6. Exploratory Data Analysis - dplyr

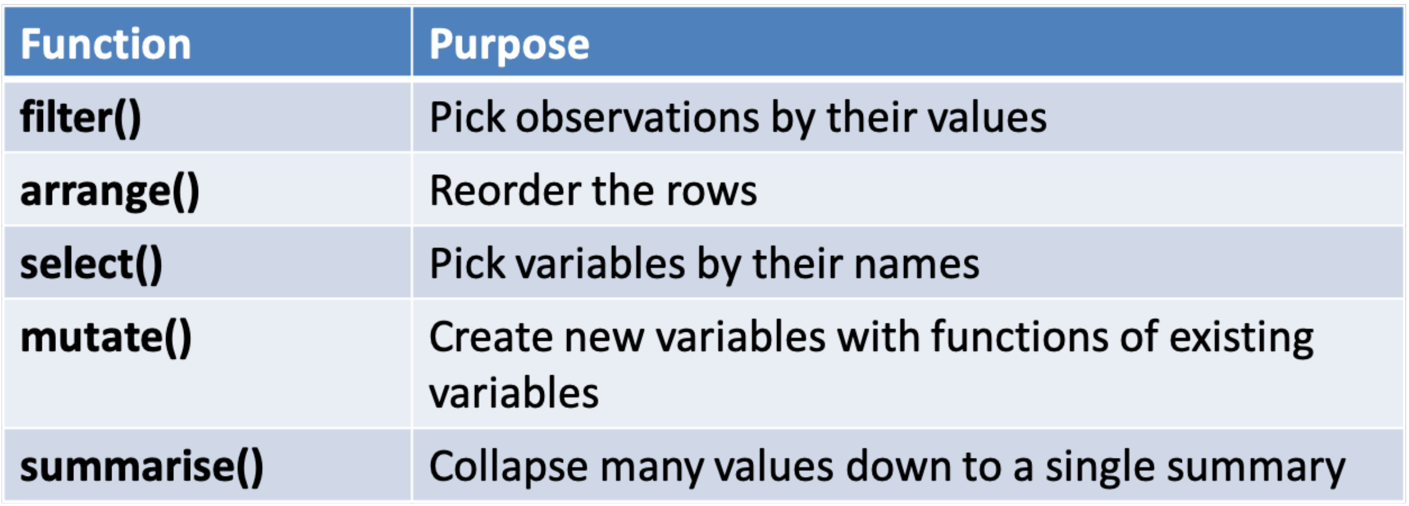
Data Science for OR - J. Duggan

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

dplyr

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



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

Sample Data set ggplot2::mpg

```
## Observations: 234
## Variables: 11
## $ manufacturer <chr> "audi", "audi", "audi", "audi", "audi'
## $ model        <chr> "a4", "a4", "a4", "a4", "a4", "a4", "a
## $ displ        <dbl> 1.8, 1.8, 2.0, 2.0, 2.8, 2.8, 3.1, 1.8
## $ year         <int> 1999, 1999, 2008, 2008, 1999, 1999, 20
## $ cyl          <int> 4, 4, 4, 4, 6, 6, 6, 4, 4, 4, 4, 6, 6,
## $ trans        <chr> "auto(l5)", "manual(m5)", "manual(m6)'
## $ drv          <chr> "f", "f", "f", "f", "f", "f", "f", "4'
## $ cty          <int> 18, 21, 20, 21, 16, 18, 18, 18, 16, 20
## $ hwy          <int> 29, 29, 31, 30, 26, 26, 27, 26, 25, 28
## $ fl           <chr> "p", "p", "p", "p", "p", "p", "p", "p'
## $ class        <chr> "compact", "compact", "compact", "comp
```

(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(mpg,manufacturer=="audi",year==1999,model=="a4")
```

```
## # A tibble: 4 x 11
##   manufacturer model displ  year   cyl trans  drv     cty <
##   <chr>        <chr> <dbl> <int> <int> <chr>  <chr> <int> <
## 1 audi         a4      1.8  1999     4 auto(~ f        18
## 2 audi         a4      1.8  1999     4 manua~ f        21
## 3 audi         a4      2.8  1999     6 auto(~ f        16
## 4 audi         a4      2.8  1999     6 manua~ f        18
```

Cars with highest mpg, lowest mpg?

```
filter(mpg,hwy==max(hwy))
```

```
## # A tibble: 2 x 11
##   manufacturer model  displ  year   cyl trans drv     cty <
##   <chr>        <chr>  <dbl> <int> <int> <chr> <chr> <int> <
## 1 volkswagen   jetta    1.9  1999     4 manu~ f        33
## 2 volkswagen   new b~   1.9  1999     4 manu~ f        35
```

```
filter(mpg,hwy==min(hwy))
```

```
## # A tibble: 5 x 11
##   manufacturer model  displ  year   cyl trans drv     cty <
##   <chr>        <chr>  <dbl> <int> <int> <chr> <chr> <int> <
## 1 dodge        dakot~   4.7  2008     8 auto~ 4         9
## 2 dodge        duran~   4.7  2008     8 auto~ 4         9
## 3 dodge        ram 1~   4.7  2008     8 auto~ 4         9
## 4 dodge        ram 1~   4.7  2008     8 manu~ 4         9
```

Challenge 2.1

- List the cars with an average city mpg greater than the median.
- Show the cars with the maximum displacement

(2) arrange()

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

```
arrange(mpg,displ)
```

```
## # A tibble: 234 x 11
##    manufacturer model displ  year   cyl trans drv     cty
##    <chr>        <chr> <dbl> <int> <int> <chr> <chr> <int> <
##  1 honda        civic   1.6  1999     4 manu~ f        28
##  2 honda        civic   1.6  1999     4 auto~ f        24
##  3 honda        civic   1.6  1999     4 manu~ f        25
##  4 honda        civic   1.6  1999     4 manu~ f        23
##  5 honda        civic   1.6  1999     4 auto~ f        24
##  6 audi         a4      1.8  1999     4 auto~ f        18
##  7 audi         a4      1.8  1999     4 manu~ f        21
##  8 audi         a4 q~   1.8  1999     4 manu~ 4        18
##  9 audi         a4 q~   1.8  1999     4 auto~ 4        16
```

Show in descending order

```
arrange(mpg,desc(displ))
```

```
## # A tibble: 234 x 11
##    manufacturer model displ  year   cyl trans drv     cty
##    <chr>        <chr> <dbl> <int> <int> <chr> <chr> <int> <
##  1 chevrolet    corv~   7    2008     8 manu~ r        15
##  2 chevrolet    k150~   6.5  1999     8 auto~ 4        14
##  3 chevrolet    corv~   6.2  2008     8 manu~ r        16
##  4 chevrolet    corv~   6.2  2008     8 auto~ r        15
##  5 jeep         gran~   6.1  2008     8 auto~ 4        11
##  6 chevrolet    c150~   6    2008     8 auto~ r        12
##  7 dodge        dura~   5.9  1999     8 auto~ 4        11
##  8 dodge        ram ~   5.9  1999     8 auto~ 4        11
##  9 chevrolet    c150~   5.7  1999     8 auto~ r        13
## 10 chevrolet    corv~   5.7  1999     8 manu~ r        16
## # ... with 224 more rows
```

Add an extra sort column

```
arrange(mpg,desc(year),desc(displ))
```

```
## # A tibble: 234 x 11
##    manufacturer model displ  year   cyl trans drv     cty
##    <chr>        <chr> <dbl> <int> <int> <chr> <chr> <int> <
##  1 chevrolet    corv~   7    2008     8 manu~ r        15
##  2 chevrolet    corv~   6.2  2008     8 manu~ r        16
##  3 chevrolet    corv~   6.2  2008     8 auto~ r        15
##  4 jeep         gran~   6.1  2008     8 auto~ 4        11
##  5 chevrolet    c150~   6    2008     8 auto~ r        12
##  6 dodge        dura~   5.7  2008     8 auto~ 4        13
##  7 dodge        ram ~   5.7  2008     8 auto~ 4        13
##  8 jeep         gran~   5.7  2008     8 auto~ 4        13
##  9 toyota       land~   5.7  2008     8 auto~ 4        13
## 10 nissan       path~   5.6  2008     8 auto~ 4        12
## # ... with 224 more rows
```

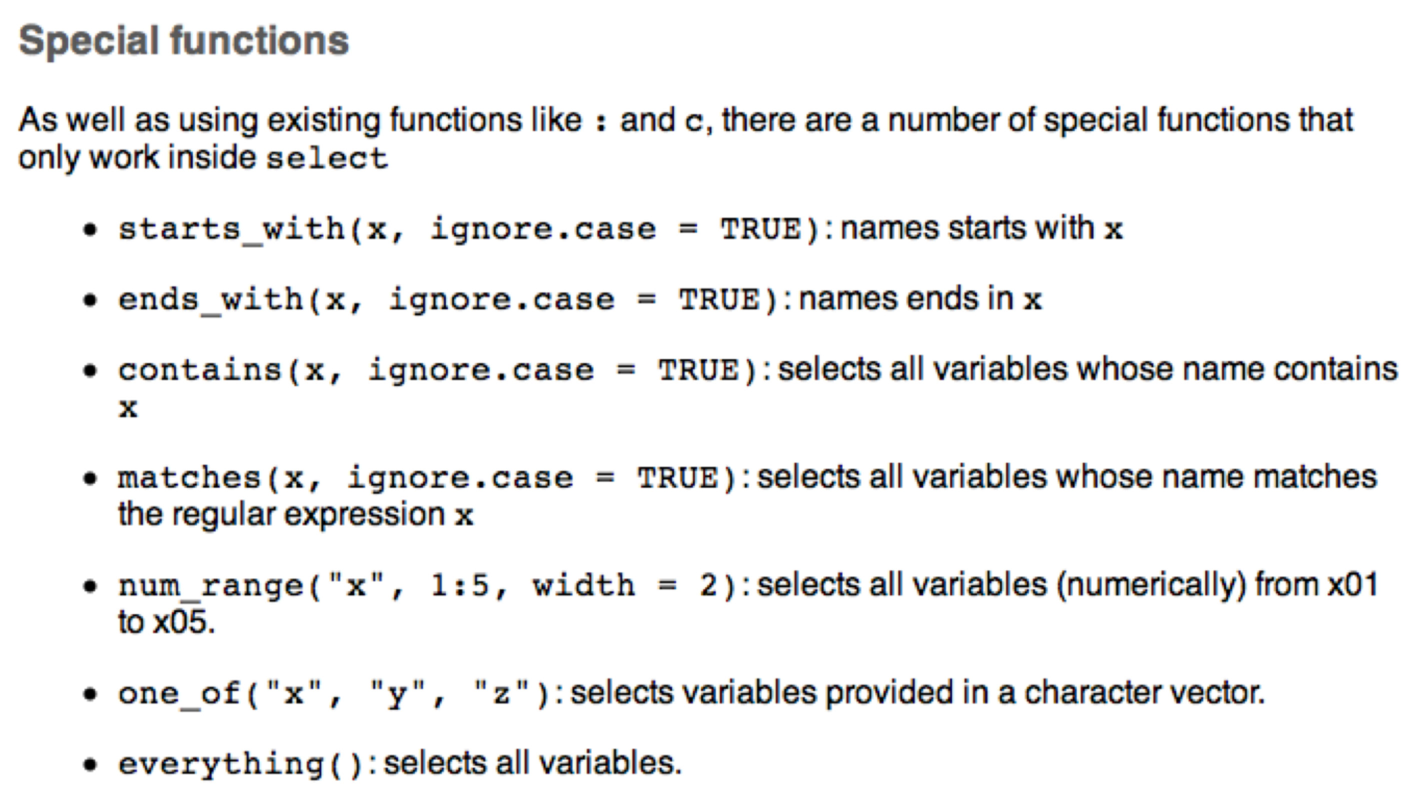
(3) select()

- 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(mpg,model,year,displ, cty, hwy)
```

```
## # A tibble: 234 x 5
##    model         year displ   cty   hwy
##    <chr>        <int> <dbl> <int> <int>
##  1 a4            1999   1.8    18    29
##  2 a4            1999   1.8    21    29
##  3 a4            2008   2      20    31
##  4 a4            2008   2      21    30
##  5 a4            1999   2.8    16    26
##  6 a4            1999   2.8    18    26
```

Special Function 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.

(4) mutate()

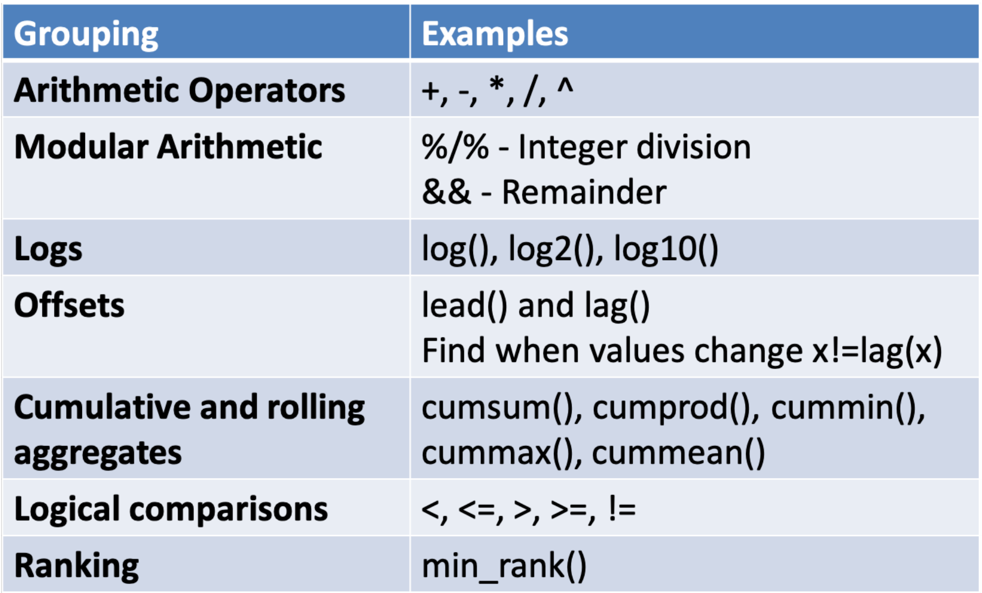
- It is often useful to add new columns that are functions of existing columns
- mutate() always adds new columns at the end of your data set.

```
sml <- select(mpg,model,displ,cty)
sml <- mutate(sml,Category=ifelse(cty>mean(cty),"AboveAvr","Be
sml
```

```
## # A tibble: 234 x 4
##    model      displ   cty Category
##    <chr>      <dbl> <int> <chr>
##  1 a4           1.8    18 AboveAvr
##  2 a4           1.8    21 AboveAvr
##  3 a4           2      20 AboveAvr
##  4 a4           2      21 AboveAvr
##  5 a4           2.8    16 BelowAvr
##  6 a4           2.8    18 AboveAvr
```

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



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 %>%
- 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)

```
mpg %>% select(model,displ,cty) %>% slice(1:2)
```

```
## # A tibble: 2 x 3
##   model displ   cty
##   <chr> <dbl> <int>
## 1 a4      1.8    18
## 2 a4      1.8    21
```

The function group_by()

- Most summary 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 can then performed “by group”.

```
gr <- group_by(mpg,year)
agg <- summarise(gr,AverageCty=mean(cty))
agg
```

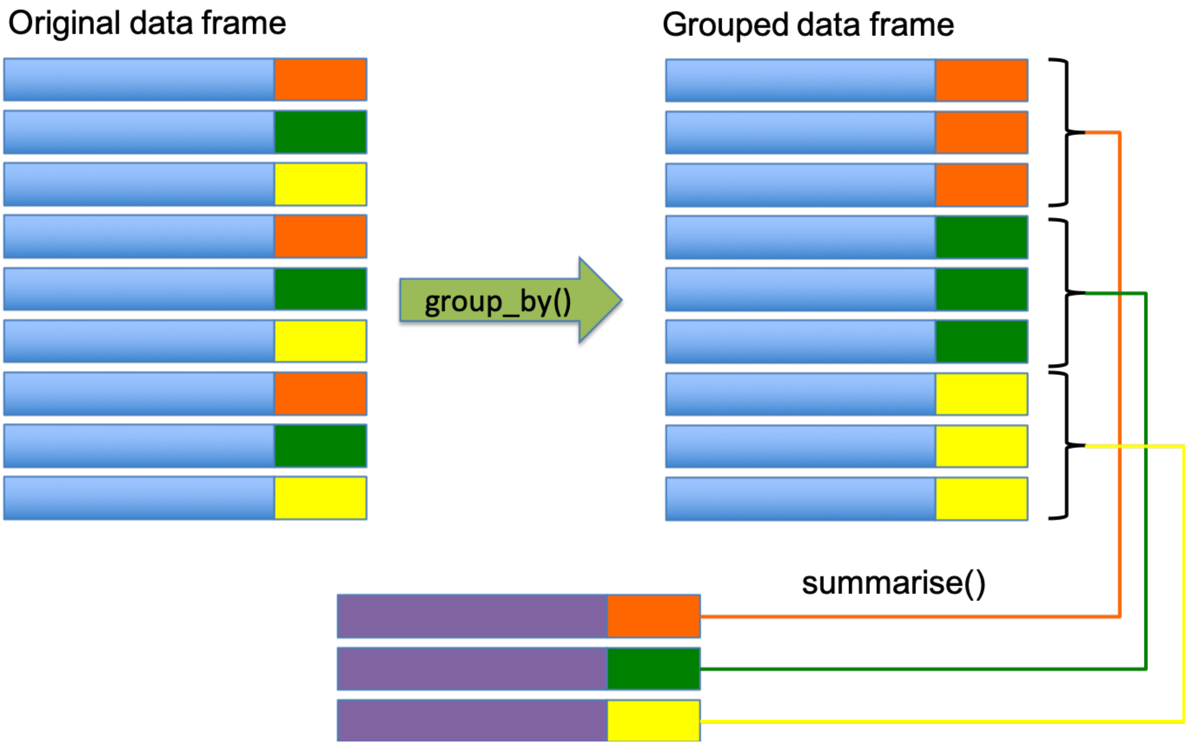
```
## # A tibble: 2 x 2
##    year AverageCty
##   <int>      <dbl>
## 1  1999       17.0
## 2  2008       16.7
```

Using %>%

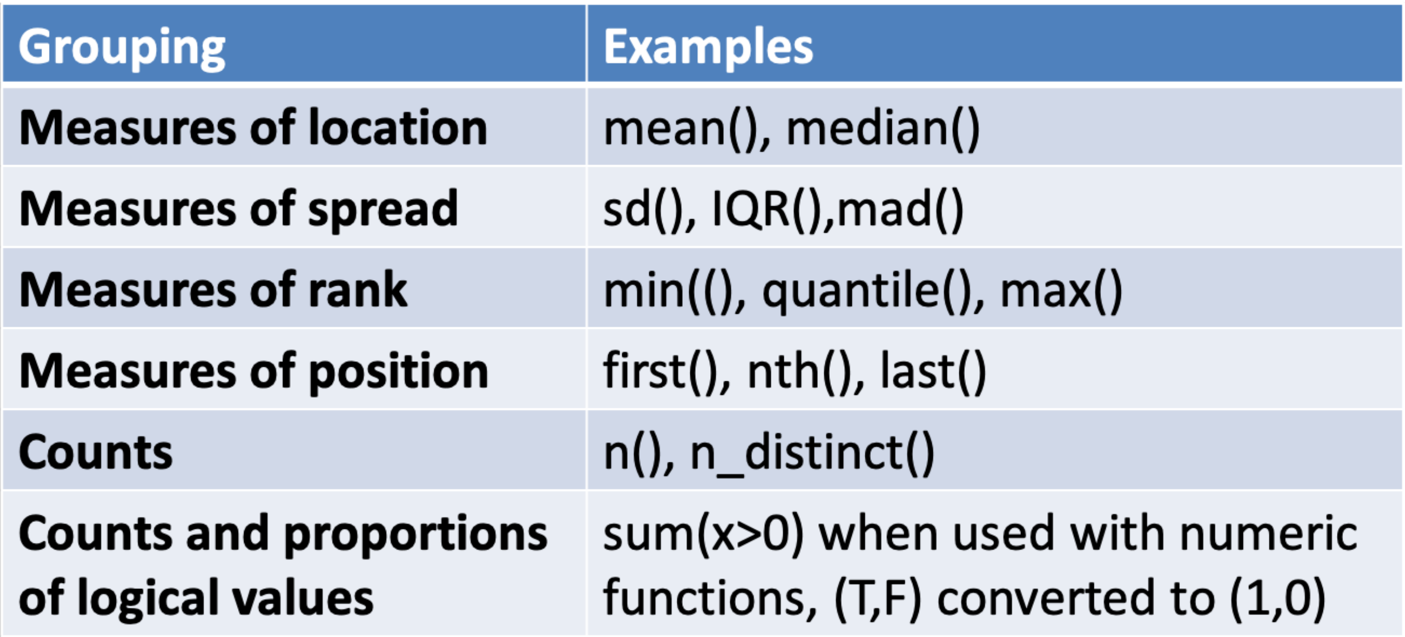
```
mpg %>% group_by(manufacturer)              %>%
        summarise(AvrCty=mean(cty),N=n()) %>%
        arrange(desc(AvrCty))             %>%
        slice(1:5)
```

```
## # A tibble: 5 x 3
##   manufacturer AvrCty     N
##   <chr>         <dbl> <int>
## 1 honda          24.4     9
## 2 volkswagen     20.9    27
## 3 subaru         19.3    14
## 4 hyundai        18.6    14
## 5 toyota         18.5    34
```

Overall idea



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)

The package nycflights13

```
glimpse(nycflights13::flights)
```

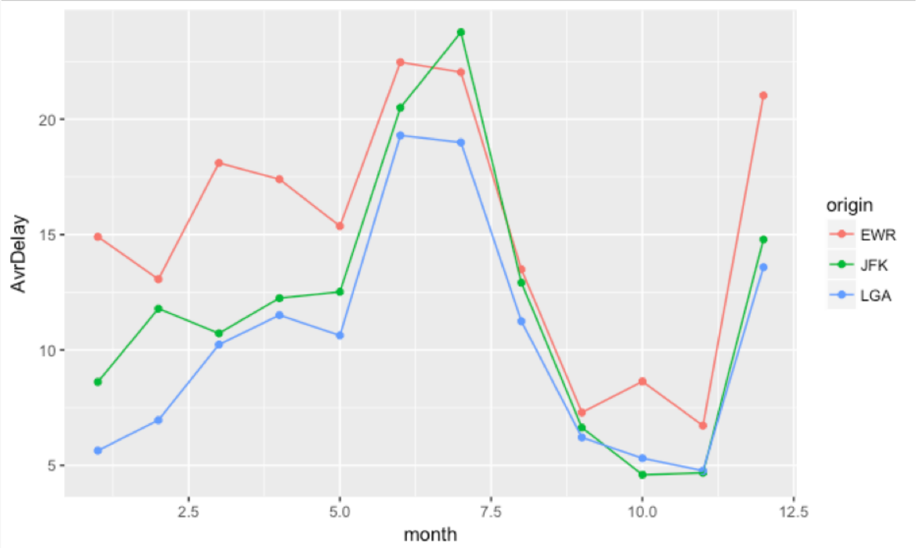
```
## Observations: 336,776
## Variables: 19
## $ year           <int> 2013, 2013, 2013, 2013, 2013, 2013,
## $ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## $ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## $ dep_time       <int> 517, 533, 542, 544, 554, 554, 555, 5
## $ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600, 6
## $ dep_delay      <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, -2,
## $ arr_time       <int> 830, 850, 923, 1004, 812, 740, 913,
## $ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 854,
## $ arr_delay      <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -
## $ carrier        <chr> "UA", "UA", "AA", "B6", "DL", "UA",
## $ flight         <int> 1545, 1714, 1141, 725, 461, 1696, 50
## $ tailnum        <chr> "N14228", "N24211", "N619AA", "N804
```

Challenge 2.2 | nycflights13::flights

Generate the following graph. Use the variable **dep_delay**. The variable **origin** indicates the departure airport.

```
unique(nycflights13::flights$origin)
```

```
## [1] "EWR" "LGA" "JFK"
```



Summary

- dplyr - a grammar of data manipulation
- Five verbs
  - **filter()**
  - **arrange()**
  - **select()**
  - **mutate()**
  - **summarise()** (along with **group_by()**)
- Usefully combined with **%>%** operator