555            600
8   2013     1     1      557            600
9   2013     1     1      557            600
10  2013     1     1      558            600
# ... with 832 more rows, and 14 more variables:
#   dep_delay <dbl>, arr_time <int>,
#   sched_arr_time <int>, arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>,
#   origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>,
#   time_hour <dttm>
```

Earliest Flight(s) that left all year?

```
> filter(flights,dep_time == min(dep_time,na.rm = T))
# A tibble: 25 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
1   2013     1    13        1           2249        72      108
2   2013     1    31        1           2100       181      124
3   2013    11    13        1           2359         2      442
4   2013    12    16        1           2359         2      447
5   2013    12    20        1           2359         2      430
6   2013    12    26        1           2359         2      437
7   2013    12    30        1           2359         2      441
8   2013     2    11        1           2100       181      111
9   2013     2    24        1           2245        76      121
10  2013     3     8        1           2355         6      431
# ... with 15 more rows, and 12 more variables:
#   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#   time_hour <dttm>
```

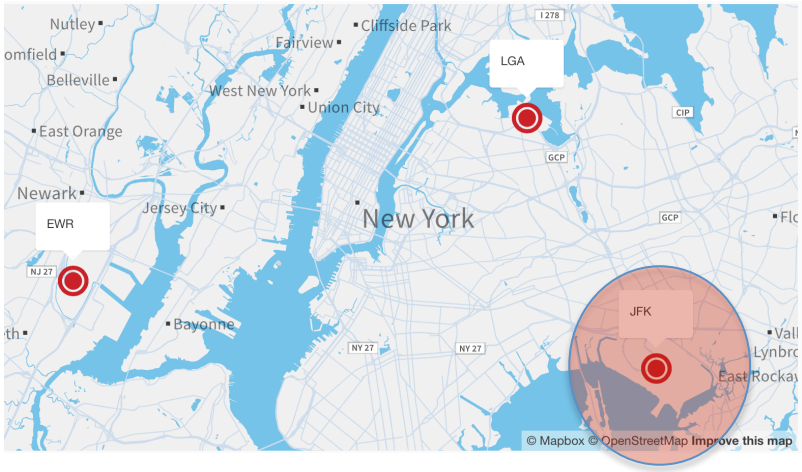
Flights with longest dep delay?

```
> filter(flights,dep_delay == max(dep_delay,na.rm = T))
# A tibble: 1 × 19
   year month   day dep_time sched_dep_time dep_delay arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>
1  2013     1     9      641            900      1301     1242
# ... with 12 more variables: sched_arr_time <int>,
#   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
#   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
> x<-filter(flights,dep_delay == max(dep_delay,na.rm = T))
```

More details…

```
> glimpse(x)
Observations: 1
Variables: 19
$ year           <int> 2013
$ month          <int> 1
$ day            <int> 9
$ dep_time       <int> 641
$ sched_dep_time <int> 900
$ dep_delay      <dbl> 1301
$ arr_time       <int> 1242
$ sched_arr_time <int> 1530
$ arr_delay      <dbl> 1272
$ carrier        <chr> "HA"
$ flight         <int> 51
$ tailnum        <chr> "N384HA"
$ origin         <chr> "JFK"
$ dest           <chr> "HNL"
$ air_time       <dbl> 640
$ distance       <dbl> 4983
$ hour           <dbl> 9
$ minute         <dbl> 0
$ time_hour      <dttm> 2013-01-09 09:00:00
```



Challenge 5.2

- Find all flights that:
  - Had an arrival delay of two or more hours
  - Flew to Houston (IAH or HOU)
  - Were operated by United, American or Delta
  - Departed in the summer (July, August and September)
  - Arrived more than 2 hours late, but didn't leave late
  - Departed between midnight and 6AM (inclusive)

2. arrange()

- Changes the order of rows.
- Takes a data frame and a set of column names to order by

```
> arrange(flights,dep_delay)
# A tibble: 336,776 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
1   2013    12     7     2040           2123       -43       40
2   2013     2     3     2022           2055       -33     2240
3   2013    11    10     1408           1440       -32     1549
```

Using desc()

```
> arrange(flights,desc(dep_delay))
# A tibble: 336,776 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
1   2013     1     9      641            900      1301     1242
2   2013     6    15     1432           1935      1137     1607
3   2013     1    10     1121           1635      1126     1239
4   2013     9    20     1139           1845      1014     1457
5   2013     7    22      845           1600      1005     1044
6   2013     4    10     1100           1900       960     1342
7   2013     3    17     2321            810       911      135
```

Challenge 5.3

- How would you use arrange() to sort all missing values to the start. (Hint: use is.na())
- Sort flights to find the fastest flights
- Which flights travelled the longest? Which traveled the shortest?

3. select()

- It is not uncommon to get datasets with hundreds, or even thousands, of variables
- A challenge is to narrow down on the variables of you're interested in
- select() allows you to rapidly zoom in on a useful subset using operations based on the variable names

```
> select(flights,year,month,day)
# A tibble: 336,776 × 3
    year month   day
   <int> <int> <int>
1   2013     1     1
2   2013     1     1
3   2013     1     1
4   2013     1     1
5   2013     1     1
6   2013     1     1
7   2013     1     1
8   2013     1     1
9   2013     1     1
10  2013     1     1
# ... with 336,766 more rows
```

Useful options with select()

```
> select(flights,year:day)
# A tibble: 336,776 × 3
    year month   day
   <int> <int> <int>
1   2013     1     1
2   2013     1     1
3   2013     1     1
4   2013     1     1
5   2013     1     1
6   2013     1     1
7   2013     1     1
8   2013     1     1
9   2013     1     1
10  2013     1     1
# ... with 336,766 more rows
```

```
> select(flights,-(month:minute))
# A tibble: 336,776 × 2
    year            time_hour
   <int>                <dttm>
1   2013 2013-01-01 05:00:00
2   2013 2013-01-01 05:00:00
3   2013 2013-01-01 05:00:00
4   2013 2013-01-01 05:00:00
5   2013 2013-01-01 06:00:00
6   2013 2013-01-01 05:00:00
7   2013 2013-01-01 06:00:00
8   2013 2013-01-01 06:00:00
9   2013 2013-01-01 06:00:00
10  2013 2013-01-01 06:00:00
# ... with 336,766 more rows
```

Special functions with select()

Special functions

As well as using existing functions like `:` and `c`, there are a number of special functions that only work inside `select`

- `starts_with(x, ignore.case = TRUE)`: names starts with `x`
- `ends_with(x, ignore.case = TRUE)`: names ends in `x`
- `contains(x, ignore.case = TRUE)`: selects all variables whose name contains `x`
- `matches(x, ignore.case = TRUE)`: selects all variables whose name matches the regular expression `x`
- `num_range("x", 1:5, width = 2)`: selects all variables (numerically) from x01 to x05.
- `one_of("x", "y", "z")`: selects variables provided in a character vector.
- `everything()`: selects all variables.

everything()

```
> select(flights,time_hour,everything())
# A tibble: 336,776 × 19
             time_hour  year month   day dep_time sched_dep_time
                 <dttm> <int> <int> <int>    <int>          <int>
1  2013-01-01 05:00:00  2013     1     1      517            515
2  2013-01-01 05:00:00  2013     1     1      533            529
3  2013-01-01 05:00:00  2013     1     1      542            540
4  2013-01-01 05:00:00  2013     1     1      544            545
5  2013-01-01 06:00:00  2013     1     1      554            600
6  2013-01-01 05:00:00  2013     1     1      554            558
7  2013-01-01 06:00:00  2013     1     1      555            600
8  2013-01-01 06:00:00  2013     1     1      557            600
9  2013-01-01 06:00:00  2013     1     1      557            600
10 2013-01-01 06:00:00  2013     1     1      558            600
# ... with 336,766 more rows, and 13 more variables:
#   dep_delay <dbl>, arr_time <int>, sched_arr_time <int>,
#   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
#   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>
```

4. mutate()

- It is often useful to add new columns that are functions of existing columns
- mutate() always adds new columns at the end of your data set.

```
sml <- select(flights,
              year:day,
              ends_with("delay"),
              distance,
              air_time)
```

Calculate any gain during flight…

```
> mutate(sml,gain=dep_delay-arr_delay)
# A tibble: 336,776 × 8
    year month   day dep_delay arr_delay distance air_time  gain
   <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl>
1   2013     1     1         2        11     1400      227    -9
2   2013     1     1         4        20     1416      227   -16
3   2013     1     1         2        33     1089      160   -31
4   2013     1     1        -1       -18     1576      183    17
5   2013     1     1        -6       -25      762      116    19
6   2013     1     1        -4        12      719      150   -16
7   2013     1     1        -5        19     1065      158   -24
8   2013     1     1        -3       -14      229       53    11
9   2013     1     1        -3        -8      944      140     5
10  2013     1     1        -2         8      733      138   -10
# ... with 336,766 more rows
```

Calculate the speed

```
> mutate(sml,speed=distance/air_time * 60)
# A tibble: 336,776 x 8
    year month   day dep_delay arr_delay distance air_time    speed
   <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl>    <dbl>
1   2013     1     1         2        11     1400      227 370.0441
2   2013     1     1         4        20     1416      227 374.2731
3   2013     1     1         2        33     1089      160 408.3750
4   2013     1     1        -1       -18     1576      183 516.7213
5   2013     1     1        -6       -25      762      116 394.1379
6   2013     1     1        -4        12      719      150 287.6000
7   2013     1     1        -5        19     1065      158 404.4304
8   2013     1     1        -3       -14      229       53 259.2453
9   2013     1     1        -3        -8      944      140 404.5714
10  2013     1     1        -2         8      733      138 318.6957
# ... with 336,766 more rows
```

Just keep the results… transmute()

```
> transmute(sml,gain=dep_delay-arr_delay,speed=distance/air_time * 60)
# A tibble: 336,776 × 2
    gain    speed
   <dbl>    <dbl>
1     -9 370.0441
2    -16 374.2731
3    -31 408.3750
4     17 516.7213
5     19 394.1379
6    -16 287.6000
7    -24 404.4304
8     11 259.2453
9      5 404.5714
10   -10 318.6957
# ... with 336,766 more rows
```

Useful Creation Functions

- There are many functions for creating new variables that can be used with mutate()
- The key property is that the function **must be vectorised:**
  - It must take a vector of values as input, and,
  - Return a vector with the same number of values as output
- Useful functions are now summarised

Useful Vectorised Functions

Grouping	Examples
Arithmetic Operators	+, -, *, /, ^
Modular Arithmetic	%/% - Integer division && - Remainder
Logs	log(), log2(), log10()
Offsets	lead() and lag() Find when values change x!=lag(x)
Cumulative and rolling aggregates	cumsum(), cumprod(), cummin(), cummax(), cummean()
Logical comparisons	<, <=, >, >=, !=
Ranking	min_rank()

5. summarise()

- The last key verb is summarise()
- It collapses a data frame into a single row
- Not very useful unless paired with group_by()
- Very useful to combine with the pipe operator

```
> summarise(flights,
+           AvrDelay=mean(dep_delay,na.rm=TRUE))
# A tibble: 1 × 1
  AvrDelay
     <dbl>
1 12.63907
```

group_by()

- Most data operations are useful done on groups defined by variables in the the dataset.
- The group_by function takes an existing tbl and converts it into a grouped tbl where operations are performed "by group".

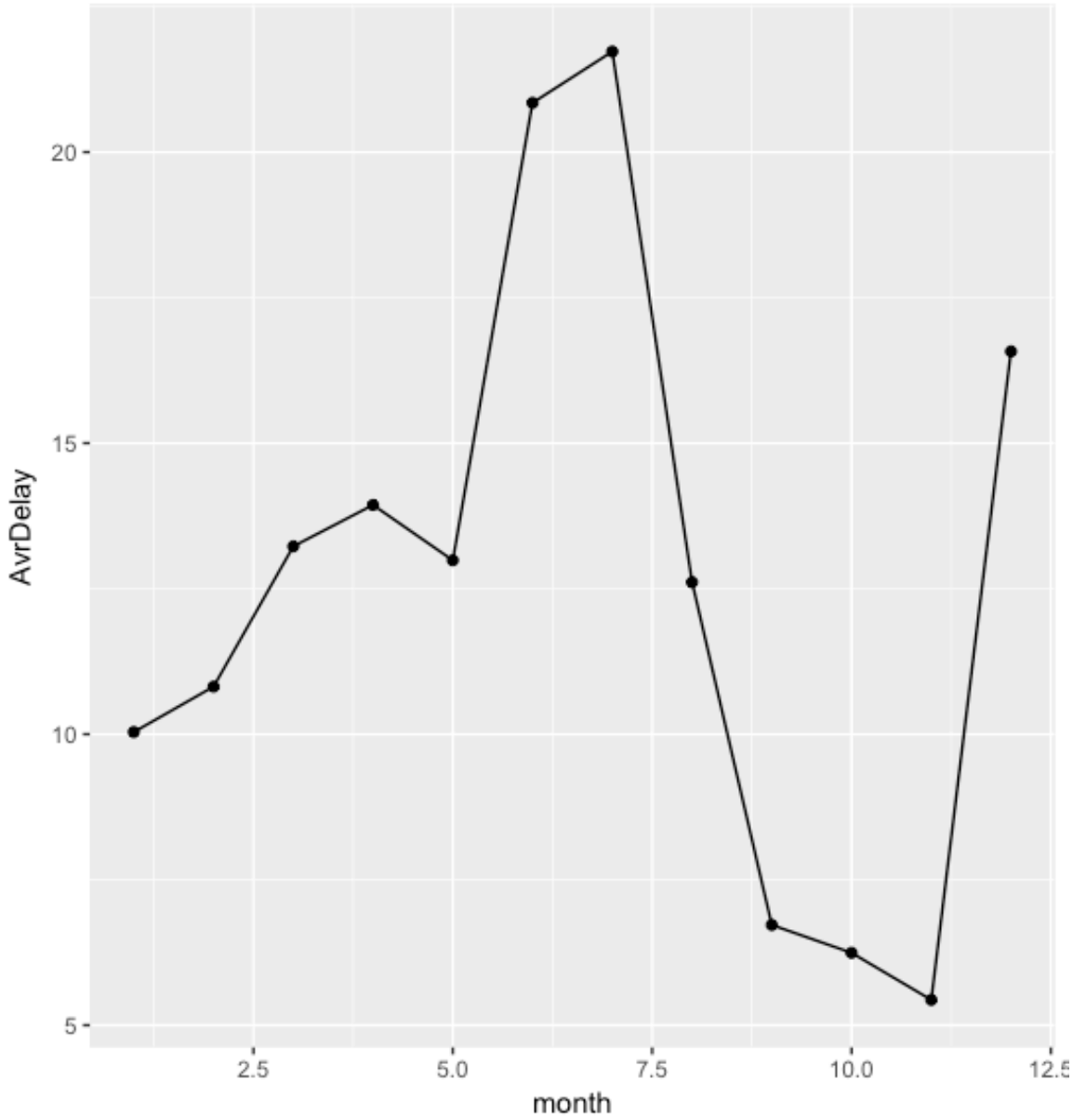
```
by_month <- group_by(flights,month)

ans <- summarise(by_month,
                 AvrDelay=mean(dep_delay,na.rm=T))
```

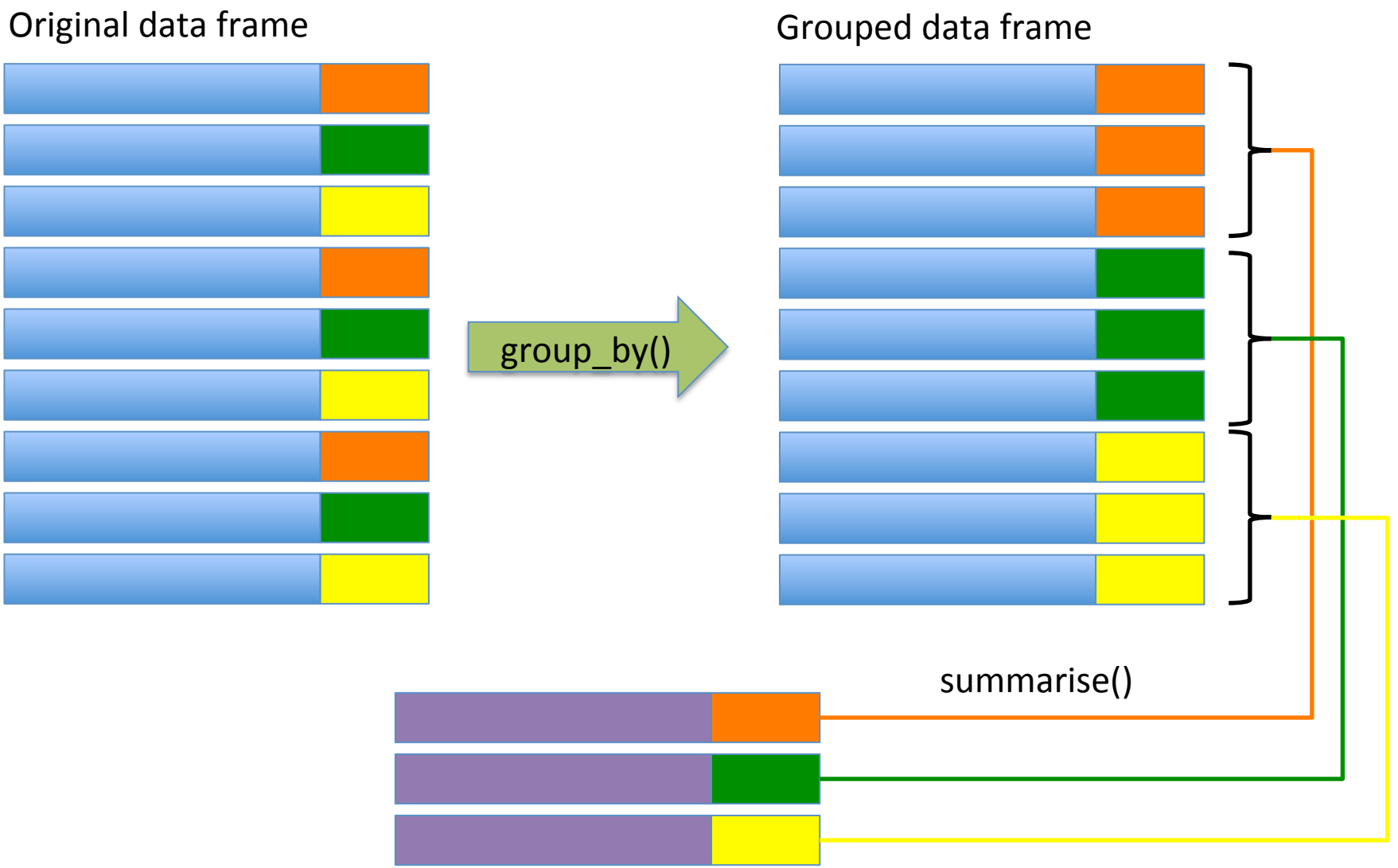
```
> by_month
Source: local data frame [336,776 x 19]
Groups: month [12]

    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
1   2013     1     1      517            515         2      830
2   2013     1     1      533            529         4      850
3   2013     1     1      542            540         2      923
4   2013     1     1      544            545        -1     1004
5   2013     1     1      554            600        -6      812
6   2013     1     1      554            558        -4      740
7   2013     1     1      555            600        -5      913
8   2013     1     1      557            600        -3      709
9   2013     1     1      557            600        -3      838
10  2013     1     1      558            600        -2      753
# ... with 336,766 more rows, and 12 more variables:
#   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#   time_hour <dttm>
```

```
> ans
# A tibble: 12 × 2
   month  AvrDelay
   <int>     <dbl>
1      1 10.036665
2      2 10.816843
3      3 13.227076
4      4 13.938038
5      5 12.986859
6      6 20.846332
7      7 21.727787
8      8 12.611040
9      9  6.722476
10    10  6.243988
11    11  5.435362
12    12 16.576688
```



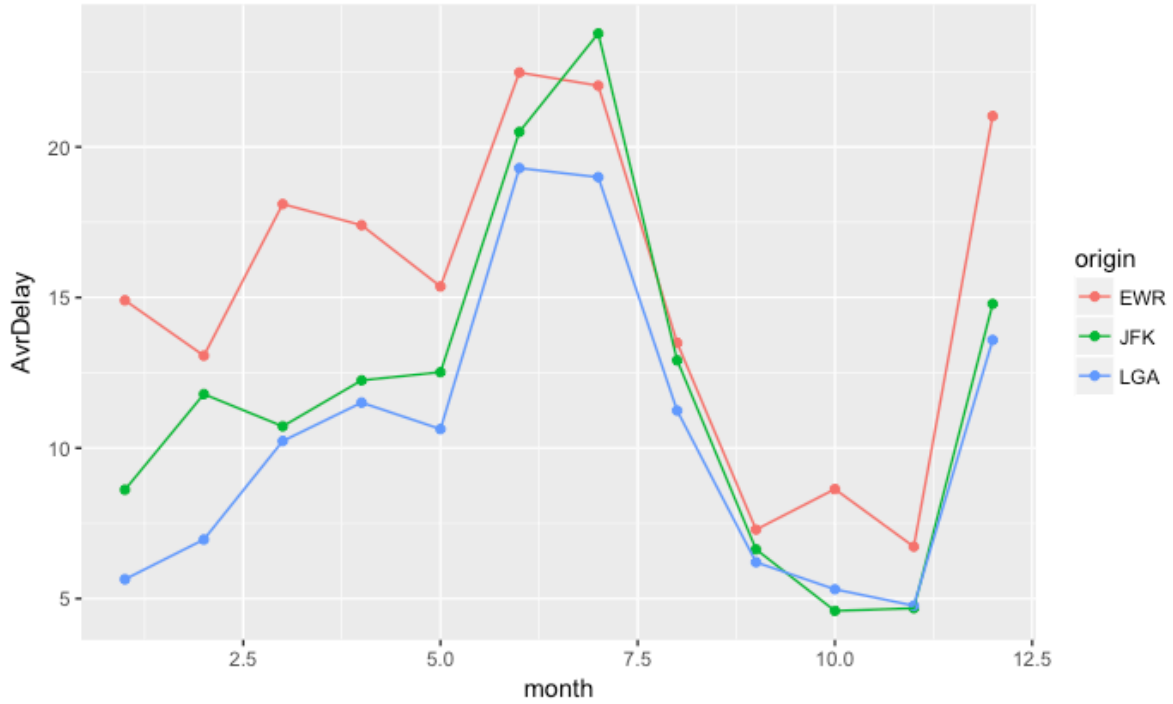
Overall idea...



```
by_month <- group_by(flights,month,origin)

ans <- summarise(by_month,AvrDelay=mean(dep_delay,na.rm=T))

ggplot(ans,mapping=aes(x=month,y=AvrDelay,colour=origin))+
        geom_point()+geom_path()
```



Combining operations with the Pipe

- The pipe %>% comes from the magrittr package (Stefan Milton Bache)
- Helps to write code that is easier to read and understand
- x %>% f(y) turns into f(x, y)
- x %>% f(y) %>% g(z) turns into g(f(x, y),z)

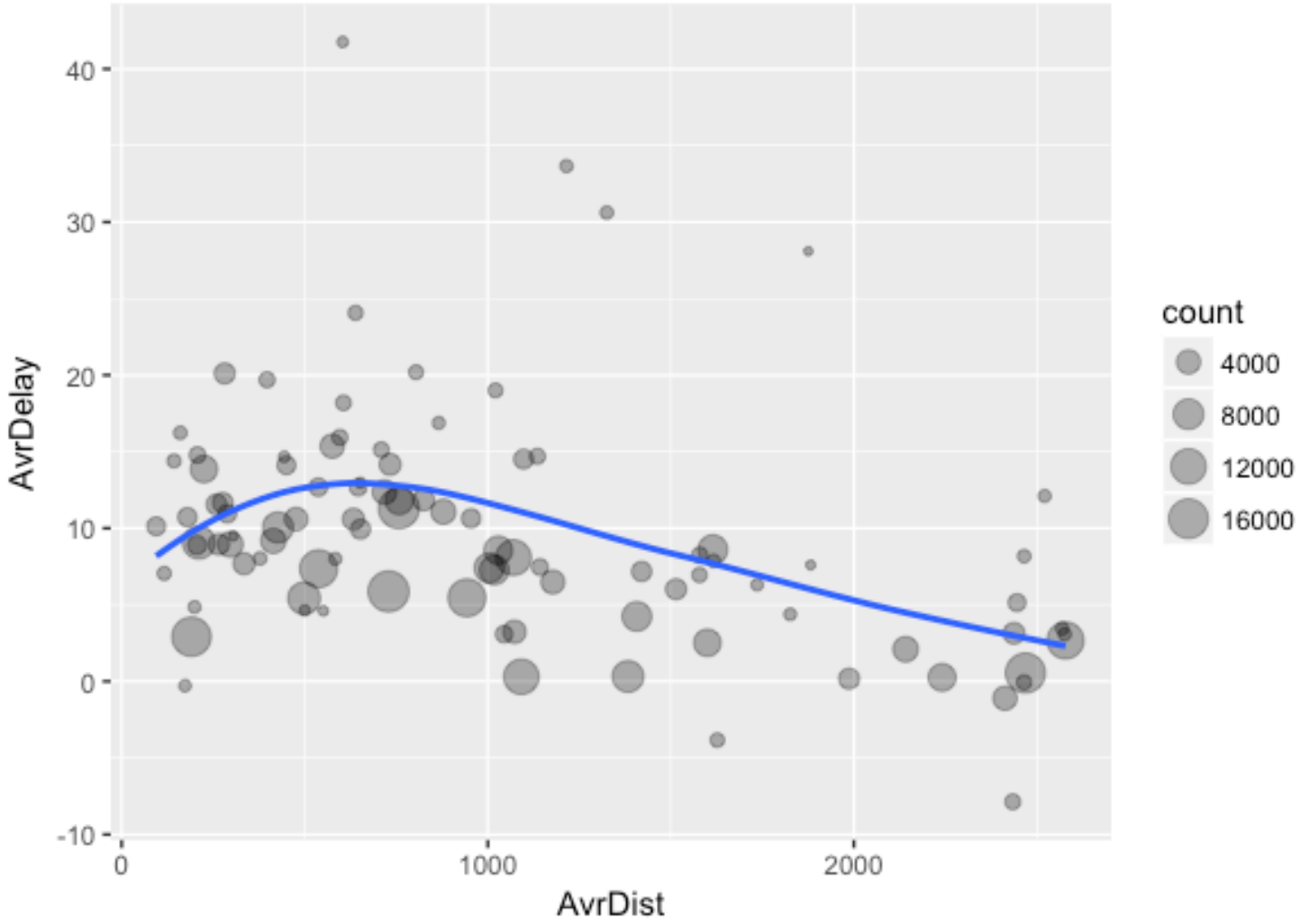
Example

- We want to explore the relationship between distance and average delay for each destination

```
delay <- flights %>% group_by(dest) %>%
          summarize(count=n(),
                    AvrDist=mean(distance,na.rm=T),
                    AvrDelay=mean(arr_delay,na.rm=T)) %>%
          arrange(dest) %>% filter(count>20,dest!="HNL")
```

```
> delay
# A tibble: 96 × 4
    dest count   AvrDist  AvrDelay
   <chr> <int>     <dbl>     <dbl>
1    ABQ   254 1826.0000  4.381890
2    ACK   265  199.0000  4.852273
3    ALB   439  143.0000 14.397129
4    ATL 17215  757.1082 11.300113
5    AUS  2439 1514.2530  6.019909
6    AVL   275  583.5818  8.003831
7    BDL   443  116.0000  7.048544
8    BGR   375  378.0000  8.027933
9    BHM   297  865.9966 16.877323
10   BNA  6333  758.2135 11.812459
# ... with 86 more rows
```

```
ggplot(data=delay,mapping = aes(x=AvrDist,y=AvrDelay)) +
  geom_point(aes(size=count),alpha=1/3)+
  geom_smooth(se=F)
```



Useful Summary Functions

Grouping	Examples
Measures of location	mean(), median()
Measures of spread	sd(), IQR(),mad()
Measures of rank	min((), quantile(), max()
Measures of position	first(), nth(), last()
Counts	n(), n_distinct()
Counts and proportions of logical values	sum(x>0) when used with numeric functions, (T,F) converted to (1,0)

Challenge 5.4

- Look at the number of cancelled flights per day. Is there a pattern? Is the proportion of cancelled flights related to the average delay?
- *Hint: for finding cancelled flights:*

```
> flights %>% filter(is.na(dep_delay),is.na(arr_delay))
# A tibble: 8,255 × 19
    year month   day dep_time sched_dep_time dep_delay
   <int> <int> <int>    <int>          <int>     <dbl>
1   2013     1     1       NA           1630        NA
2   2013     1     1       NA           1935        NA
3   2013     1     1       NA           1500        NA
4   2013     1     1       NA            600        NA
```

dplyr 0.7.0, some new additions

- Two new data sets: starwars and storms
- pull() verb
- case_when(), valuable for mutate operations instead of nested ifs
- tidyeval, using group_by in functions (not for today!)

```
> starwars
# A tibble: 87 x 13
                   name height  mass    hair_color
                  <chr>  <int> <dbl>         <chr>
 1     Luke Skywalker    172    77         blond
 2              C-3PO    167    75          <NA>
 3              R2-D2     96    32          <NA>
 4        Darth Vader    202   136          none
 5        Leia Organa    150    49         brown
 6          Owen Lars    178   120   brown, grey
 7 Beru Whitesun lars    165    75         brown
 8              R5-D4     97    32          <NA>
 9  Biggs Darklighter    183    84         black
10     Obi-Wan Kenobi    182    77 auburn, white
# ... with 77 more rows, and 9 more variables:
#   skin_color <chr>, eye_color <chr>, birth_year <dbl>,
#   gender <chr>, homeworld <chr>, species <chr>,
#   films <list>, vehicles <list>, starships <list>
```

pull() – selecting one column, in vector format

```
> select(mpg,hwy)
# A tibble: 234 x 1
     hwy
   <int>
 1    29
 2    29
 3    31
 4    30
 5    26
 6    26
 7    27
 8    26
 9    25
10    28
# ... with 224 more rows
```

```
> pull(mpg,hwy)
  [1] 29 29 31 30 26 26 27 26 25 28 27 25
 [13] 25 25 25 24 25 23 20 15 20 17 17 26
 [25] 23 26 25 24 19 14 15 17 27 30 26 29
 [37] 26 24 24 22 22 24 24 17 22 21 23 23
 [49] 19 18 17 17 19 19 12 17 15 17 17 12
 [61] 17 16 18 15 16 12 17 17 16 12 15 16
 [73] 17 15 17 17 18 17 19 17 19 19 17 17
 [85] 17 16 16 17 15 17 26 25 26 24 21 22
 [97] 23 22 20 33 32 32 29 32 34 36 36 29
[109] 26 27 30 31 26 26 28 26 29 28 27 24
[121] 24 24 22 19 20 17 12 19 18 14 15 18
[133] 18 15 17 16 18 17 19 19 17 29 27 31
[145] 32 27 26 26 25 25 17 17 20 18 26 26
[157] 27 28 25 25 24 27 25 26 23 26 26 26
[169] 26 25 27 25 27 20 20 19 17 20 17 29
```

pull() – Also works with column numbers, default is last column

```
> head(pull(mpg))
[1] "compact" "compact" "compact" "compact"
[5] "compact" "compact"
>
> head(pull(mpg,1))
[1] "audi" "audi" "audi" "audi" "audi"
[6] "audi"
>
> head(pull(mpg,-1))
[1] "compact" "compact" "compact" "compact"
[5] "compact" "compact"
```

case_when()

- This function allows you to vectorise multiple if and else if statements. It is an R equivalent of the SQL CASE WHEN statement. Arguments:
  - A sequence of two-sided formulas. The left hand side (LHS) determines which values match this case. The right hand side (RHS) provides the replacement value.
  - The LHS must evaluate to a logical vector. Each logical vector can either have length 1 or a common length. All RHSs must evaluate to the same type of vector.

Useful in mutate()

```
starwars %>%
  select(name:mass, gender, species) %>%
  mutate(
    type = case_when(
      height > 200 | mass > 200 ~ "large",
      species == "Droid"        ~ "robot",
      TRUE                      ~  "other"
    )
```

```
# A tibble: 87 x 6
                 name height  mass gender species  type
                <chr>  <int> <dbl>  <chr>   <chr> <chr>
 1     Luke Skywalker    172    77   male   Human other
 2              C-3PO    167    75   <NA>   Droid robot
```