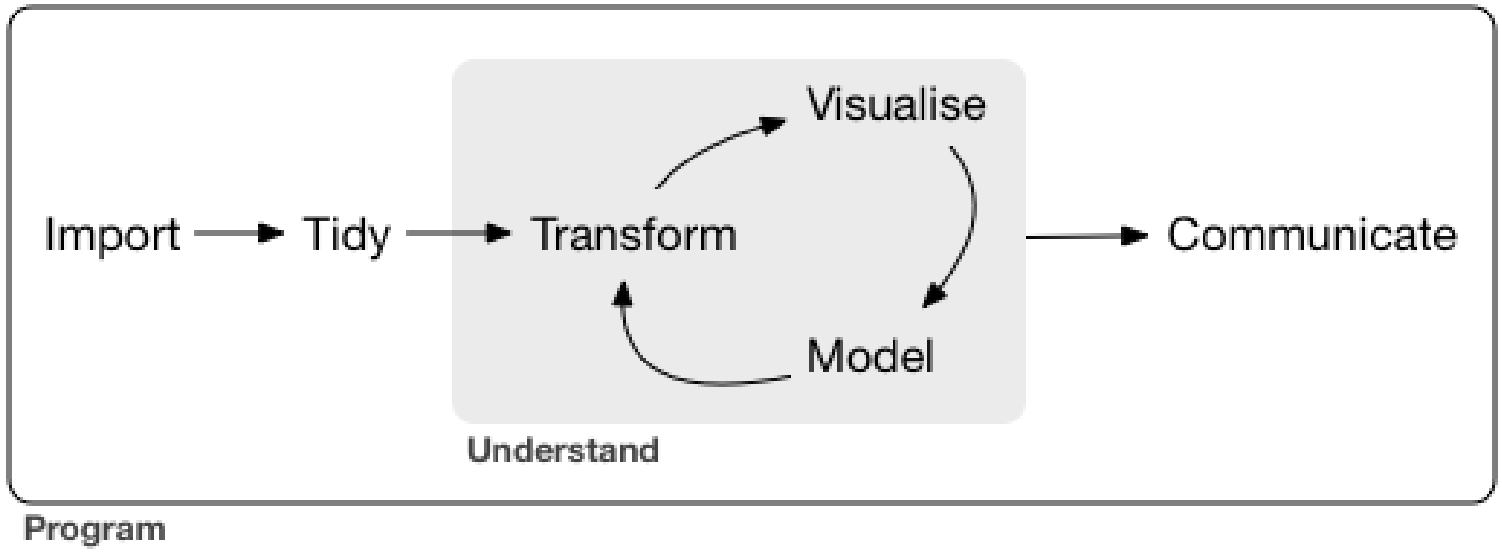
Real world data with R

Data cleaning

Alexandru Cernat

Data analysis framework



Basics of R

Why use R?

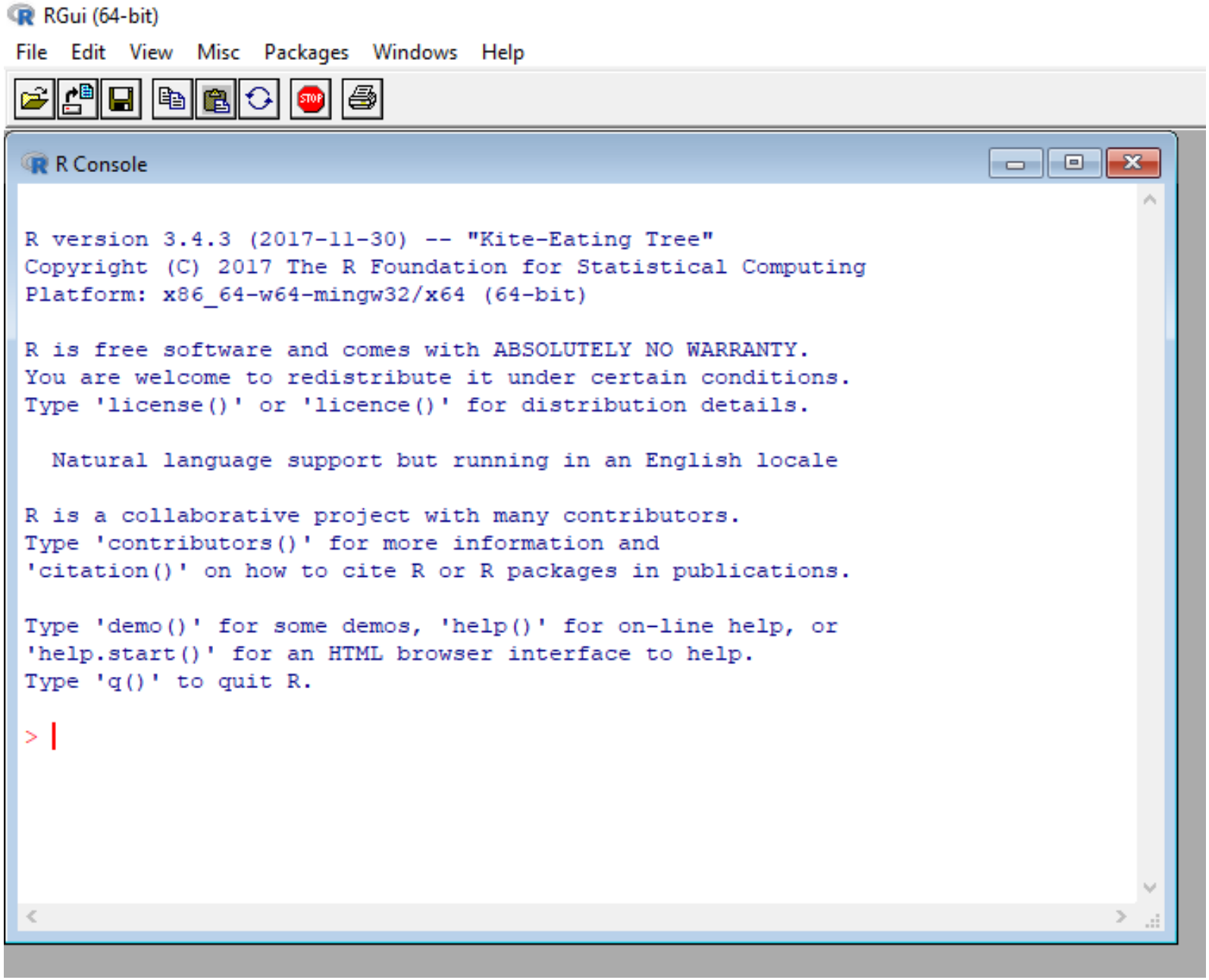
Open source

Flexible programming

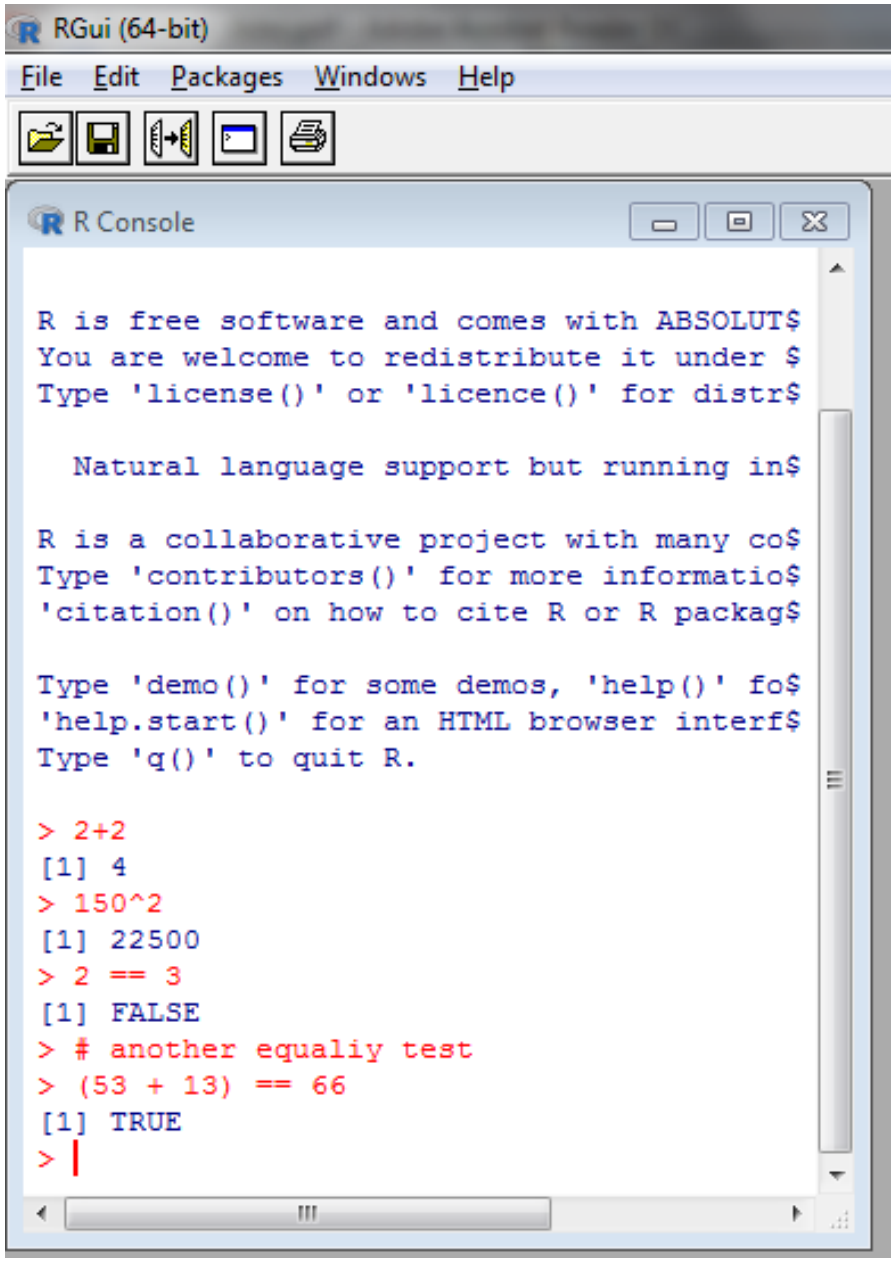
Wide range of statistical methods

Beautiful graphs

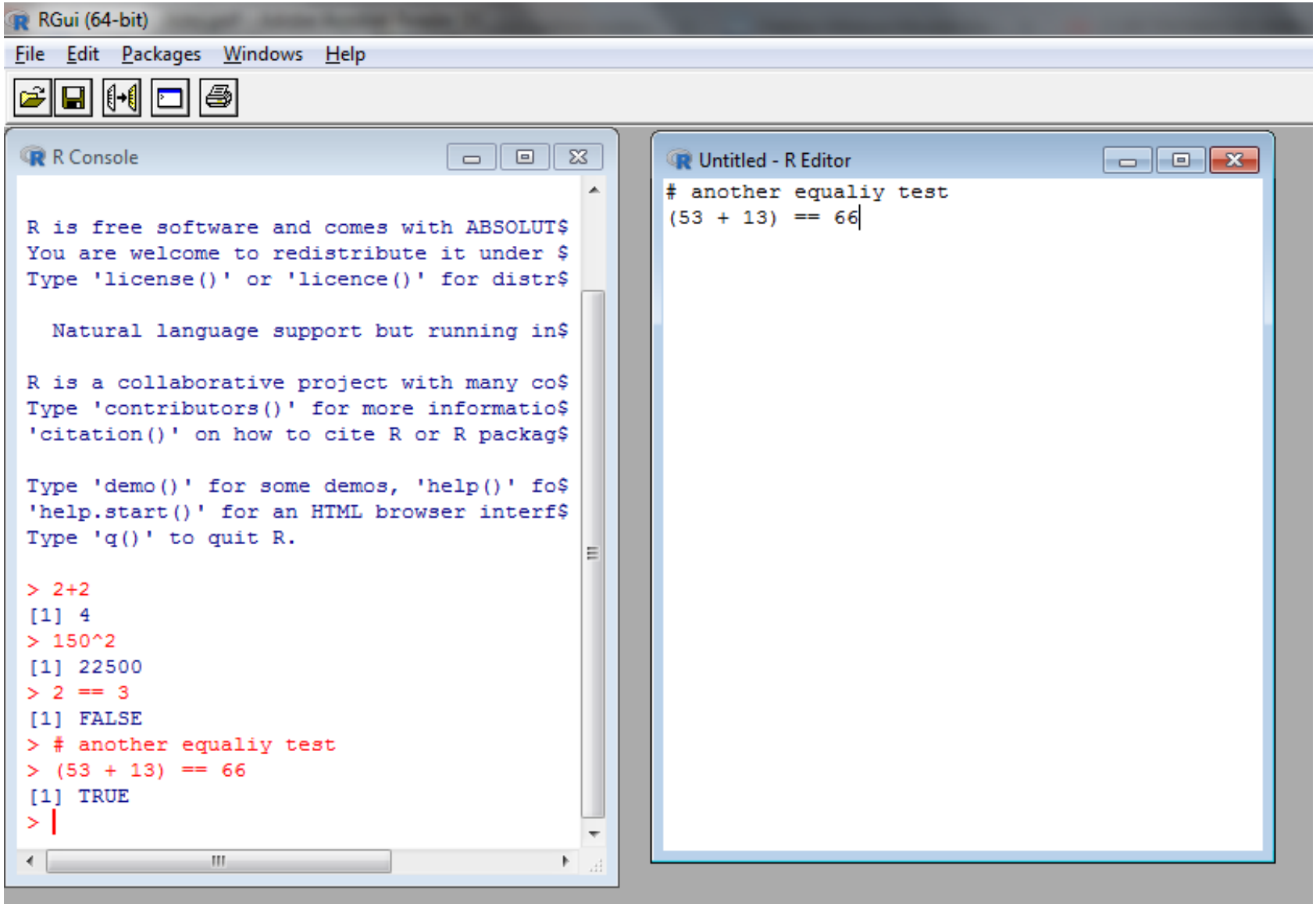
R console


RGui (64-bit)
File Edit View Misc Packages Windows Help
R Console
R version 3.4.3 (2017-11-30) -- "Kite-Eating Tree"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>

R console interactive



R console and script

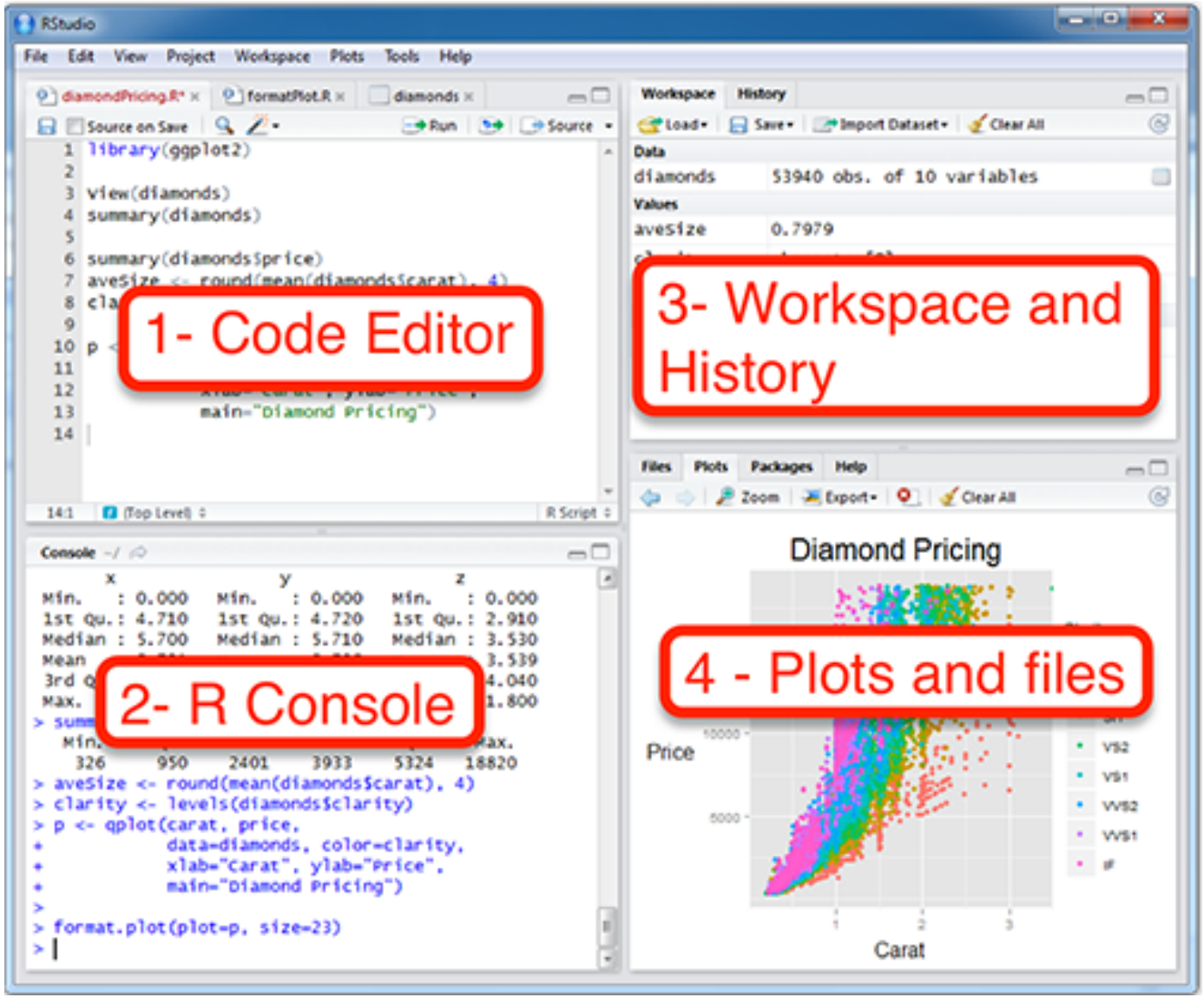


RStudio

Integrated development environment (IDE)

It includes:
console
syntax-highlighting
direct code execution
history
debugging
work-space management

Rstudio interface



Packages

Collections of functions and objects

Thousands free of user-written packages

Can significantly extend the capabilities of R

Package install and load

```r
# install package
install.packages("tidyverse")

# load package
library(tidyverse)
```

Basic work-flow

1. Make new project (optional)
or set up working directory

Basic work-flow

1. Make new project (optional)
or set up working directory

2. Make new R script/syntax
Run syntax using "Ctrl + enter"

Basic work-flow

1. Make new project (optional)
or set up working directory

2. Make new R script/syntax
Run syntax using "Ctrl + enter"

3. Save syntax and data for later use

Style guide

Good writing practice is essential
for **future proofing**
and **collaboration**

Whatever you choose, **be consistent!**

https://style.tidyverse.org/

Using the tidyverse

A package collection for management and graphing

Examples of packages:
- readr
- tidyr
- dplyr
- forcats
- stringr
- ggplot2

Tidy data

What is tidy data?

Tidy data is a standard way of mapping the meaning of a dataset to its structure

1. Each variable forms a column

2. Each observation forms a row

3. Each unit type forms a table

Typical issues with "found" data

Column headers are values

Multiple variables in one column

Variables stored in rows and columns

Multiple types of units in the same table

A single unit in multiple tables

How to solve these problems?

Re-code variables

Restructure data using ‘pivot‘

Learn more:

```
vignette("pivot")
```

Selecting cases and variables

Looking at flight data

```
library(nycflights13)
flights
```

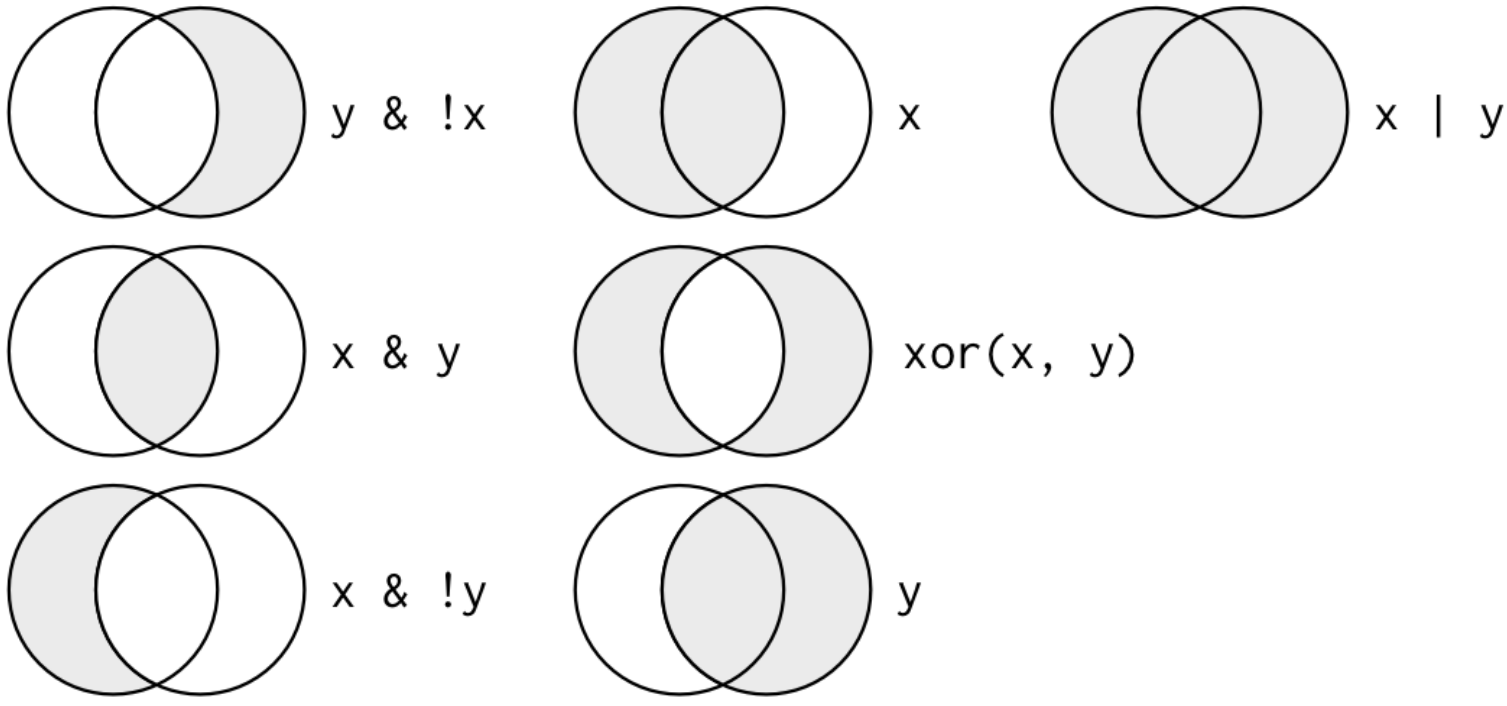
```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_ti
##    <int> <int> <int>    <int>          <int>     <dbl>    <in
##  1  2013     1     1      517            515         2      8
##  2  2013     1     1      533            529         4      8
##  3  2013     1     1      542            540         2      9
##  4  2013     1     1      544            545        -1     10
##  5  2013     1     1      554            600        -6      8
##  6  2013     1     1      554            558        -4      7
##  7  2013     1     1      555            600        -5      9
##  8  2013     1     1      557            600        -3      7
##  9  2013     1     1      557            600        -3      8
## 10  2013     1     1      558            600        -2      7
## # ... with 336,766 more rows, and 11 more variables: arr_dela
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
```

Filtering rows with filter()

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

```
## # A tibble: 842 x 19
##      year month   day dep_time sched_dep_time dep_delay arr_ti
##     <int> <int> <int>    <int>          <int>     <dbl>    <in
##  1  2013     1     1      517            515         2      8
##  2  2013     1     1      533            529         4      8
##  3  2013     1     1      542            540         2      9
##  4  2013     1     1      544            545        -1     10
##  5  2013     1     1      554            600        -6      8
##  6  2013     1     1      554            558        -4      7
##  7  2013     1     1      555            600        -5      9
##  8  2013     1     1      557            600        -3      7
##  9  2013     1     1      557            600        -3      8
## 10  2013     1     1      558            600        -2      7
## # ... with 832 more rows, and 11 more variables: arr_delay <d
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
```

Logical operators



More complex selections

```
filter(flights, month == 11 | month == 12)
```

```
## # A tibble: 55,403 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_ti
##    <int> <int> <int>    <int>          <int>     <dbl>    <in
##  1  2013    11     1        5           2359         6      3
##  2  2013    11     1       35           2250       105      1
##  3  2013    11     1      455            500        -5      6
##  4  2013    11     1      539            545        -6      8
##  5  2013    11     1      542            545        -3      8
##  6  2013    11     1      549            600       -11      9
##  7  2013    11     1      550            600       -10      7
##  8  2013    11     1      554            600        -6      6
##  9  2013    11     1      554            600        -6      8
## 10  2013    11     1      554            600        -6      7
## # ... with 55,393 more rows, and 11 more variables: arr_delay
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
```

More complex selections

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

```
## # A tibble: 55,403 x 19
##      year month   day dep_time sched_dep_time dep_delay arr_ti
##     <int> <int> <int>    <int>          <int>     <dbl>    <in
##  1  2013    11     1        5           2359         6      3
##  2  2013    11     1       35           2250       105      1
##  3  2013    11     1      455            500        -5      6
##  4  2013    11     1      539            545        -6      8
##  5  2013    11     1      542            545        -3      8
##  6  2013    11     1      549            600       -11      9
##  7  2013    11     1      550            600       -10      7
##  8  2013    11     1      554            600        -6      6
##  9  2013    11     1      554            600        -6      8
## 10  2013    11     1      554            600        -6      7
## # ... with 55,393 more rows, and 11 more variables: arr_delay
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
```

More complex selections

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

# or

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

Arranging and selecting columns

Arrange rows

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

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_ti
##    <int> <int> <int>    <int>          <int>     <dbl>    <in
##  1  2013     1     1      517            515         2      8
##  2  2013     1     1      533            529         4      8
##  3  2013     1     1      542            540         2      9
##  4  2013     1     1      544            545        -1     10
##  5  2013     1     1      554            600        -6      8
##  6  2013     1     1      554            558        -4      7
##  7  2013     1     1      555            600        -5      9
##  8  2013     1     1      557            600        -3      7
##  9  2013     1     1      557            600        -3      8
## 10  2013     1     1      558            600        -2      7
## # ... with 336,766 more rows, and 11 more variables: arr_dela
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
```

Arrange rows descending

```
arrange(flights, desc(dep_time))
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_ti
##    <int> <int> <int>    <int>          <int>     <dbl>    <in
##  1  2013    10    30     2400           2359         1      3
##  2  2013    11    27     2400           2359         1      5
##  3  2013    12     5     2400           2359         1      4
##  4  2013    12     9     2400           2359         1      4
##  5  2013    12     9     2400           2250        70
##  6  2013    12    13     2400           2359         1      4
##  7  2013    12    19     2400           2359         1      4
##  8  2013    12    29     2400           1700       420      3
##  9  2013     2     7     2400           2359         1      4
## 10  2013     2     7     2400           2359         1      4
## # ... with 336,766 more rows, and 11 more variables: arr_dela
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
```

Selecting columns with select()

```
select(flights, year, month, day)
# or
```

```
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
```

Excluding columns

```
select(flights, -(year:day))
```

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

Other useful commands

```
- start_with("abc")

- ends_with("xyz")

- matches("(.)\\1")

- num_range("x", 1:3)
```

Ordering columns

```
select(flights, time_hour, air_time, everything())
```

```
## # A tibble: 336,776 x 19
##    time_hour           air_time  year month   day dep_time sc
##    <dttm>                 <dbl> <int> <int> <int>    <int>
##  1 2013-01-01 05:00:00      227  2013     1     1      517
##  2 2013-01-01 05:00:00      227  2013     1     1      533
##  3 2013-01-01 05:00:00      160  2013     1     1      542
##  4 2013-01-01 05:00:00      183  2013     1     1      544
##  5 2013-01-01 06:00:00      116  2013     1     1      554
##  6 2013-01-01 05:00:00      150  2013     1     1      554
##  7 2013-01-01 06:00:00      158  2013     1     1      555
##  8 2013-01-01 06:00:00       53  2013     1     1      557
##  9 2013-01-01 06:00:00      140  2013     1     1      557
## 10 2013-01-01 06:00:00      138  2013     1     1      558
## # ... with 336,766 more rows, and 12 more variables: dep_dela
## #   arr_time <int>, sched_arr_time <int>, arr_delay <dbl>, ca
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, di
## #   hour <dbl>, minute <dbl>
```

Renaming variables

```
rename(flights, deptime = dep_time)
```

```
## # A tibble: 336,776 x 19
##      year month   day deptime sched_dep_time dep_delay arr_tim
##     <int> <int> <int>   <int>          <int>     <dbl>    <int
##  1  2013     1     1     517            515         2      83
##  2  2013     1     1     533            529         4      85
##  3  2013     1     1     542            540         2      92
##  4  2013     1     1     544            545        -1     100
##  5  2013     1     1     554            600        -6      81
##  6  2013     1     1     554            558        -4      74
##  7  2013     1     1     555            600        -5      91
##  8  2013     1     1     557            600        -3      70
##  9  2013     1     1     557            600        -3      83
## 10  2013     1     1     558            600        -2      75
## # ... with 336,766 more rows, and 11 more variables: arr_dela
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
```

Re-coding data

Introducing the pipe operator "%>%"

```
arrange(select(flights, time_hour, air_time, everything()),

# same as
flights %>%
  select(time_hour, air_time, everything()) %>%
  arrange(time_hour)
```

Add new variables with mutate()

```
flights %>%
  select(ends_with("delay"),
         distance, air_time) %>%
  mutate(gain = arr_delay - dep_delay,
         speed = distance / air_time * 60)
```

```
## # A tibble: 336,776 x 6
##    dep_delay arr_delay distance air_time  gain speed
##        <dbl>     <dbl>    <dbl>    <dbl> <dbl> <dbl>
##  1         2        11     1400      227     9  370.
##  2         4        20     1416      227    16  374.
##  3         2        33     1089      160    31  408.
##  4        -1       -18     1576      183   -17  517.
##  5        -6       -25      762      116   -19  394.
##  6        -4        12      719      150    16  288.
##  7        -5        19     1065      158    24  404.
##  8        -3       -14      229       53   -11  259.
##  9        -3        -8      944      140    -5  405.
## 10        -2         8      733      138    10  319.
## # ... with 336,766 more rows
```

Keep only new variables with transmute()

```
transmute(flights,
          gain = arr_delay - dep_delay,
          speed = distance / air_time * 60)
```

```
## # A tibble: 336,776 x 2
##     gain speed
##    <dbl> <dbl>
##  1     9  370.
##  2    16  374.
##  3    31  408.
##  4   -17  517.
##  5   -19  394.
##  6    16  288.
##  7    24  404.
##  8   -11  259.
##  9    -5  405.
## 10    10  319.
## # ... with 336,766 more rows
```

Using modular arithmetic

```
transmute(flights, dep_time,
          hour = dep_time %/% 100,
          minute = dep_time %% 100)
```

```
## # A tibble: 336,776 x 3
##    dep_time  hour minute
##       <int> <dbl>  <dbl>
##  1      517     5     17
##  2      533     5     33
##  3      542     5     42
##  4      544     5     44
##  5      554     5     54
##  6      554     5     54
##  7      555     5     55
##  8      557     5     57
##  9      557     5     57
## 10      558     5     58
## # ... with 336,766 more rows
```

Factor variables

Factor variables in R

Used to work with categorical variables

They have **fixed** and **known** possible values

Avoids coding errors and helps ordering

Using factors in R

```r
x1 <- c("Dec", "Apr", "Jan", "Mar")

month_levels <- c(
  "Jan", "Feb", "Mar", "Apr", "May", "Jun",
  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
)

y1 <- factor(x1, levels = month_levels)
y1
```

```
## [1] Dec Apr Jan Mar
## Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

Sorting characters versus factors

```
sort(x1)
```

```
## [1] "Apr" "Dec" "Jan" "Mar"
```

```
sort(y1)
```

```
## [1] Jan Mar Apr Dec
## Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

Accessing the levels of a factor

```
levels(y1)
```

```
##  [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Se
```

You cannot add invalid levels

```
f2[5] <- "July"
```

```
## Error in f2[5] <- "July": object 'f2' not found
```

Reordering levels of factor

```
levels(gss_cat$rincome)
```

```
##  [1] "No answer"      "Don't know"     "Refused"        "$25000 or more"
##  [5] "$20000 - 24999" "$15000 - 19999" "$10000 - 14999" "$8000 to 9999"
##  [9] "$7000 to 7999"  "$6000 to 6999"  "$5000 to 5999"  "$4000 to 4999"
## [13] "$3000 to 3999"  "$1000 to 2999"  "Lt $1000"       "Not applicable"
```

```
gss_cat$rincome2 <- fct_relevel(gss_cat$rincome, "Not applicable")
levels(gss_cat$rincome2)
```

```
##  [1] "Not applicable" "No answer"      "Don't know"     "Refused"
##  [5] "$25000 or more" "$20000 - 24999" "$15000 - 19999" "$10000 - 14999"
##  [9] "$8000 to 9999"  "$7000 to 7999"  "$6000 to 6999"  "$5000 to 5999"
## [13] "$4000 to 4999"  "$3000 to 3999"  "$1000 to 2999"  "Lt $1000"
```

Modifying factor levels

```
count(gss_cat, partyid)
```

```
## # A tibble: 10 x 2
##    partyid              n
##  * <fct>            <int>
##  1 No answer          154
##  2 Don't know           1
##  3 Other party        393
##  4 Strong republican 2314
##  5 Not str republican 3032
##  6 Ind,near rep       1791
##  7 Independent        4119
##  8 Ind,near dem       2499
##  9 Not str democrat   3690
## 10 Strong democrat    3490
```

Modifying factor levels tidyverse way

```
gss_cat %>%
  mutate(partyid = fct_recode(partyid,
    "Republican, strong"    = "Strong republican",
    "Republican, weak"      = "Not str republican",
    "Independent, near rep" = "Ind,near rep",
    "Independent, near dem" = "Ind,near dem",
    "Democrat, weak"        = "Not str democrat",
    "Democrat, strong"      = "Strong democrat"
  ))
```

Modifying factor levels tidyverse way

```
## # A tibble: 10 x 2
##    partyid                     n
##  * <fct>                   <int>
##  1 No answer                 154
##  2 Don't know                  1
##  3 Other party               393
##  4 Republican, strong       2314
##  5 Republican, weak         3032
##  6 Independent, near rep    1791
##  7 Independent              4119
##  8 Independent, near dem    2499
##  9 Democrat, weak           3690
## 10 Democrat, strong         3490
```

Collapsing categories

```
gss_cat %>%
  mutate(partyid = fct_collapse(partyid,
    other = c("No answer", "Don't know", "Other party"),
    rep = c("Republican, strong", "Republican, weak"),
    ind = c("Independent, near rep", "Independent, near dem"),
    dem = c("Democrat, weak", "Democrat, strong")
  )) %>%
  count(partyid)
```

```
## # A tibble: 8 x 2
##   partyid                n
## * <fct>                <int>
## 1 other                  548
## 2 Strong republican     2314
## 3 Not str republican    3032
## 4 Ind,near rep          1791
## 5 Independent           4119
## 6 Ind,near dem          2499
## 7 Not str democrat      3690
## 8 Strong democrat       3490
```

Lumping categories together by size

```
count(gss_cat, relig) %>% arrange(n)
```

```
## # A tibble: 15 x 2
##    relig                      n
##    <fct>                  <int>
##  1 Don't know                15
##  2 Native american           23
##  3 Other eastern             32
##  4 Hinduism                  71
##  5 No answer                 93
##  6 Orthodox-christian        95
##  7 Moslem/islam             104
##  8 Inter-nondenominational  109
##  9 Buddhism                 147
## 10 Other                    224
## 11 Jewish                   388
## 12 Christian                689
## 13 None                    3523
## 14 Catholic                5124
## 15 Protestant             10846
```

Lumping categories together by size

```
mutate(gss_cat, relig2 = fct_lump(relig, n = 13)) %>%
  count(relig2) %>% arrange(n)
```

```
## # A tibble: 13 x 2
##    relig2                      n
##    <fct>                   <int>
##  1 Other eastern              32
##  2 Hinduism                   71
##  3 No answer                  93
##  4 Orthodox-christian         95
##  5 Moslem/islam              104
##  6 Inter-nondenominational   109
##  7 Buddhism                  147
##  8 Other                     262
##  9 Jewish                    388
## 10 Christian                 689
## 11 None                     3523
## 12 Catholic                 5124
## 13 Protestant              10846
```

Merging data

Data at different levels

Our data ideally should be tidy but reality is often more complex

Data can come from different sources

Information can be at different levels:
- individual
- household
- relationship
- area

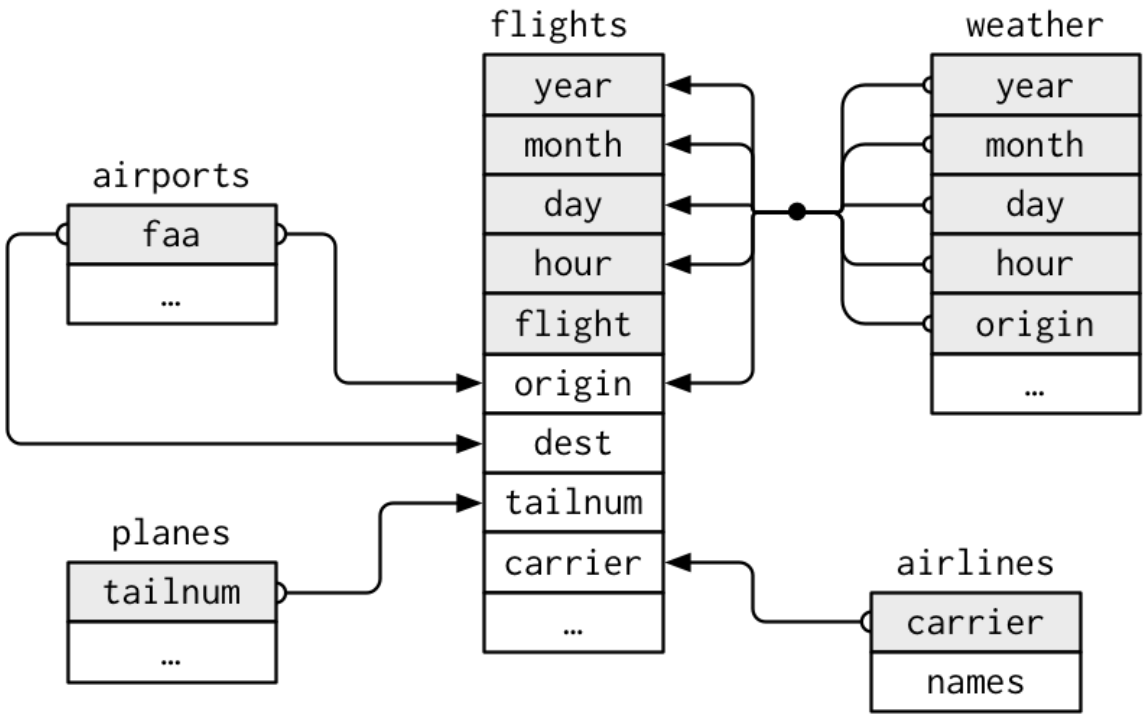
Using keys

Keys uniquely identify cases
at different levels

Primary key
identify cases in it's own table

Foreign key
identify cases in another table

Example from the flights data



Checking if keys are unique in table

```
count(planes, tailnum)
```

```
## # A tibble: 3,322 x 2
##    tailnum     n
##  * <chr>   <int>
##  1 N10156      1
##  2 N102UW      1
##  3 N103US      1
##  4 N104UW      1
##  5 N10575      1
##  6 N105UW      1
##  7 N107US      1
##  8 N108UW      1
##  9 N109UW      1
## 10 N110UW      1
## # ... with 3,312 more rows
```

Checking if keys are unique in table

```
count(planes, tailnum) %>%
  filter(n > 1)
```

```
## # A tibble: 0 x 2
## # ... with 2 variables: tailnum <chr>, n <int>
```

Checking if keys are unique in table

```
count(weather, year, month, day, hour, origin) %>%
  filter(n > 1)
```

```
## # A tibble: 3 x 6
##    year month   day  hour origin     n
##   <int> <int> <int> <int> <chr>  <int>
## 1  2013    11     3     1 EWR        2
## 2  2013    11     3     1 JFK        2
## 3  2013    11     3     1 LGA        2
```

When no key is present

When no key is present

```
count(flights, year, month, day, flight) %>%
  filter(n > 1)
```

```
## # A tibble: 29,768 x 5
##     year month   day flight     n
##    <int> <int> <int>  <int> <int>
##  1  2013     1     1      1     2
##  2  2013     1     1      3     2
##  3  2013     1     1      4     2
##  4  2013     1     1     11     3
##  5  2013     1     1     15     2
##  6  2013     1     1     21     2
##  7  2013     1     1     27     4
##  8  2013     1     1     31     2
##  9  2013     1     1     32     2
## 10  2013     1     1     35     2
## # ... with 29,758 more rows
```

Creating a key

```
flights <- mutate(flights, flight_id = row_number()) %>%
  select(flight_id, everything())
```

```
## # A tibble: 336,776 x 20
##    flight_id  year month   day dep_time sched_dep_time dep_de
##        <int> <int> <int> <int>    <int>          <int>     <d
##  1         1  2013     1     1      517            515
##  2         2  2013     1     1      533            529
##  3         3  2013     1     1      542            540
##  4         4  2013     1     1      544            545
##  5         5  2013     1     1      554            600
##  6         6  2013     1     1      554            558
##  7         7  2013     1     1      555            600
##  8         8  2013     1     1      557            600
##  9         9  2013     1     1      557            600
## 10        10  2013     1     1      558            600
## # ... with 336,766 more rows, and 12 more variables: sched_ar
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <ch
## #   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, m
## #   time_hour <dttm>
```

Different types of joins in R

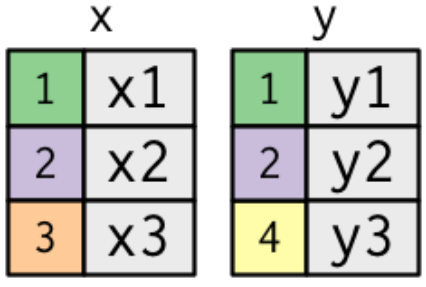
Mutating joins

Combine variables from two datasets

Match tables by key and
copy variables from on table to another

Conceptual representation of join

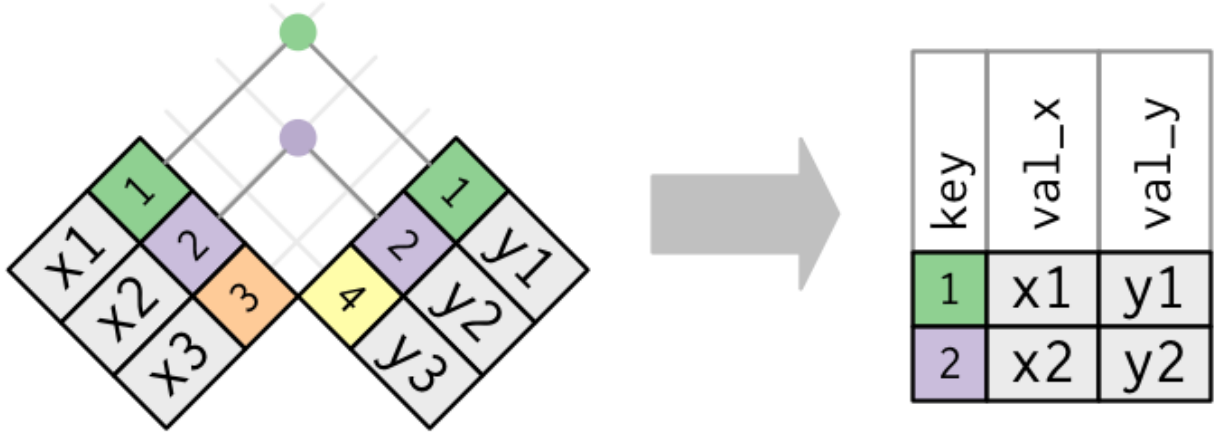
```
x <- tribble(
  ~key,~val_x,
     1, "x1",
     2, "x2",
     3, "x3"
)

y <- tribble(
  ~key, ~val_y,
     1, "y1",
     2, "y2",
     4, "y3"
)
```



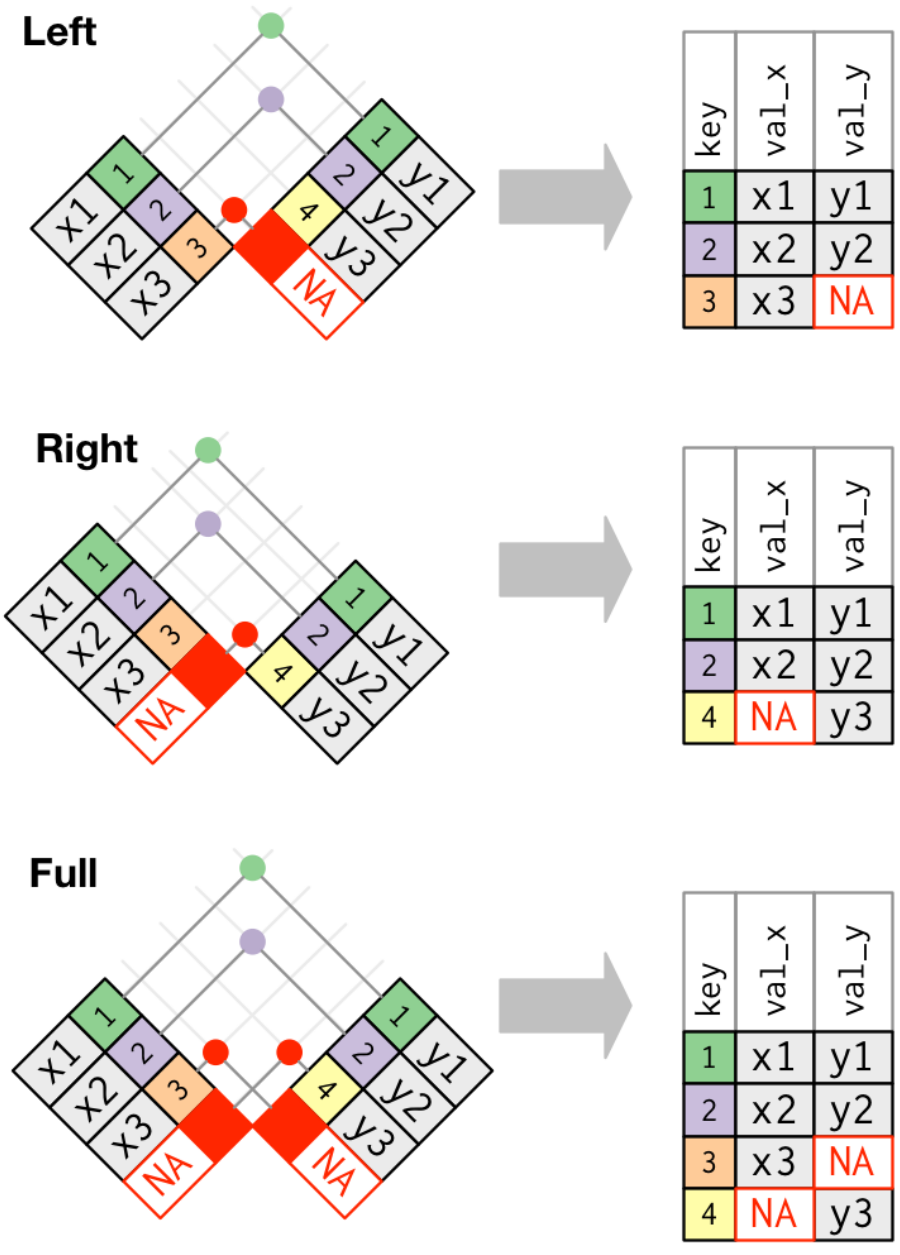
Inner join

```
inner_join(x, y, by = "key")
```

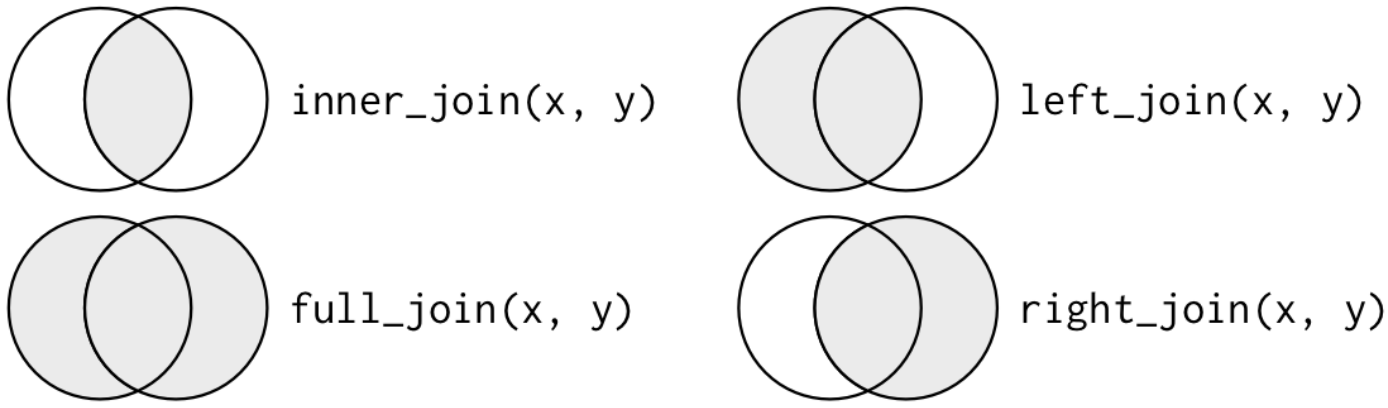
```
## # A tibble: 2 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
```



Outer joins diagram



Outer joins Venn diagram



Outer joins

```
left_join(x, y, by = "key")
```

```
## # A tibble: 3 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
## 3     3 x3    <NA>
```

```
right_join(x, y, by = "key")
```

```
## # A tibble: 3 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
## 3     4 <NA>  y3
```

Full join

```
full_join(x, y, by = "key")
```

```
## # A tibble: 4 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
## 3     3 x3    <NA>
## 4     4 <NA>  y3
```

Different names for keys

```
# add info to destination airports
left_join(flights2, airports, c("dest" = "faa"))

# add info to origin airports
left_join(flights2, airports, c("origin" = "faa"))
```

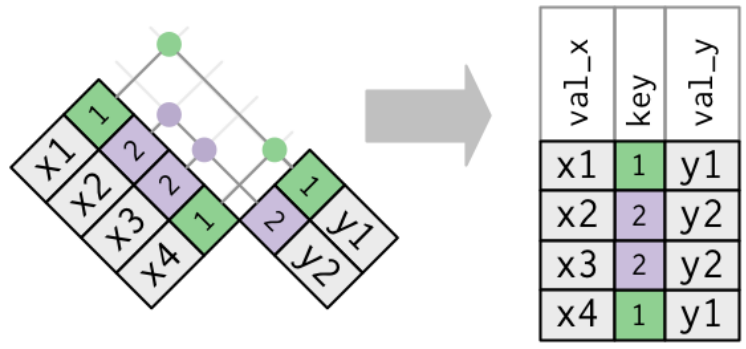
Merging complex data

In order to use complex data we have to merge different types of information

Can be of different types:

- 1:1
- 1:m
- m:m

Joining one to many



val_x	key	val_y
x1	1	y1
x2	2	y2
x3	2	y2
x4	1	y1

Joining one to many

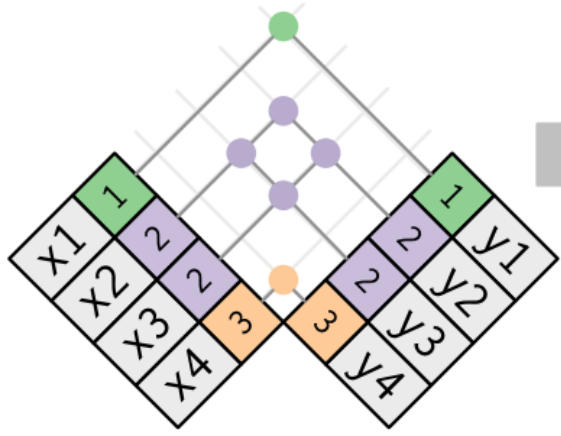
```
x <- tribble(
  ~key, ~val_x,
     1, "x1",
     2, "x2",
     2, "x3",
     1, "x4"
)
y <- tribble(
  ~key, ~val_y,
     1, "y1",
     2, "y2"
)
```

Joining one to many

```
left_join(x, y, by = "key")
```

```
## # A tibble: 4 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
## 3     2 x3    y2
## 4     1 x4    y1
```

Joining many to many



key	val_x	val_y
1	x1	y1
2	x2	y2
2	x2	y3
2	x3	y2
2	x3	y3
3	x4	y4

Joining many to many

```
x <- tribble(
  ~key, ~val_x,
     1, "x1",
     2, "x2",
     2, "x3",
     3, "x4"
)
y <- tribble(
  ~key, ~val_y,
     1, "y1",
     2, "y2",
     2, "y3",
     3, "y4"
)
```

Joining many to many

```
left_join(x, y, by = "key")
```

```
## # A tibble: 6 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
## 3     2 x2    y3
## 4     2 x3    y2
## 5     2 x3    y3
## 6     3 x4    y4
```

Issues with joining data

Understand the key variables

Check primary keys are unique
(and not missing)

Check foreign keys match primary keys
(use `anti_join()`)

Practical 1