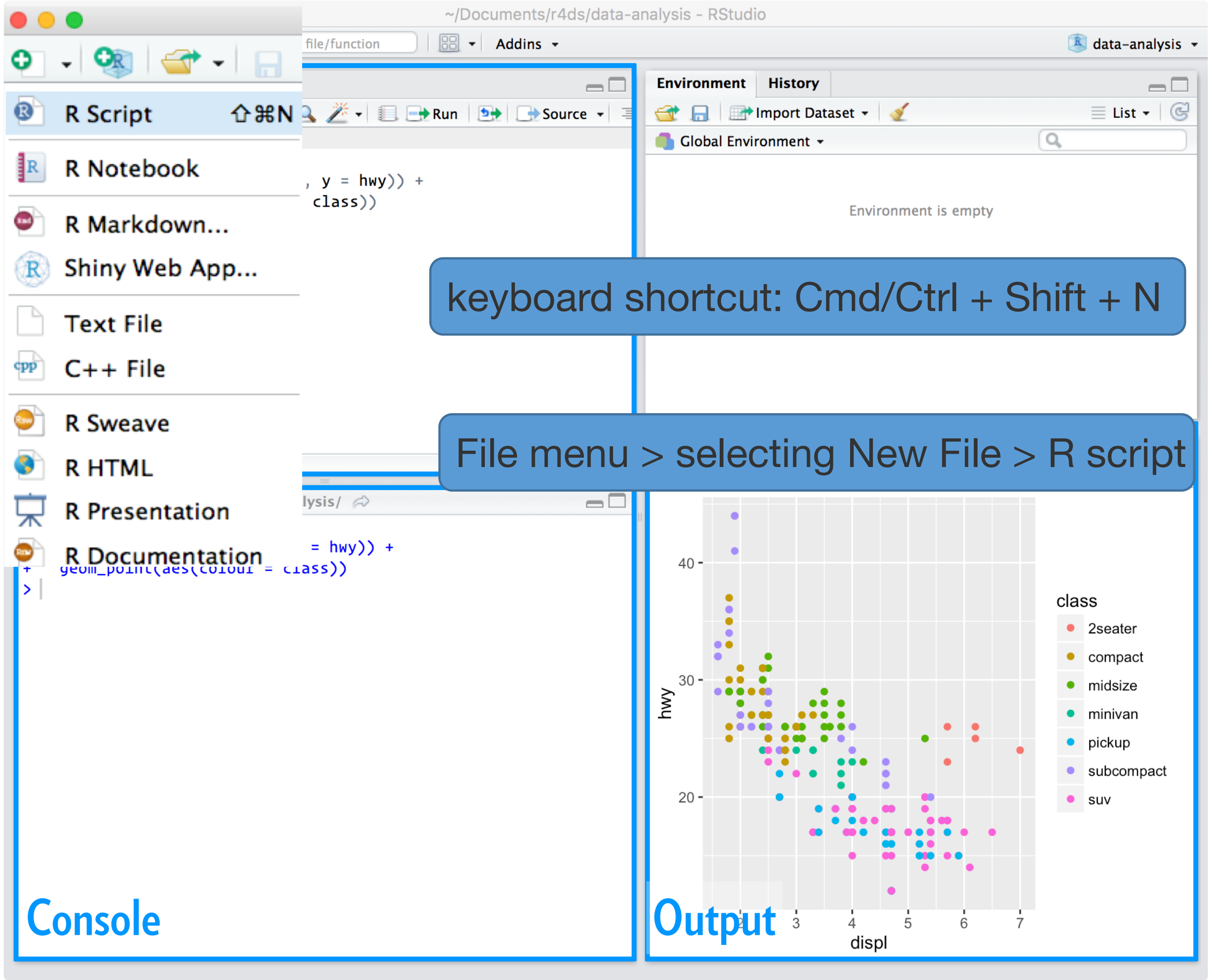
Epidemiology & Data Science

+ Script

Yanru Xing
UNIVERSITY OF FLORIDA

R for Data Science
Hadley Wickham & Garrett Grolemund

Workflow: scripts

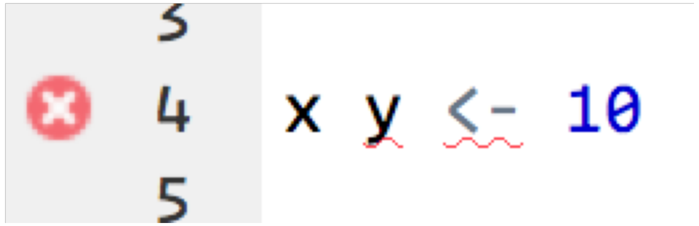


Benefits of using scripts editor:

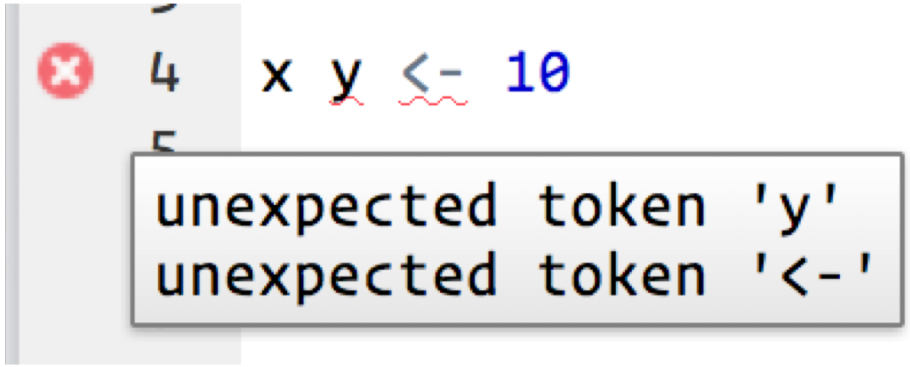
1. Automatically save and automatically load
2. Running code line by line, or run multiple lines by selecting
3. Save script as file

RStudio diagnostics

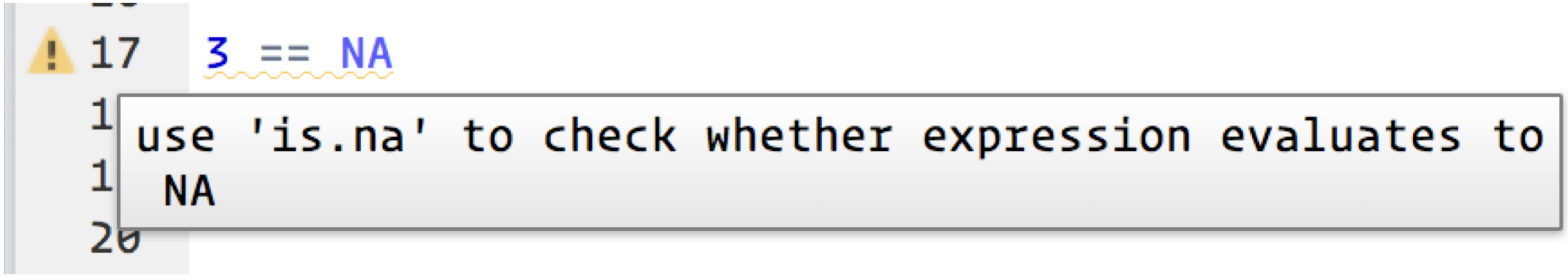
The script editor will highlight syntax errors with a red squiggly line and a cross in the sidebar:



Hover over the cross to see what the problem is:



RStudio will also let you know about potential problems:



What other common mistakes will RStudio diagnostics report? Readhttps://support.rstudio.com/hc/en-us/articles/205753617-Code-Diagnostics to find out.

Tidyverse

install.packages("tidyverse")

```
— Attaching packages ——————————————————————————————— t
✔ ggplot2 2.2.1     ✔ purrr   0.2.5
✔ tibble  1.4.2     ✔ dplyr   0.7.6
✔ tidyr   0.8.1     ✔ stringr 1.3.1
✔ readr   1.1.1     ✔ forcats 0.3.0
package 'dplyr' was built under R version 3.5.1— Conflicts
—————————————————————————————— tidyverse_conflicts()
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
```

dplyr --- A Grammar of Data Manipulation

library(nycflights13)
library(tidyverse)

```
nycflights13::flights #This data frame contains all 336,776 flights that departed from New York City in 2013
```

```
## # A tibble: 336,776 x 19
##     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>
```

Type of each variable:

1. ***int*** --- intergers
2. ***dbl*** --- doubles or real numbers
3. ***chr*** --- character vectors, or strings
4. ***dttm*** --- date-times (a data + a time)
5. ***lgl*** --- logical, vectors contain only TRUE or FALSE
6. ***fctr*** --- factors, categorical variables
7. ***data*** --- dates

Five key dplyr functions

- ***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

1. The first argument is a data frame
2. The subsequent arguments describe what to do with the data frame
3. The result is new data frame

Filter rows with *filter( )*

filter() allows you to subset observations based on their values. The first argument is the name of the data frame. The second and subsequent arguments are the expressions that filter the data frame. For example, we can select all flights on January 1st with:

```
filter(flights, month == 1, day == 1)
```

```
## # A tibble: 842 x 19
##     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 832 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>
```

```r
jan1 <- filter(flights, month == 1, day == 1)
jan1
```

```
## # A tibble: 842 x 19
##     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 832 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>
```

Comparison Operators

Select observations using the comparison operators

Standard suite: ***>, >=, <, <=, != (not equal), and == (equal)***

The easiest mistake to make:
filter(flights , month = 1)
#> Error: filter() takes unnamed arguments. Do you need `==`?

Another common problem you might encounter when using ==: floating point numbers:

```
sqrt(2) ^ 2 == 2
#> [1] FALSE
1 / 49 * 49 == 1
#> [1] FALSE
```

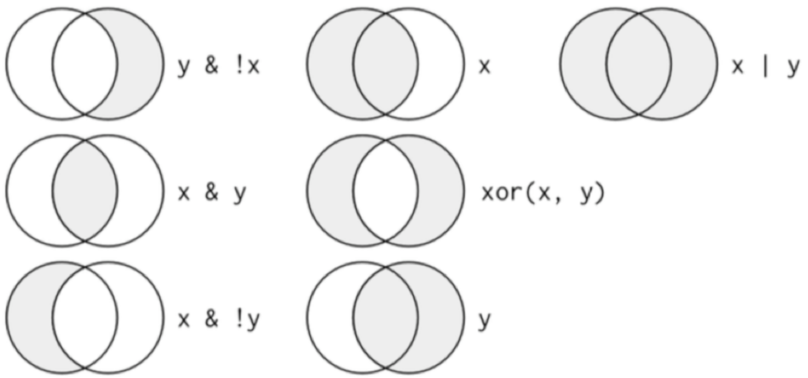
```
near(sqrt(2) ^ 2, 2)
#> [1] TRUE
near(1 / 49 * 49, 1)
#> [1] TRUE
```

dplyr

Logical Operators

Boolean operators:

`&` ***is "and"***
`|` ***is "or"***
`!` ***is "not"***



Complete set of boolean operations. x is the left-hand circle, y is the right-hand circle, and the shaded region show which parts each operator selects.

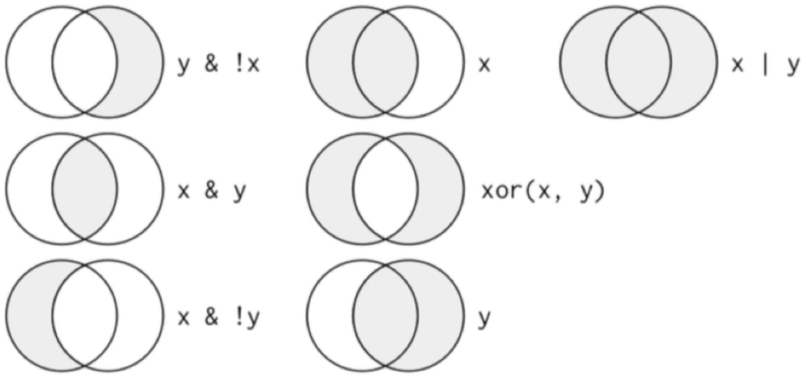
```
# The following code finds all flights that departed in November or December:
filter(flights, month == 11 | month == 12)
```

```
## # A tibble: 55,403 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013    11     1        5           2359         6      352
##  2  2013    11     1       35           2250       105      123
##  3  2013    11     1      455            500        -5      641
##  4  2013    11     1      539            545        -6      856
##  5  2013    11     1      542            545        -3      831
##  6  2013    11     1      549            600       -11      912
##  7  2013    11     1      550            600       -10      705
##  8  2013    11     1      554            600        -6      659
##  9  2013    11     1      554            600        -6      826
## 10  2013    11     1      554            600        -6      749
## # ... with 55,393 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>
```

filter(flights, month %in% c(11, 12))

Logical Operators

!(x | y) **is the same as *!x & !y***



Find flights that weren't delayed (on arrival or departure) by more than two hours, you could use either of the following two filters:

```
filter(flights, !(arr_delay > 120 | dep_delay > 120))

filter(flights, arr_delay <= 120, dep_delay <= 120)
```

Missing Values

Missing values, or NAs ("not availables")
NA represents an unknown value so missing values are "contagious"; almost any operation involving an unknown value will also be unknown:

```
NA > 5
#> [1] NA
10 == NA
#> [1] NA
NA + 10
#> [1] NA
NA / 2
#> [1] NA
```

```
NA == NA
#> [1] NA
```

```
# Let x be Mary's age. We don't know how old she is.
x <- NA

# Let y be John's age. We don't know how old he is.
y <- NA

# Are John and Mary the same age?
x == y
#> [1] NA
# We don't know!
```

```
is.na(x)
#> [1] TRUE
```

filter()

Filter( ) only includes rows where the condition is TRUE; it excludes both FALSE and NA values. If you want to preserve missing values, ask for them explicitly:

```
df <- tibble(x = c(1, NA, 3))
filter(df, x > 1)
```

```
## # A tibble: 1 x 1
##       x
##   <dbl>
## 1     3
```

```
filter(df, is.na(x) | x > 1)
```

```
## # A tibble: 2 x 1
##       x
##   <dbl>
## 1    NA
## 2     3
```

Arrange Rows with *arrange()*

arrange() works similarly to ***filter()*** except that instead of selecting rows, it changes their order. It takes a data frame and a set of column names (or more complicated expressions) to order by.

If you provide more than one column name, each additional column will be used to break ties in the values of preceding columns:

```
arrange(flights, year, month, day)
```

```
## # A tibble: 336,776 x 19
##     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>
```

Use desc() to re-order by a column in descending order:

```
arrange(flights, desc(dep_delay))
```

```
## # A tibble: 336,776 x 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
##  8  2013     6    27      959           1900       899     1236
##  9  2013     7    22     2257            759       898      121
## 10  2013    12     5      756           1700       896     1058
## # ... 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>
```

Missing values are always sorted at the end:

```
df <- tibble(x = c(5, 2, NA))
arrange(df, x)
#> # A tibble: 3 x 1
#>       x
#>   <dbl>
#> 1     2
#> 2     5
#> 3    NA
arrange(df, desc(x))
#> # A tibble: 3 x 1
#>       x
#>   <dbl>
#> 1     5
#> 2     2
#> 3    NA
```

Select Columns with *select()*

It's not uncommon to get datasets with hundreds or even thousands of variables. In this case, the first challenge is often narrowing in on the variables you're actually interested in.

select() allows you to rapidly zoom in on a useful subset using operations based on the names of the variables.

select() is not terribly useful with the flight data because we only have 19 variables, but you can still get the general idea:

```r
# Select columns by name
select(flights, year, month, day)
```

```
## # A tibble: 336,776 x 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
```

```r
# Select all columns between year and day (inclusive)
select(flights, year:day)
```

```
## # A tibble: 336,776 x 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
```

dplyr

```r
# Select all columns except those from year to day (inclusive)
select(flights, -(year:day))
```

```
## # A tibble: 336,776 x 16
##    dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
##       <int>          <int>     <dbl>    <int>          <int>     <dbl>
##  1      517            515         2      830            819        11
##  2      533            529         4      850            830        20
##  3      542            540         2      923            850        33
##  4      544            545        -1     1004           1022       -18
##  5      554            600        -6      812            837       -25
##  6      554            558        -4      740            728        12
##  7      555            600        -5      913            854        19
##  8      557            600        -3      709            723       -14
##  9      557            600        -3      838            846        -8
## 10      558            600        -2      753            745         8
## # ... with 336,766 more rows, and 10 more variables: carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
## #   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

```r
select(flights, contains("ime"))
```

```
## # A tibble: 336,776 x 6
##    dep_time sched_dep_time arr_time sched_arr_time air_time
##       <int>          <int>    <int>          <int>    <dbl>
##  1      517            515      830            819      227
##  2      533            529      850            830      227
##  3      542            540      923            850      160
##  4      544            545     1004           1022      183
##  5      554            600      812            837      116
##  6      554            558      740            728      150
##  7      555            600      913            854      158
##  8      557            600      709            723       53
##  9      557            600      838            846      140
## 10      558            600      753            745      138
## # ... with 336,766 more rows, and 1 more variable: time_hour <dttm>
```

Add new variables with *mutate()*

mutate() adds new columns at the end of your dataset, new columns that are functions of existing columns

```
flights_sml <- select(flights,
  year:day,
  ends_with("delay"),
  distance,
  air_time
)
mutate(flights_sml,
  gain = dep_delay - arr_delay,
  speed = distance / air_time * 60
)
```

```r
flights_sml <- select(flights,
  year:day,
  ends_with("delay"),
  distance,
  air_time
)
mutate(flights_sml,
  gain = dep_delay - arr_delay,
  speed = distance / air_time * 60
)
```

```
## # A tibble: 336,776 x 9
##     year month   day dep_delay arr_delay distance air_time  gain speed
##    <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl> <dbl>
##  1  2013     1     1         2        11     1400      227    -9  370.
##  2  2013     1     1         4        20     1416      227   -16  374.
##  3  2013     1     1         2        33     1089      160   -31  408.
##  4  2013     1     1        -1       -18     1576      183    17  517.
##  5  2013     1     1        -6       -25      762      116    19  394.
##  6  2013     1     1        -4        12      719      150   -16  288.
##  7  2013     1     1        -5        19     1065      158   -24  404.
##  8  2013     1     1        -3       -14      229       53    11  259.
```

Refer to columns that you've just created ***mutate()***

```
mutate(flights_sml,
  gain = dep_delay - arr_delay,
  hours = air_time / 60,
  gain_per_hour = gain / hours
)
```

```
## # A tibble: 336,776 x 10
##      year month   day dep_delay arr_delay distance air_time  gain hours
##     <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl> <dbl>
##  1  2013     1     1         2        11     1400      227    -9 3.78
##  2  2013     1     1         4        20     1416      227   -16 3.78
##  3  2013     1     1         2        33     1089      160   -31 2.67
##  4  2013     1     1        -1       -18     1576      183    17 3.05
##  5  2013     1     1        -6       -25      762      116    19 1.93
##  6  2013     1     1        -4        12      719      150   -16 2.5
##  7  2013     1     1        -5        19     1065      158   -24 2.63
##  8  2013     1     1        -3       -14      229       53    11 0.883
##  9  2013     1     1        -3        -8      944      140     5 2.33
## 10  2013     1     1        -2         8      733      138   -10 2.3
## # ... with 336,766 more rows, and 1 more variable: gain_per_hour <dbl>
```

Keep the new variables ***transmute()***

```
transmute(flights,
  gain = dep_delay - arr_delay,
  hours = air_time / 60,
  gain_per_hour = gain / hours
)
```

```
## # A tibble: 336,776 x 3
##      gain hours gain_per_hour
##     <dbl> <dbl>         <dbl>
##  1     -9 3.78          -2.38
##  2    -16 3.78          -4.23
##  3    -31 2.67         -11.6
##  4     17 3.05           5.57
##  5     19 1.93           9.83
##  6    -16 2.5           -6.4
##  7    -24 2.63          -9.11
##  8     11 0.883         12.5
##  9      5 2.33           2.14
## 10    -10 2.3           -4.35
## # ... with 336,766 more rows
```

Grouped summaries with *summarise()*

summarise( ) collapses a data frame to a single row

```
# summarise() collapses a data frame to a single row:
summarise(flights, delay = mean(dep_delay, na.rm = TRUE))
```

```
## # A tibble: 1 x 1
##   delay
##   <dbl>
## 1  12.6
```

group_by(), changes the unit of analysis from the complete dataset to individual groups

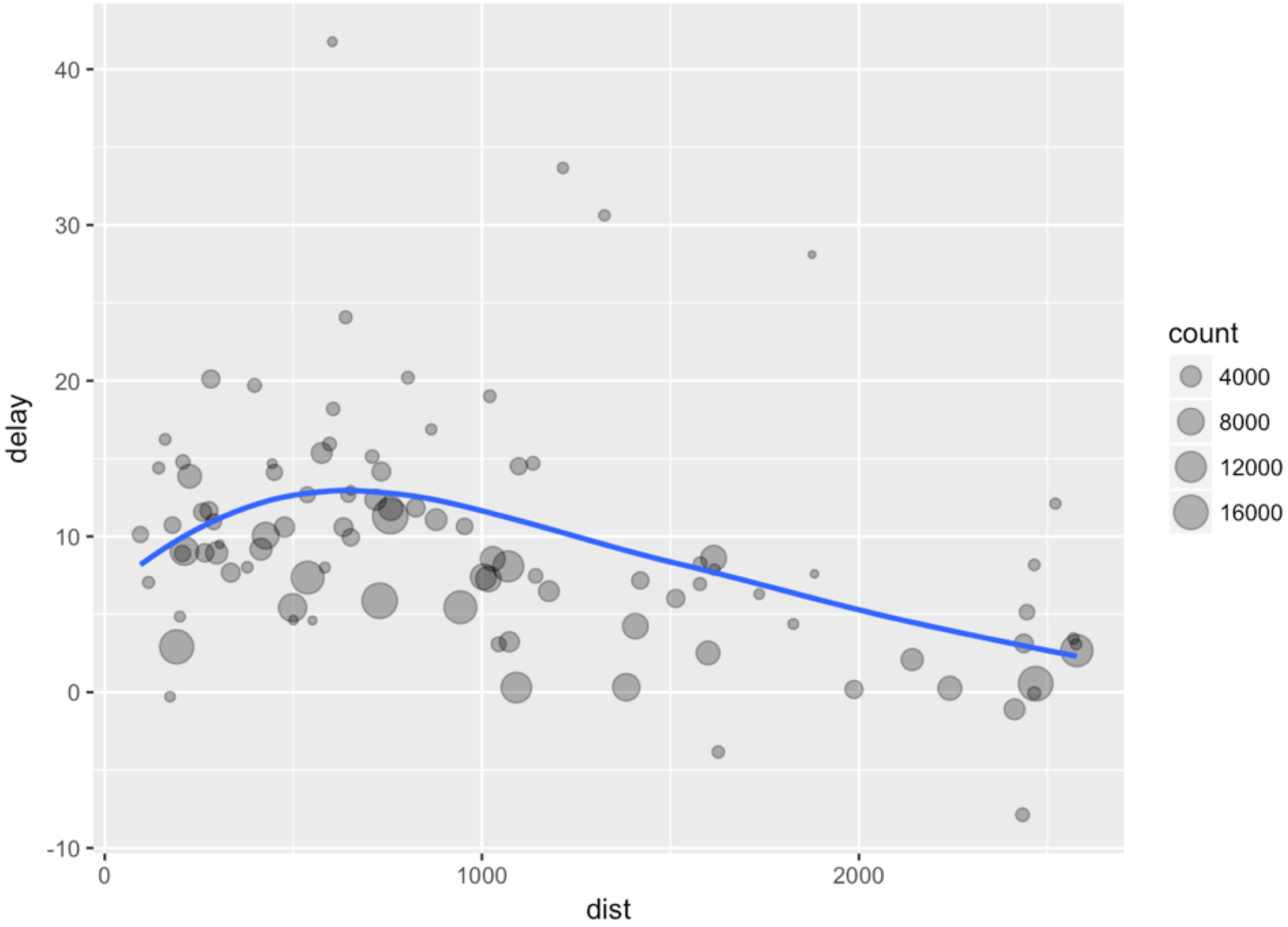
```
by_day <- group_by(flights, year, month, day)
summarise(by_day, delay = mean(dep_delay, na.rm = TRUE))
```

```
## # A tibble: 365 x 4
## # Groups:   year, month [?]
##     year month   day delay
##    <int> <int> <int> <dbl>
##  1  2013     1     1 11.5
##  2  2013     1     2 13.9
##  3  2013     1     3 11.0
##  4  2013     1     4  8.95
##  5  2013     1     5  5.73
##  6  2013     1     6  7.15
##  7  2013     1     7  5.42
##  8  2013     1     8  2.55
##  9  2013     1     9  2.28
## 10  2013     1    10  2.84
## # ... with 355 more rows
```

Combining multiple operations with the pipe

explore the relationship between the distance and average delay for each location

```
by_dest <- group_by(flights, dest)
delay <- summarise(by_dest,
  count = n(),
  dist = mean(distance, na.rm = TRUE),
  delay = mean(arr_delay, na.rm = TRUE)
)
delay <- filter(delay, count > 20, dest != "HNL")
# It looks like delays increase with distance up to ~750 miles
# and then decrease. Maybe as flights get longer there's more
# ability to make up delays in the air?
ggplot(data = delay, mapping = aes(x = dist, y = delay)) +
  geom_point(aes(size = count), alpha = 1/3) +
  geom_smooth(se = FALSE)
```



40
30
20
10
0
-10
0
1000
2000
delay
dist
count
4000
8000
12000
16000

Combining multiple operations with the pipe

```
not_cancelled <- flights %>%
  filter(!is.na(dep_delay), !is.na(arr_delay))

not_cancelled %>%
  group_by(year, month, day) %>%
  summarise(mean = mean(dep_delay))
```

```
## # A tibble: 365 x 4
## # Groups:   year, month [?]
##     year month   day  mean
##    <int> <int> <int> <dbl>
##  1  2013     1     1 11.4
##  2  2013     1     2 13.7
##  3  2013     1     3 10.9
##  4  2013     1     4  8.97
##  5  2013     1     5  5.73
##  6  2013     1     6  7.15
##  7  2013     1     7  5.42
##  8  2013     1     8  2.56
##  9  2013     1     9  2.30
## 10  2013     1    10  2.84
## # ... with 355 more rows
```

Grouped mutates (and filters)

Grouping is most useful in conjunction with summarise(), but you can also do convenient operations with ***mutate()*** and ***filter():***

```
# Find the worst members of each group:
flights_sml %>%
  group_by(year, month, day) %>%
  filter(rank(desc(arr_delay)) < 10)
```

```
## # A tibble: 3,306 x 7
## # Groups:   year, month, day [365]
##     year month   day dep_delay arr_delay distance air_time
##    <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl>
##  1  2013     1     1       853       851      184       41
##  2  2013     1     1       290       338     1134      213
##  3  2013     1     1       260       263      266       46
##  4  2013     1     1       157       174      213       60
##  5  2013     1     1       216       222      708      121
##  6  2013     1     1       255       250      589      115
##  7  2013     1     1       285       246     1085      146
##  8  2013     1     1       192       191      199       44
##  9  2013     1     1       379       456     1092      222
## 10  2013     1     2       224       207      550       94
## # ... with 3,296 more rows
```

```r
# Find all groups bigger than a threshold:
popular_dests <- flights %>%
  group_by(dest) %>%
  filter(n() > 365)
popular_dests
```

```
## # A tibble: 332,577 x 19
## # Groups:   dest [77]
##     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 332,567 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>
```

```r
# Standardise to compute per group metrics:
popular_dests %>%
  filter(arr_delay > 0) %>%
  mutate(prop_delay = arr_delay / sum(arr_delay)) %>%
  select(year:day, dest, arr_delay, prop_delay)
```

```
## # A tibble: 131,106 x 6
## # Groups:   dest [77]
##     year month   day dest  arr_delay prop_delay
##    <int> <int> <int> <chr>     <dbl>      <dbl>
##  1  2013     1     1 IAH          11  0.000111
##  2  2013     1     1 IAH          20  0.000201
##  3  2013     1     1 MIA          33  0.000235
##  4  2013     1     1 ORD          12  0.0000424
##  5  2013     1     1 FLL          19  0.0000938
##  6  2013     1     1 ORD           8  0.0000283
##  7  2013     1     1 LAX           7  0.0000344
##  8  2013     1     1 DFW          31  0.000282
##  9  2013     1     1 ATL          12  0.0000400
## 10  2013     1     1 DTW          16  0.000116
## # ... with 131,096 more rows
```