# R Lab3: Data Management

**Haocheng Hu and Zhaopeng Qu**

**Nanjing University**

*3/12/2025*

Section 1

# Data Management in R

Subsection 1

## Introduction

# Common Types Of Variable in Statistics

- **Quantitative**(定量)
  - Income,Testscore,Weight,Height
- **Qualitative**(定性)
  - Gender, Nationality, Identity, Profession...
- **Date**(时间)
  - day,month,year,lunar calendar

# Types in Qualitative(定性)

- Qualitative (定性)
  - **categoried(nominal) variable** 分类变量
    - 性别、民族、行业...
  - **ordered(ordinal) variable** 有序分类变量
    - 教育水平、自评健康、幸福感

# Data Types in R

- **Numeric Data**
  - integer: only store whole numbers: `2L`
  - double: floating point numbers (i.e. with a decimal part)
- **Character Data**: store strings of characters.
  - character:
  - factor:
- **Dates**
  - Date: the number of days
  - POSIXct: the number of seconds
- **Logical**: store `TRUE` or `FALSE`
  - TRUR==1 and FALSE==0

# Data Structure in R

- Scalars(标量)
- Vectors(向量)
- Matrix(矩阵)
- Data.frames(数据框)
- List(列表)
- Array(数组)

# Vectors(向量)

- All elements in a vector are the same type.
  - c(1,2,3,4)
  - c("R","Stata","Python")
- **Factor Vectors**

```
q<-c("soccer","badminton","soccer",
     "tennis","soccer","tennis")
q
```

```
## [1] "soccer"    "badminton" "soccer"    "tennis"    "soccer
```

- transform into a **factor vector**

```
q.factor <- as.factor(q)
q.factor
```

```
## [1] soccer    badminton soccer    tennis    soccer    tenni
```

# Data.frames(数据框)

- `data.frame` is just like an **Excel spreadsheet** in that it has columns and rows.
- Each column is a *variable* and each row is an *observation*.
- A `data.frame` can be seen as a collection of **vectors**.

# Data.frames(数据框)

- generate a data.frame

```
x  <-  6:1
y  <- -4:1
df <- data.frame(x,y,q)
df
```

```
##   x  y         q
## 1 6 -4    soccer
## 2 5 -3 badminton
## 3 4 -2    soccer
## 4 3 -1    tennis
## 5 2  0    soccer
## 6 1  1    tennis
```

# Data.frames(数据框)

- naming variables

```
df <- data.frame(Frist=x,Second=y,Sport=q)
df
```

```
##   Frist Second     Sport
## 1     6     -4    soccer
## 2     5     -3 badminton
## 3     4     -2    soccer
## 4     3     -1    tennis
## 5     2      0    soccer
## 6     1      1    tennis
```

# List(列表)

- A **list** is a container which can contain all *numerics* or *characters* elements or a mix of the two or `date.frames` or other `lists`.

```
list(1,2,3)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] 3
```

```
list(c(1,2,3))
```

```
## [[1]]
## [1] 1 2 3
```

# List(列表)

- combination of a data.frame and a vector

```
list(df,1:6)
```

```
## [[1]]
##   Frist Second     Sport
## 1     6     -4    soccer
## 2     5     -3 badminton
## 3     4     -2    soccer
## 4     3     -1    tennis
## 5     2      0    soccer
## 6     1      1    tennis
##
## [[2]]
## [1] 1 2 3 4 5 6
```

# Conversion functions

- `is.datatype()` return TRUE or FALSE
- `as.datatype()` converts the argument to that type

| Test | Convert |
|---|---|
| `is.numeric()` | `as.numeric()` |
| `is.character()` | `as.character()` |
| `is.vector()` | `as.vector()` |
| `is.matrix()` | `as.matrix()` |
| `is.data.frame()` | `as.data.frame()` |
| `is.factor()` | `as.factor()` |
| `is.logical()` | `as.logical()` |

Subsection 2

# Preparation for Data Analysis

# Working directory

- Working in an appropriate directory.

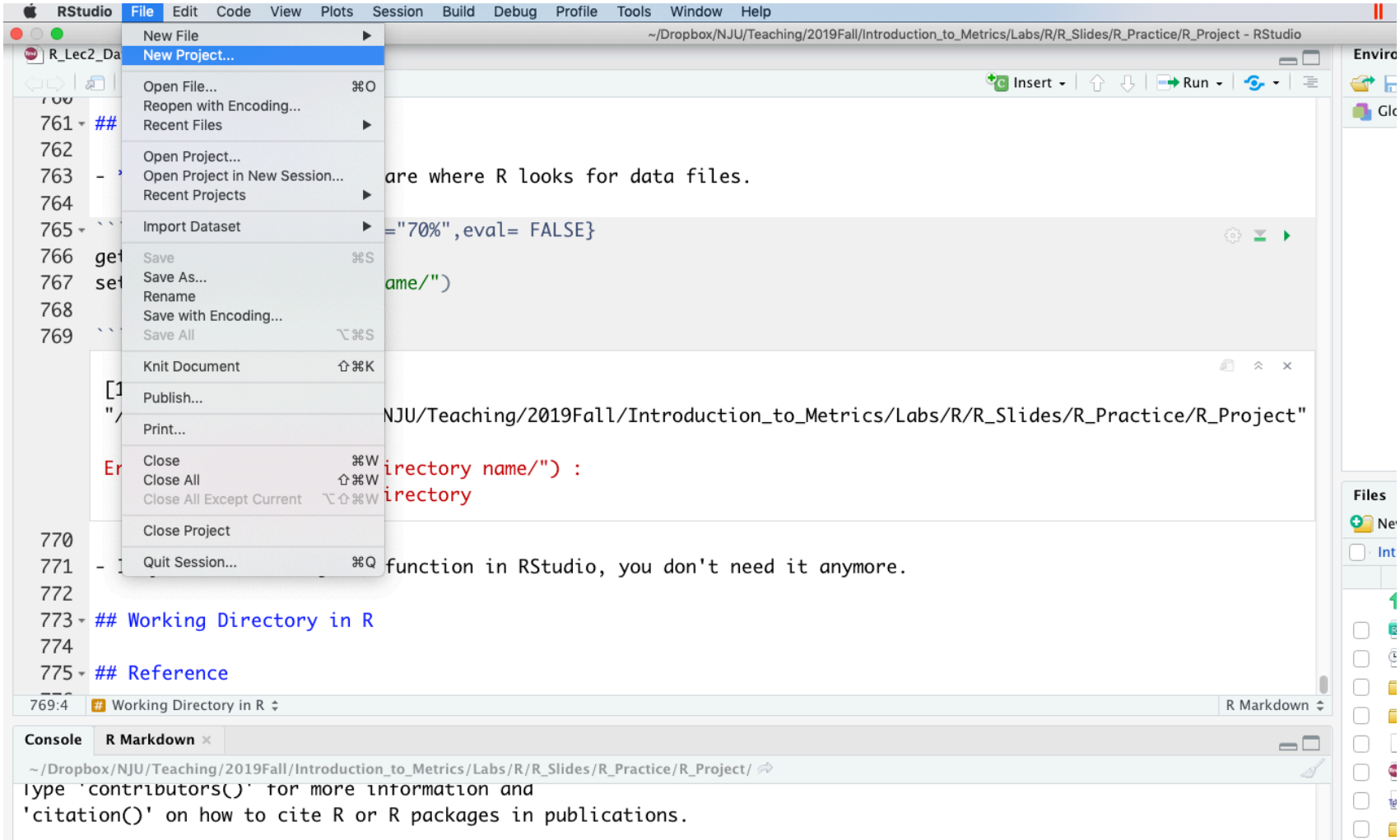
```
getwd()
setwd("D:/Documents/Rlab")
```

- Clean out your workspace
  - `Session -> Clear`

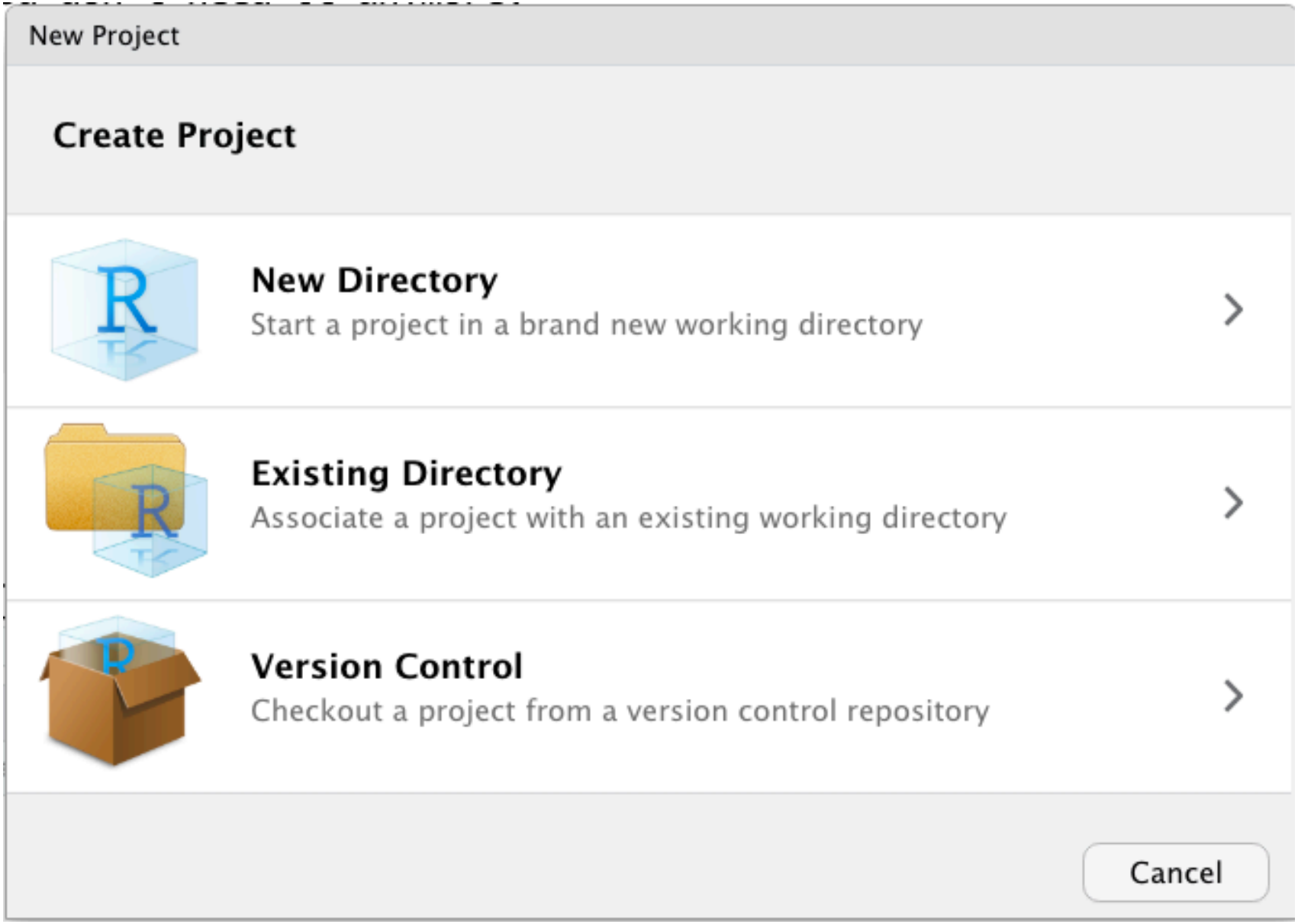
```
rm(list=ls())
```

- `Restart R` and open a new `RScript` file.
- If you use the Projects function in RStudio, you don't need it anymore.

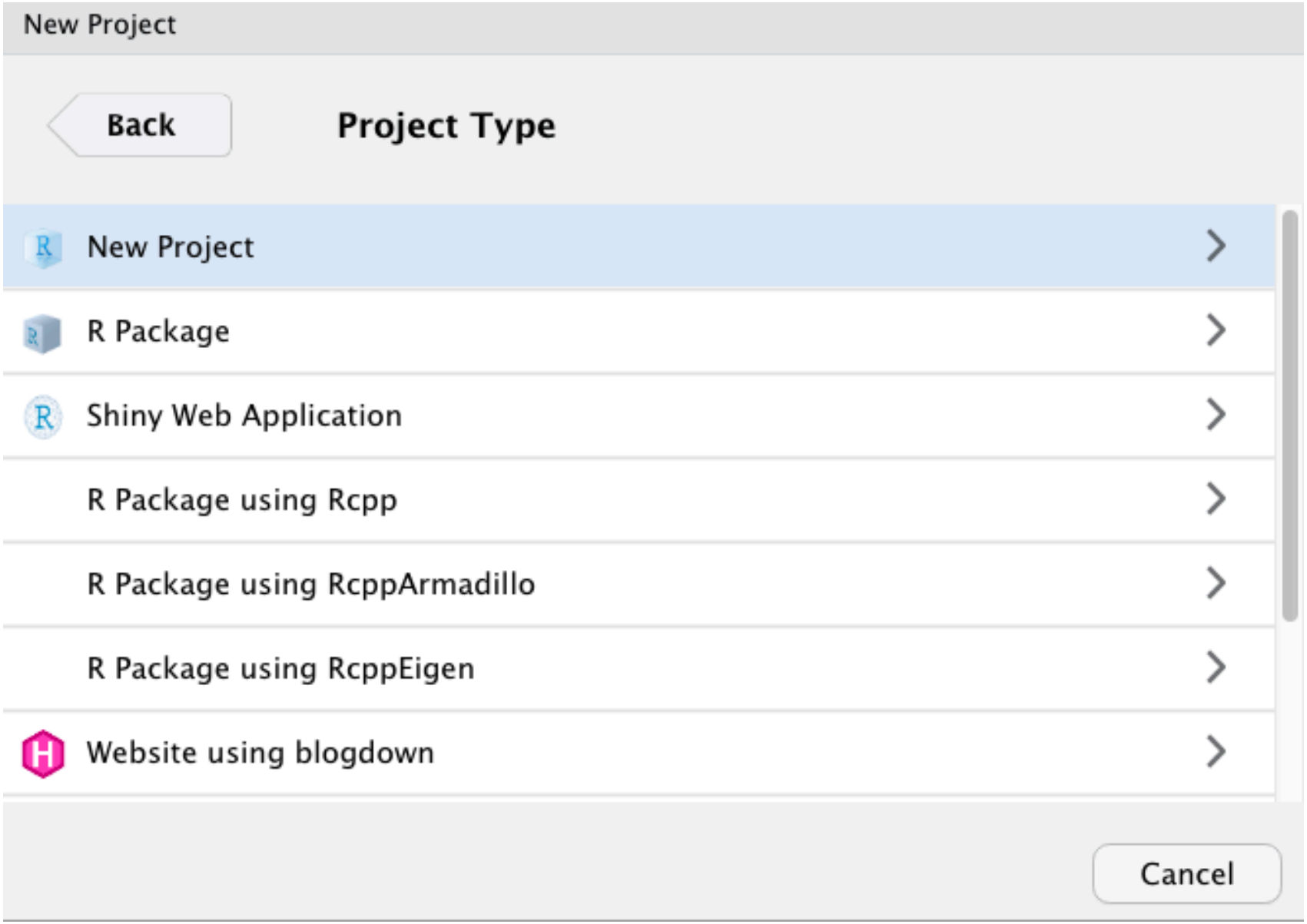
# Projects function in RStudio



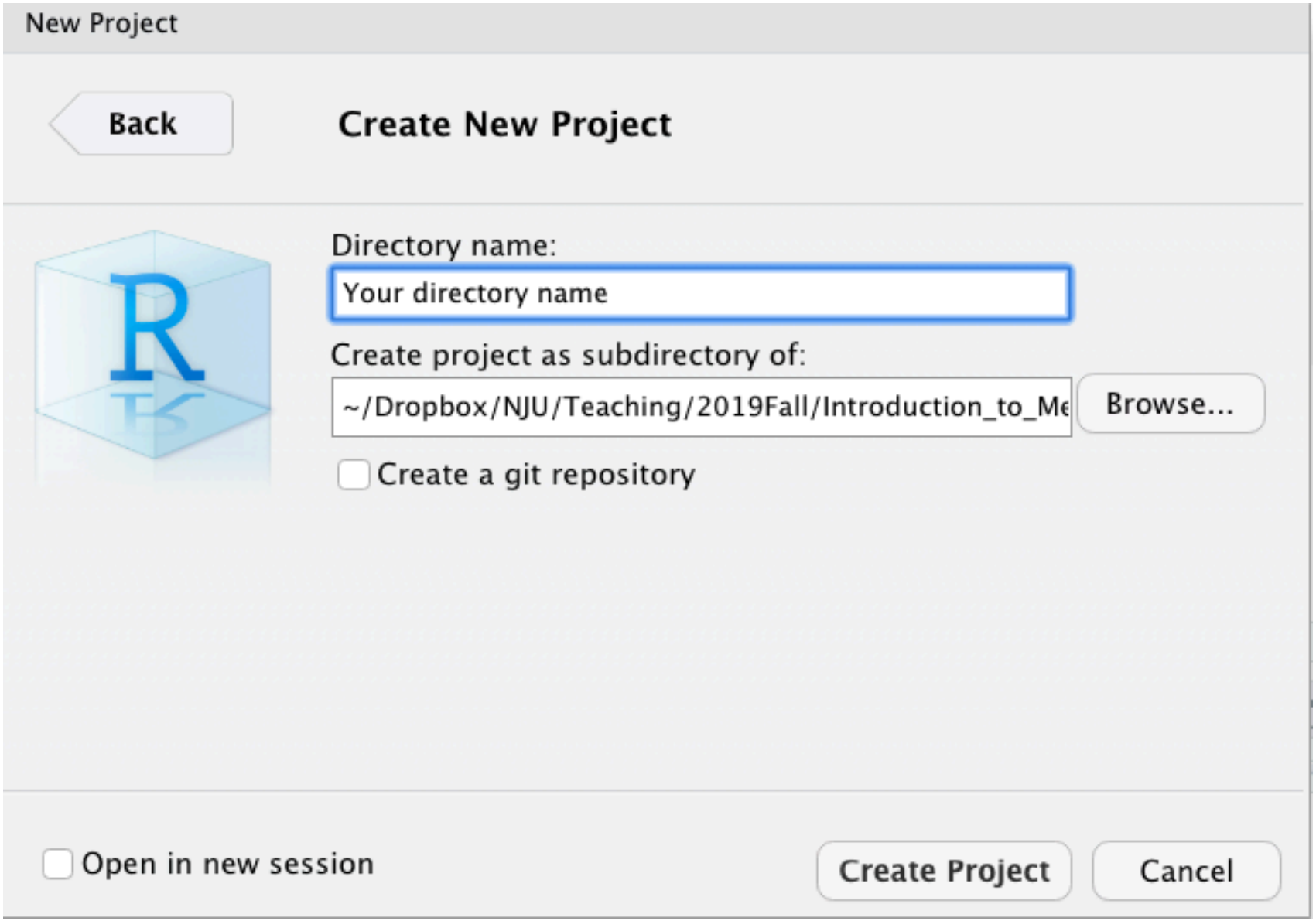
# Projects function in RStudio



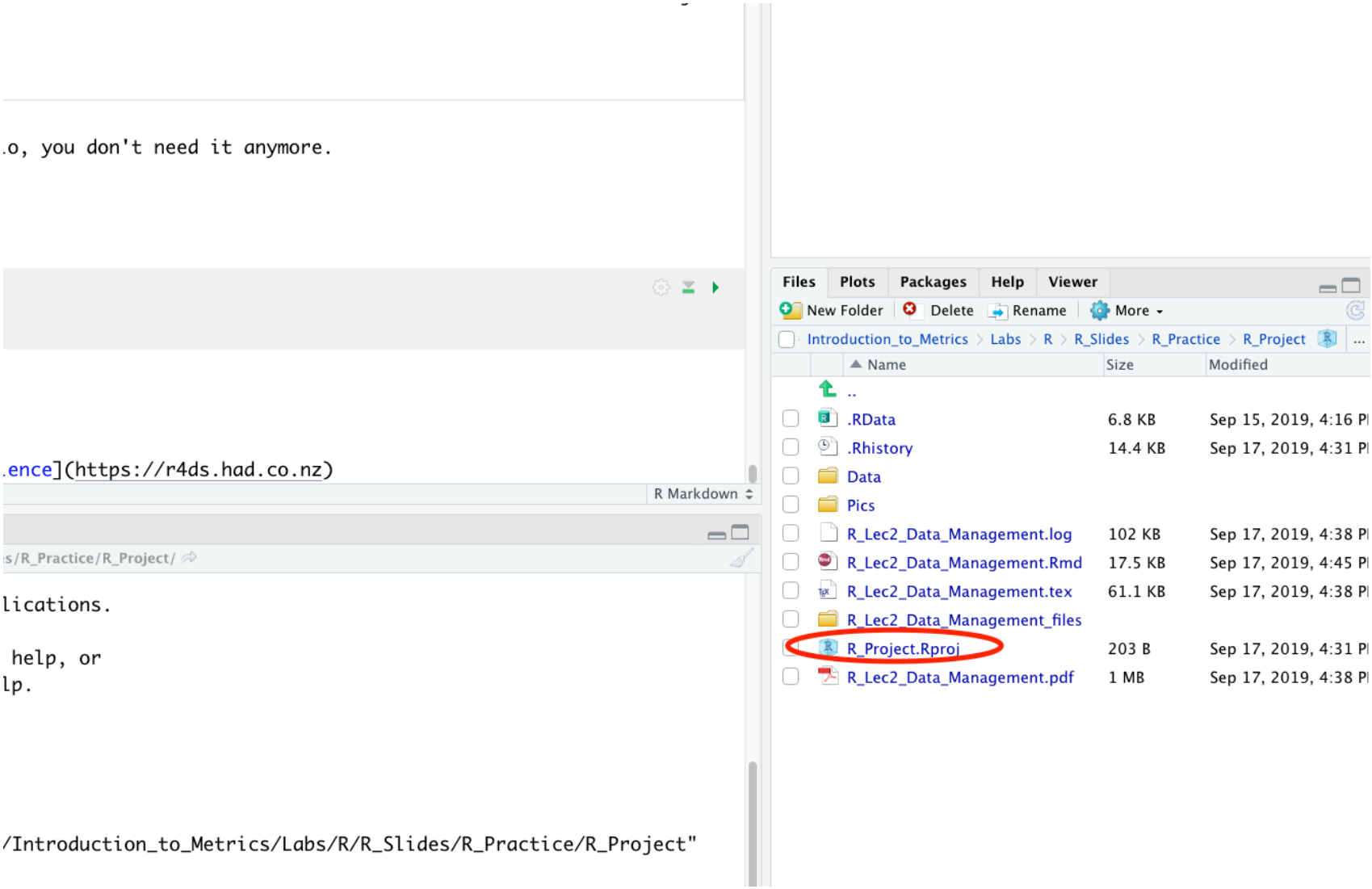
# Projects function in RStudio



# Projects function in RStudio



# Projects function in RStudio



# Section 2

# "Fancy" Data Management using Tidyverse

Subsection 1

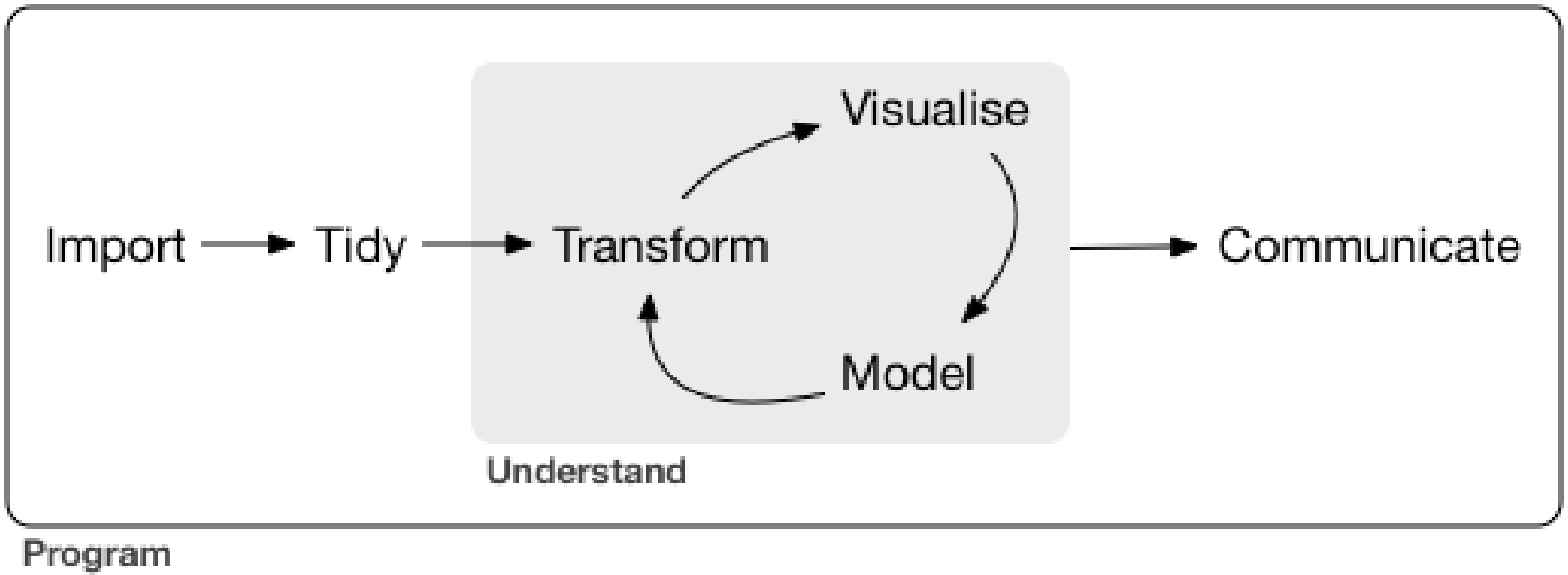
# Introduction to tidydata

# Introduction

- A set of **packages** that are useful specifically for *data manipulation*, *exploration* and *visualization* with a common philosophy: *"tidy"* data.
- Developed by a leading figure in R field: **Hadley Wickham**

# Introduction

- The **workflow** of data analysis is as follows



# Install and Library tidyverse

- We can install and load the set of R packages using `install.packages("tidyverse")` function.

```
library(tidyverse) ##
```

```
## Warning: package 'tidyverse' was built under R version 4.4.
## Warning: package 'readr' was built under R version 4.4.3
## Warning: package 'forcats' was built under R version 4.4.3
## Warning: package 'lubridate' was built under R version 4.4.
```

# Install and Library tidyverse

- When we load the tidyverse package using `library(tidyverse)`, there are **six** core R packages that load:
  - `readr` and `haven`, for **data import**.
  - `tidyr`, for **data tidying**.
  - `dplyr`, for **data wrangling**.
  - `ggplot2`, for **data visualization**.
  - `purrr`, for functional programming.
  - `tibble`, for tibbles, a modern re-imagining of data frames.

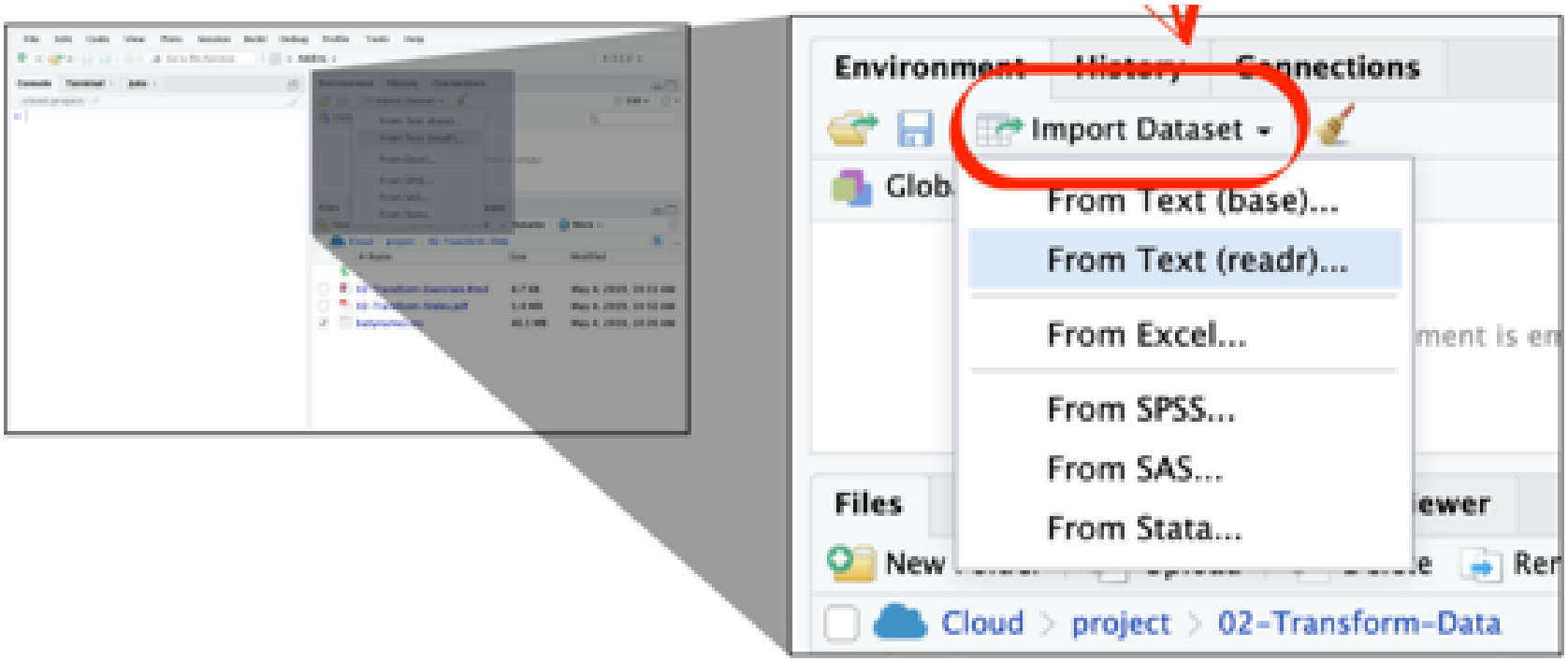
Subsection 2

# Import Data

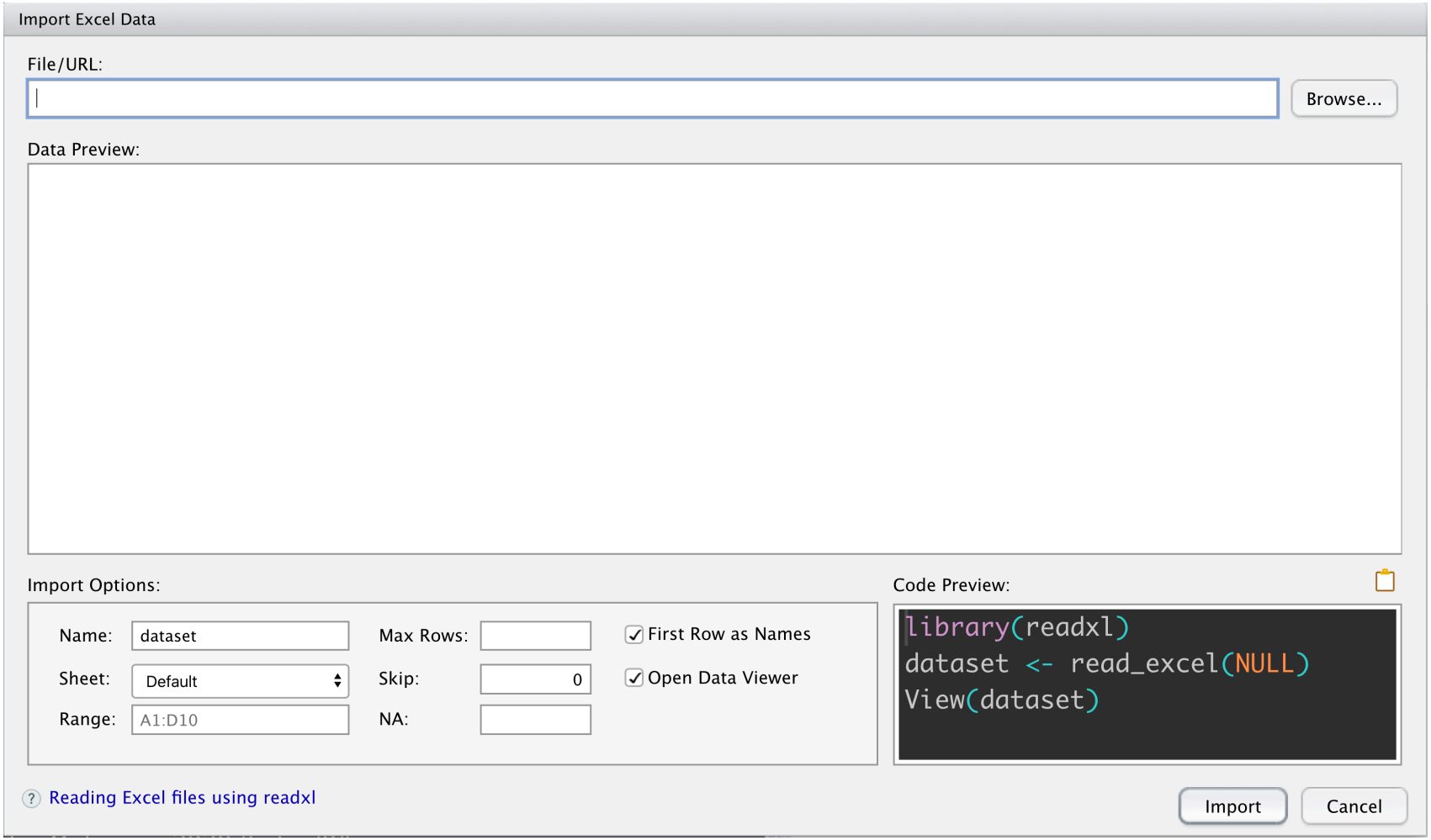
# Import data by readr and haven

- read_csv():comma separated (CSV) files
- read_tsv():tab separated files
- read_delim(): delimited files
- read_table():tabular files where columns are separated by white-space.
- read_log():web log files
- read_sas() : reads .sas7bdat + .sas7bcat files
- read_spss(): reads .sav files of SPSS
- read_stata() : reads .dta files (up to version 15)

# Import data by readr and haven



# Import data by readxl



Subsection 3

# "Tidy" Data

# What does tidy data mean?

- Structure
  - An easy to deal and clear form for data analysis
- Observation quality:
  - Missing values
  - Duplicates
  - Outliers
- Variables
  - Selecting and Generate variables

# Which data is tidy?

```
table1
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <dbl>  <dbl>      <dbl>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

```
table2
```

```
## # A tibble: 12 x 4
##    country      year type            count
##    <chr>       <dbl> <chr>           <dbl>
##  1 Afghanistan  1999 cases             745
##  2 Afghanistan  1999 population   19987071
##  3 Afghanistan  2000 cases            2666
##  4 Afghanistan  2000 population   20595360
```

# Which data is tidy?

```
table3
```

```
## # A tibble: 6 x 3
##   country      year rate
##   <chr>       <dbl> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
## 4 Brazil       2000 80488/174504898
## 5 China        1999 212258/1272915272
## 6 China        2000 213766/1280428583
```

# Which data is tidy?

```
table4a
```

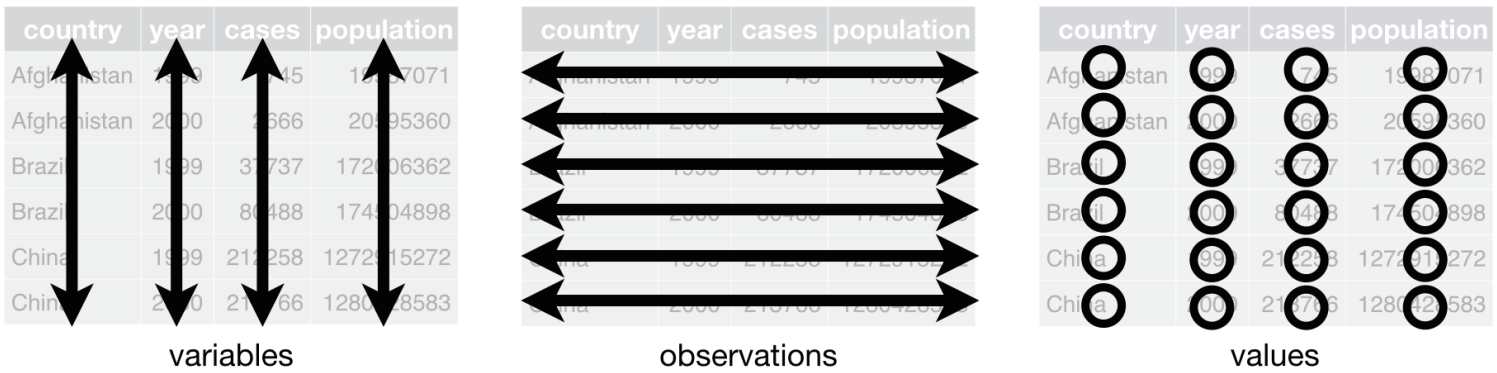
```
## # A tibble: 3 x 3
##   country     `1999` `2000`
##   <chr>        <dbl>  <dbl>
## 1 Afghanistan    745   2666
## 2 Brazil       37737  80488
## 3 China       212258 213766
```

```
table4b
```

```
## # A tibble: 3 x 3
##   country         `1999`     `2000`
##   <chr>            <dbl>      <dbl>
## 1 Afghanistan   19987071   20595360
## 2 Brazil       172006362  174504898
## 3 China       1272915272 1280428583
```

# a tidy data strcture

- Each variable forms a column.
- Each observation forms a row.
- Each type of observational unit forms a table.



Subsection 4

## Reconstruct data into tidy data

# tidyr package

- tidyr is a one such package which was built for the sole purpose of simplifying the process of creating tidy data.
- Four fundamental functions of data tidying that tidyr provides:
  - gather() makes "wide" data longer
  - spread() makes "long" data wider
  - separate() splits a character column into multiple columns
  - unite() combines multiple columns into a single column

# tidy data using spread()

```
table1
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <dbl>  <dbl>      <dbl>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

```
table2
```

```
## # A tibble: 12 x 4
##    country      year type            count
##    <chr>       <dbl> <chr>           <dbl>
##  1 Afghanistan  1999 cases             745
##  2 Afghanistan  1999 population   19987071
##  3 Afghanistan  2000 cases            2666
##  4 Afghanistan  2000 population   20595360
```

# tidy data using spread()

```
library(tidyr)
spread(table2,type,count) #or
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <dbl>  <dbl>      <dbl>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

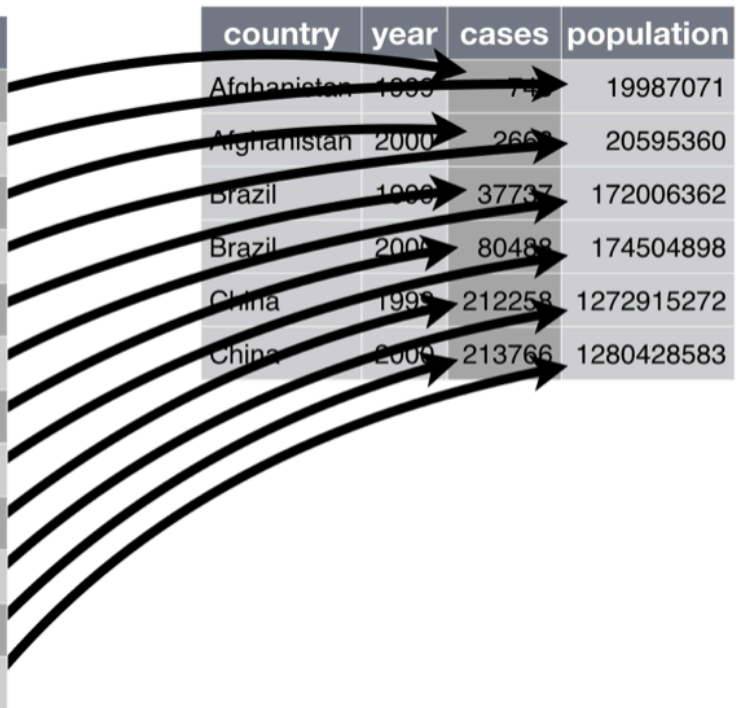
```
pivot_wider(table2,names_from=type,values_from=count)
```

```
## # A tibble: 6 x 4
##   country      year  cases population
```

# tidy data using spread()

| country | year | key | value |
|---|---|---|---|
| Afghanistan | 1999 | cases | 745 |
| Afghanistan | 1999 | population | 19987071 |
| Afghanistan | 2000 | cases | 2666 |
| Afghanistan | 2000 | population | 20595360 |
| Brazil | 1999 | cases | 37737 |
| Brazil | 1999 | population | 172006362 |
| Brazil | 2000 | cases | 80488 |
| Brazil | 2000 | population | 174504898 |
| China | 1999 | cases | 212258 |
| China | 1999 | population | 1272915272 |
| China | 2000 | cases | 213766 |
| China | 2000 | population | 1280428583 |

table2



# tidy data using gather()

```
table1
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <dbl>  <dbl>      <dbl>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

```
table4a
```

```
## # A tibble: 3 x 3
##   country     `1999` `2000`
##   <chr>        <dbl>  <dbl>
## 1 Afghanistan    745   2666
## 2 Brazil       37737  80488
## 3 China       212258 213766
```

# tidy data using gather()

```
gather(table4a,"year","cases",2:3) #or
```

```
## # A tibble: 6 x 3
##   country     year   cases
##   <chr>       <chr>  <dbl>
## 1 Afghanistan 1999     745
## 2 Brazil      1999   37737
## 3 China       1999  212258
## 4 Afghanistan 2000    2666
## 5 Brazil      2000   80488
## 6 China       2000  213766
```

```
pivot_longer(table4a,2:3,names_to="year",values_to="cases")
```

```
## # A tibble: 6 x 3
##   country     year   cases
##   <chr>       <chr>  <dbl>
```

# tidy data using gather()

| country | year | cases |
|---|---|---|
| Afghanistan | 1999 | 745 |
| Afghanistan | 2000 | 2666 |
| Brazil | 1999 | 37737 |
| Brazil | 2000 | 80488 |
| China | 1999 | 212258 |
| China | 2000 | 213766 |

| country | 1999 | 2000 |
|---|---|---|
| Afghanistan | 745 | 2666 |
| Brazil | 37737 | 80488 |
| China | 212258 | 213766 |

table4

# tidy data using seprate()

```
table1
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <dbl>  <dbl>      <dbl>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

```
table3
```

```
## # A tibble: 6 x 3
##   country      year rate
##   <chr>       <dbl> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
## 4 Brazil       2000 80488/174504898
```

# tidy data using seprate()

```
separate(table3,rate,into = c("cases","population"),sep="/")
```

```
## # A tibble: 6 x 4
##   country      year cases  population
##   <chr>       <dbl> <chr>  <chr>
## 1 Afghanistan  1999 745    19987071
## 2 Afghanistan  2000 2666   20595360
## 3 Brazil       1999 37737  172006362
## 4 Brazil       2000 80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

# tidy data using unite()

```
table5 <- separate(table3,rate,into = c("cases","population"),
```

```
unite(table5,"rate",cases,population,sep="/")
```

```
## # A tibble: 6 x 3
##   country      year rate
##   <chr>       <dbl> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
## 4 Brazil       2000 80488/174504898
## 5 China        1999 212258/1272915272
## 6 China        2000 213766/1280428583
```

Subsection 5

# Data Cleaning

# Review: tidy data

- Data Structure Transforming
- Data Cleaning:
  - Dealing with missing values and duplicates
- Data Relating:
  - Merge or join multiple data
- Data Wrangling:
  - Selecting and Creating Variables
  - Selecting observations

Subsection 6

# Data Wrangling:

# Data Wrangling: using dplyr

- the most time-consuming and dirty work by the package dplyr

The important dplyr verbs to remember are:

| dplyr verbs | Description |
|---|---|
| select() | select columns |
| filter() | filter rows |
| arrange() | re-order or arrange rows |
| mutate() | create new columns |
| rename() | rename columns |
| summarize() | summarize values |
| group_by() | allows for group operations in the "split-apply-combine" concept |

# Pipe operator: %>%

- This operator allows you to pipe the output from one function to the input of another function.
- Instead of nesting functions (reading from the inside to the outside), the idea of of piping is to read the functions from left to right.

```
## Warning: package 'gapminder' was built under R version 4.4.
```

# Selecting columns using select()

- To select all the columns *except* a specific column, use the "-" (subtraction) operator. (drop a variable)

```
gapminder %>%
  select(-lifeExp)
```

```
## # A tibble: 1,704 x 5
##    country     continent  year      pop gdpPercap
##    <fct>       <fct>     <int>    <int>     <dbl>
##  1 Afghanistan Asia       1952  8425333      779.
##  2 Afghanistan Asia       1957  9240934      821.
##  3 Afghanistan Asia       1962 10267083      853.
##  4 Afghanistan Asia       1967 11537966      836.
##  5 Afghanistan Asia       1972 13079460      740.
##  6 Afghanistan Asia       1977 14880372      786.
##  7 Afghanistan Asia       1982 12881816      978.
##  8 Afghanistan Asia       1987 13867957      852.
```

# Selecting columns using select()

- To select a range of columns by name, use the ":" (colon) operator

```
gapminder %>%
  select(lifeExp:gdpPercap)
```

```
## # A tibble: 1,704 x 3
##    lifeExp      pop gdpPercap
##      <dbl>    <int>     <dbl>
##  1    28.8  8425333      779.
##  2    30.3  9240934      821.
##  3    32.0 10267083      853.
##  4    34.0 11537966      836.
##  5    36.1 13079460      740.
##  6    38.4 14880372      786.
##  7    39.9 12881816      978.
##  8    40.8 13867957      852.
##  9    41.7 16317921      649.
```

# Additional options to select()

| selection function | meaning |
|---|---|
| starts_with("c") | Select columns that start with a character string "c" |
| ends_with() | Select columns that end with a character string |
| contains() | Select columns that contain a character string |
| matches() | Select columns that match a regular expression |
| one_of() | Select columns names that are from a group of names |

# Selecting rows using filter()

- only want the data from 2007.

```
gapminder %>%
  filter(year == 2007)
```

```
## # A tibble: 142 x 6
##    country     continent  year lifeExp      pop gdpPercap
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Afghanistan Asia       2007    43.8 31889923      975.
##  2 Albania     Europe     2007    76.4  3600523     5937.
##  3 Algeria     Africa     2007    72.3 33333216     6223.
##  4 Angola      Africa     2007    42.7 12420476     4797.
##  5 Argentina   Americas   2007    75.3 40301927    12779.
##  6 Australia   Oceania    2007    81.2 20434176    34435.
##  7 Austria     Europe     2007    79.8  8199783    36126.
##  8 Bahrain     Asia       2007    75.6   708573    29796.
##  9 Bangladesh  Asia       2007    64.1 150448339     1391.
```

# Selecting rows using `filter()`

- only want the data from China.

```
gapminder %>%
  filter(country == "China" ) ## String can be used too ""
```

```
## # A tibble: 12 x 6
##    country continent  year lifeExp      pop gdpPercap
##    <fct>   <fct>     <int>   <dbl>    <int>     <dbl>
##  1 China   Asia       1952    44   556263527      400.
##  2 China   Asia       1957    50.5 637408000      576.
##  3 China   Asia       1962    44.5 665770000      488.
##  4 China   Asia       1967    58.4 754550000      613.
##  5 China   Asia       1972    63.1 862030000      677.
##  6 China   Asia       1977    64.0 943455000      741.
##  7 China   Asia       1982    65.5 1000281000     962.
##  8 China   Asia       1987    67.3 1084035000    1379.
##  9 China   Asia       1992    68.7 1164970000    1656.
```

# Selecting rows using filter()

- use the boolean operators (e.g. >, <, >=, <=, !=, %in%) to create logical tests.For example, if we wanted only years after 1977,then

```
gapminder %>%
  filter(country == "China",year > 1977)
```

```
## # A tibble: 6 x 6
##   country continent  year lifeExp       pop gdpPercap
##   <fct>   <fct>     <int>   <dbl>     <int>     <dbl>
## 1 China   Asia       1982    65.5 1000281000      962.
## 2 China   Asia       1987    67.3 1084035000     1379.
## 3 China   Asia       1992    68.7 1164970000     1656.
## 4 China   Asia       1997    70.4 1230075000     2289.
## 5 China   Asia       2002    72.0 1280400000     3119.
## 6 China   Asia       2007    73.0 1318683096     4959.
```

# Use mutate() to add new variables

- mutate() is a function that defines and inserts new variables into a tibble. You can refer to existing variables by name.

```
gapminder %>%
    mutate(gdp = pop * gdpPercap)
```

```
## # A tibble: 1,704 x 7
##    country     continent  year lifeExp      pop gdpPercap          gdp
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>        <dbl>
##  1 Afghanistan Asia       1952    28.8  8425333      779.  6567086330.
##  2 Afghanistan Asia       1957    30.3  9240934      821.  7585448670.
##  3 Afghanistan Asia       1962    32.0 10267083      853.  8758855797.
##  4 Afghanistan Asia       1967    34.0 11537966      836.  9648014150.
##  5 Afghanistan Asia       1972    36.1 13079460      740.  9678553274.
##  6 Afghanistan Asia       1977    38.4 14880372      786. 11697659231.
##  7 Afghanistan Asia       1982    39.9 12881816      978. 12598563401.
##  8 Afghanistan Asia       1987    40.8 13867957      852. 11820990309.
##  9 Afghanistan Asia       1992    41.7 16317921      649. 10595901589.
## 10 Afghanistan Asia       1997    41.8 22227415      635. 14121995875.
## # i 1,694 more rows
```

# Use mutate() to add new variables

```
gapminder %>%
    filter(year==2007) %>% ## only data in 2007
    select(-continent) %>% ## drop continent variable
    mutate(gdp = pop * gdpPercap) ## generate a new variable "gdp"
```

```
## # A tibble: 142 x 6
##    country      year lifeExp      pop gdpPercap          gdp
##    <fct>       <int>   <dbl>    <int>     <dbl>        <dbl>
##  1 Afghanistan  2007    43.8 31889923      975.  31079291949.
##  2 Albania      2007    76.4  3600523     5937.  21376411360.
##  3 Algeria      2007    72.3 33333216     6223. 207444851958.
##  4 Angola       2007    42.7 12420476     4797.  59583895818.
##  5 Argentina    2007    75.3 40301927    12779. 515033625357.
##  6 Australia    2007    81.2 20434176    34435. 703658358894.
##  7 Austria      2007    79.8  8199783    36126. 296229400691.
##  8 Bahrain      2007    75.6   708573    29796.  21112675360.
##  9 Bangladesh   2007    64.1 150448339     1391. 209311822134.
## 10 Belgium      2007    79.4 10392226    33693. 350141166520.
## # i 132 more rows
```

# Use rename() to change the name of variables

```
gapminder %>%
rename(life_exp = lifeExp,
         gdp_percap = gdpPercap)
```

```
## # A tibble: 1,704 x 6
##    country     continent  year life_exp      pop gdp_percap
##    <fct>       <fct>     <int>    <dbl>    <int>      <dbl>
##  1 Afghanistan Asia       1952     28.8  8425333       779.
##  2 Afghanistan Asia       1957     30.3  9240934       821.
##  3 Afghanistan Asia       1962     32.0 10267083       853.
##  4 Afghanistan Asia       1967     34.0 11537966       836.
##  5 Afghanistan Asia       1972     36.1 13079460       740.
##  6 Afghanistan Asia       1977     38.4 14880372       786.
##  7 Afghanistan Asia       1982     39.9 12881816       978.
##  8 Afghanistan Asia       1987     40.8 13867957       852.
##  9 Afghanistan Asia       1992     41.7 16317921       649.
## 10 Afghanistan Asia       1997     41.8 22227415       635.
## # i 1,694 more rows
```

# Sort or re-order rows using arrange()

```
gapminder %>%
    filter(year==2007,continent=="Asia") %>%
    select(-c(continent,pop,gdpPercap)) %>%
    arrange(lifeExp)
```

```
## # A tibble: 33 x 3
##    country      year lifeExp
##    <fct>       <int>   <dbl>
##  1 Afghanistan  2007    43.8
##  2 Iraq         2007    59.5
##  3 Cambodia     2007    59.7
##  4 Myanmar      2007    62.1
##  5 Yemen, Rep.  2007    62.7
##  6 Nepal        2007    63.8
##  7 Bangladesh   2007    64.1
##  8 India        2007    64.7
##  9 Pakistan     2007    65.5
```

# Sort or re-order rows using arrange()

```
gapminder %>%
    filter(year==2007,continent=="Asia") %>%
    select(-c(continent,gdpPercap)) %>%
    arrange(desc(lifeExp)) ## descend order
```

```
## # A tibble: 33 x 4
##    country             year lifeExp      pop
##    <fct>              <int>   <dbl>    <int>
##  1 Japan               2007    82.6 127467972
##  2 Hong Kong, China    2007    82.2   6980412
##  3 Israel              2007    80.7   6426679
##  4 Singapore           2007    80.0   4553009
##  5 Korea, Rep.         2007    78.6  49044790
##  6 Taiwan              2007    78.4  23174294
##  7 Kuwait              2007    77.6   2505559
##  8 Oman                2007    75.6   3204897
##  9 Bahrain             2007    75.6    708573
```

# Summary Statistics by summarise() and group_by

```
gapminder %>%
    filter(year==2007) %>%
    group_by(continent,year) %>%
    summarize(gdpPercap_con = mean(gdpPercap)) %>%
    select(c(continent,gdpPercap_con,year)) %>%
    arrange(desc(gdpPercap_con)) ## descend order
```

```
## `summarise()` has grouped output by 'continent'. You can ov
## `.groups` argument.
```

```
## # A tibble: 5 x 3
## # Groups:   continent [5]
##   continent gdpPercap_con  year
##   <fct>             <dbl> <int>
## 1 Oceania          29810.  2007
## 2 Europe           25054.  2007
## 3 Asia             12473.  2007
```

# Summary Statistics by summarise() and group_by

```
gapminder %>%
    filter(continent=="Asia") %>%
    group_by(country) %>%
    summarize(gdpPercap_coun = mean(gdpPercap)) %>%
    arrange(desc(gdpPercap_coun)) ## descend order
```

```
## # A tibble: 33 x 2
##    country           gdpPercap_coun
##    <fct>                      <dbl>
##  1 Kuwait                    65333.
##  2 Saudi Arabia              20262.
##  3 Bahrain                   18078.
##  4 Japan                     17751.
##  5 Singapore                 17425.
##  6 Hong Kong, China          16229.
##  7 Israel                    14161.
```

# Summary Statistics by count()

```
gapminder %>%
    filter(year == 2002) %>%
    count(continent, sort = TRUE)
```

```
## # A tibble: 5 x 2
##   continent     n
##   <fct>     <int>
## 1 Africa       52
## 2 Asia         33
## 3 Europe       30
## 4 Americas     25
## 5 Oceania       2
```

# Let's have a exercise

- **Question**: Which country in which continent experienced the sharpest 5-year drop in life expectancy?

# Let's have a exercise

- **Question**: Which country in which continent experienced the sharpest 5-year drop in life expectancy?

```
gapminder %>%
  select(country, year, continent, lifeExp) %>%
  group_by(continent, country) %>%
  ## within country, take (lifeExp in year i) - (lifeExp in year i - 1)
  ## positive means lifeExp went up, negative means it went down
  mutate(le_delta = lifeExp - lag(lifeExp)) %>%
  ## within country, retain the worst lifeExp change = smallest or most neg
  summarize(worst_le_delta = min(le_delta, na.rm = TRUE)) %>%
  ## within continent, retain the row with the lowest worst_le_delta
  top_n(-1, wt = worst_le_delta) %>%
  arrange(worst_le_delta)
```

# Let's have a exercise

- Answer

```
## `summarise()` has grouped output by 'continent'. You can override using
## `.groups` argument.
```

```
## # A tibble: 5 x 3
## # Groups:   continent [5]
##   continent country     worst_le_delta
##   <fct>     <fct>                <dbl>
## 1 Africa    Rwanda               -20.4
## 2 Asia      Cambodia              -9.10
## 3 Americas  El Salvador           -1.51
## 4 Europe    Montenegro            -1.46
## 5 Oceania   Australia              0.170
```

Subsection 7

# Relational data

# Introduction

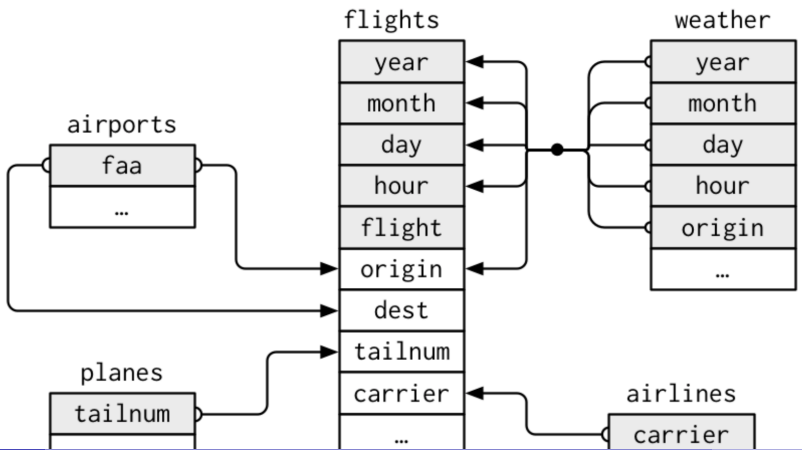
- Collectively, multiple tables of data are called relational data because it is the relations, not just the individual datasets, that are important.
- eg. a household survey often composites with several subsets: some are reported in household level: such as consumption,assets,etc. others are reported in individual level: like working information.

# Types of join

- **Mutation joins** first matches observations by their keys, then copies across variables from one table to the other.
- **Filtering joins** match observations in the same way as mutating joins, but affect the **observations**, not the variables.

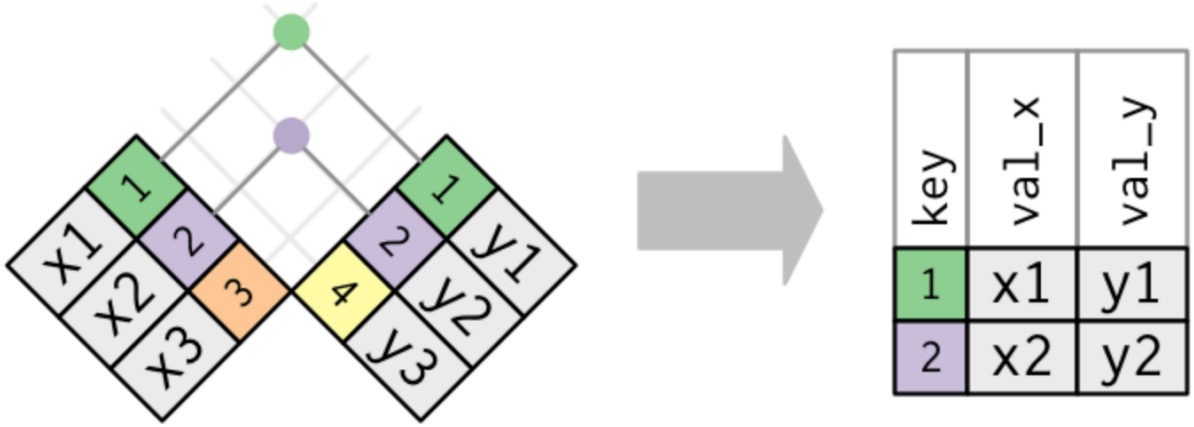
# Understanding joins

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



# Inner joins

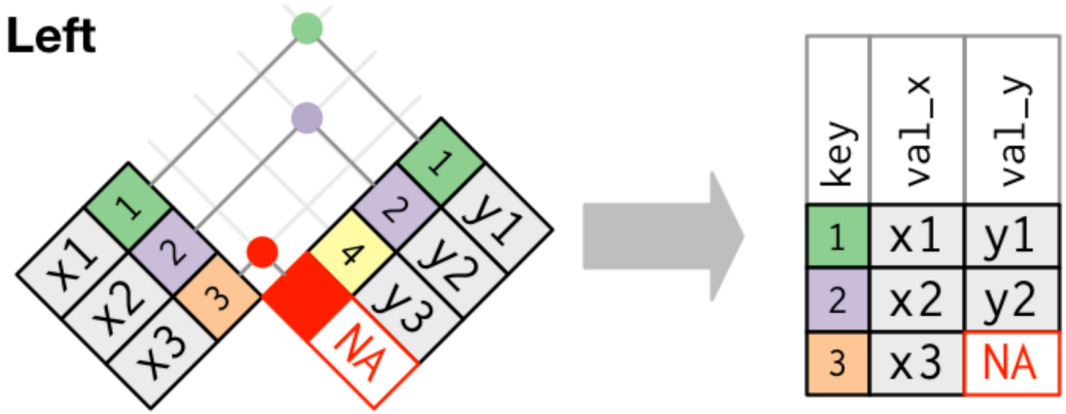
- **Inner join** matches pairs of observations whenever their keys are equal.




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

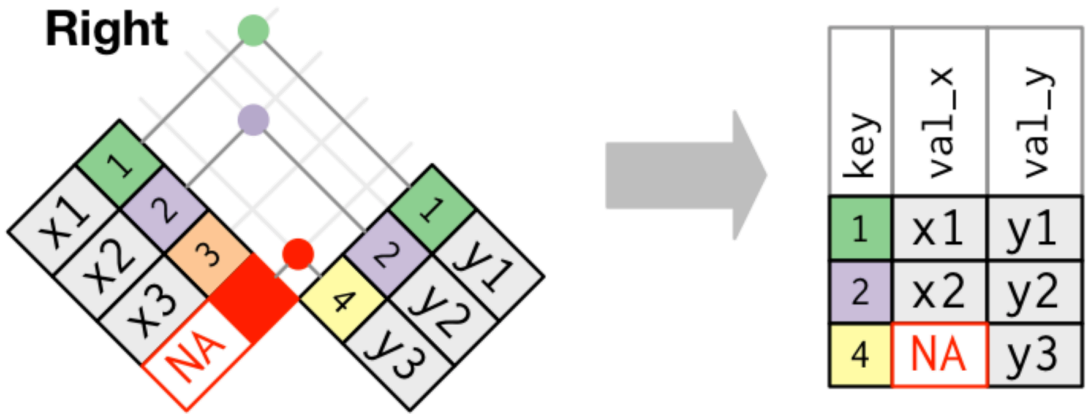
# Outer joins:left

- A **left join** keeps all observations in x.



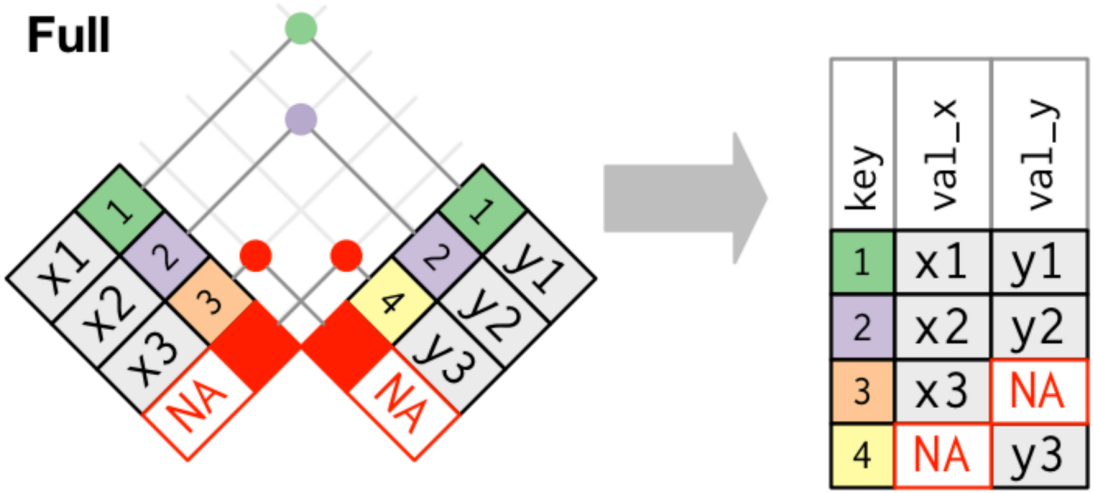
# Outer joins: right

- A **right join** keeps all observations in y.



# Outer joins: full

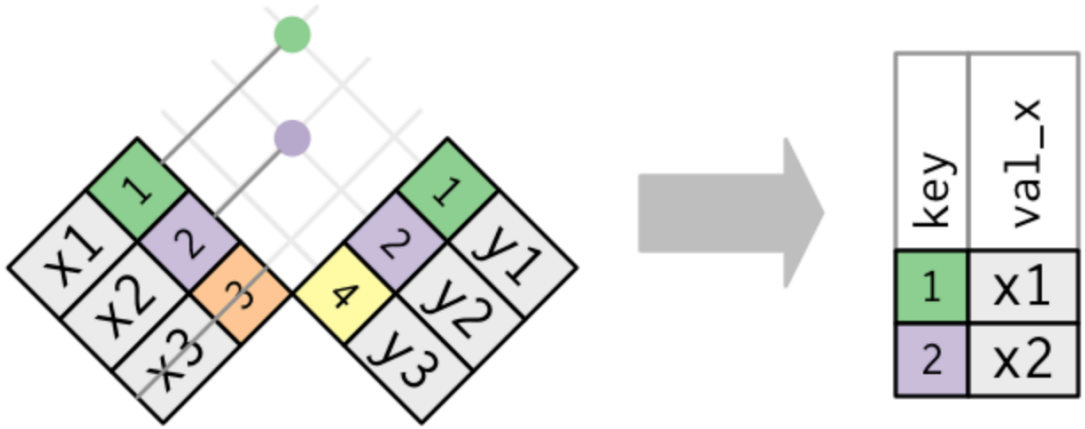
- A **full join** keeps all observations in x and y.




| key | val_x | val_y |
|---|---|---|
| 1 | x1 | y1 |
| 2 | x2 | y2 |
| 3 | x3 | NA |
| 4 | NA | y3 |

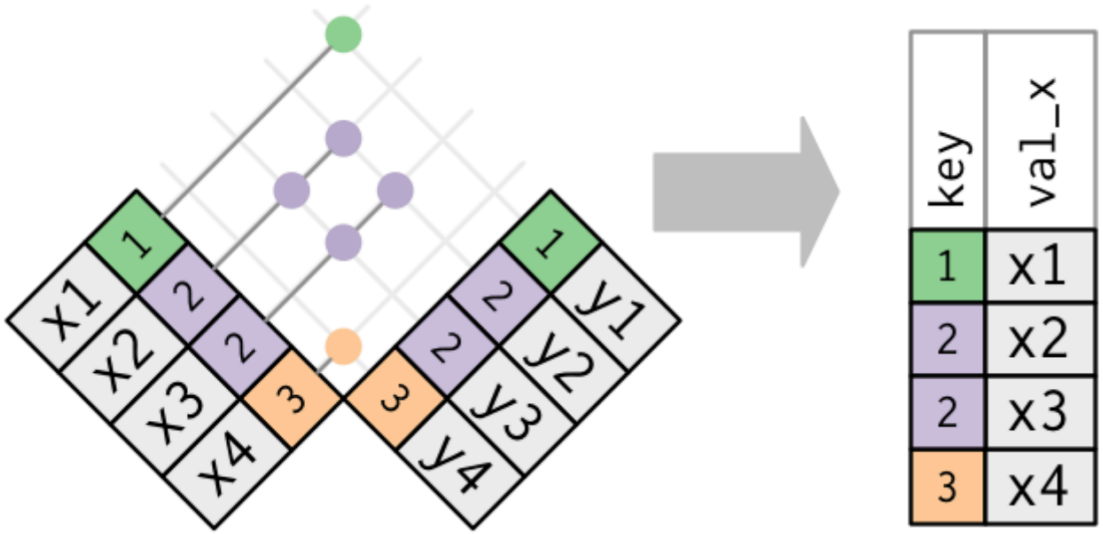
# Filtering joins

- `semi_join(x,y)` **keeps** all observations in x that have a match in y.



# Filtering joins

- Only the existence of a match is important it does not matter which observation is matched.
- This means that filtering joins never duplicate rows like mutating joins do:



# Filtering joins

- `anti_join(x,y)` **drops** all observations in x that have a match in y.

# nycflights13 data

- a dataset on flights departing New York City in 2013.

```
installed.packages(nycflights13)
```

```
library(nycflights13)
```

```
## Warning: package 'nycflights13' was built under R version 4
```

```
airlines
```

```
## # A tibble: 16 x 2
##    carrier name
##    <chr>   <chr>
##  1 9E      Endeavor Air Inc.
##  2 AA      American Airlines Inc.
##  3 AS      Alaska Airlines Inc.
##  4 B6      JetBlue Airways
##  5 DL      Delta Air Lines Inc.
```

# nycflights13:airports

```
airports
```

```
## # A tibble: 1,458 x 8
##    faa   name                              lat    lon   alt
##    <chr> <chr>                           <dbl>  <dbl> <dbl>
##  1 04G   Lansdowne Airport                41.1  -80.6  1044
##  2 06A   Moton Field Municipal Airport    32.5  -85.7   264
##  3 06C   Schaumburg Regional              42.0  -88.1   801
##  4 06N   Randall Airport                  41.4  -74.4   523
##  5 09J   Jekyll Island Airport            31.1  -81.4    11
##  6 0A9   Elizabethton Municipal Airport   36.4  -82.2  1593
##  7 0G6   Williams County Airport          41.5  -84.5   730
##  8 0G7   Finger Lakes Regional Airport    42.9  -76.8   492
##  9 0P2   Shoestring Aviation Airfield     39.8  -76.6  1000
## 10 0S9   Jefferson County Intl            48.1 -123.    108
## # i 1,448 more rows
```

# nycflights13:planes

```
planes
```

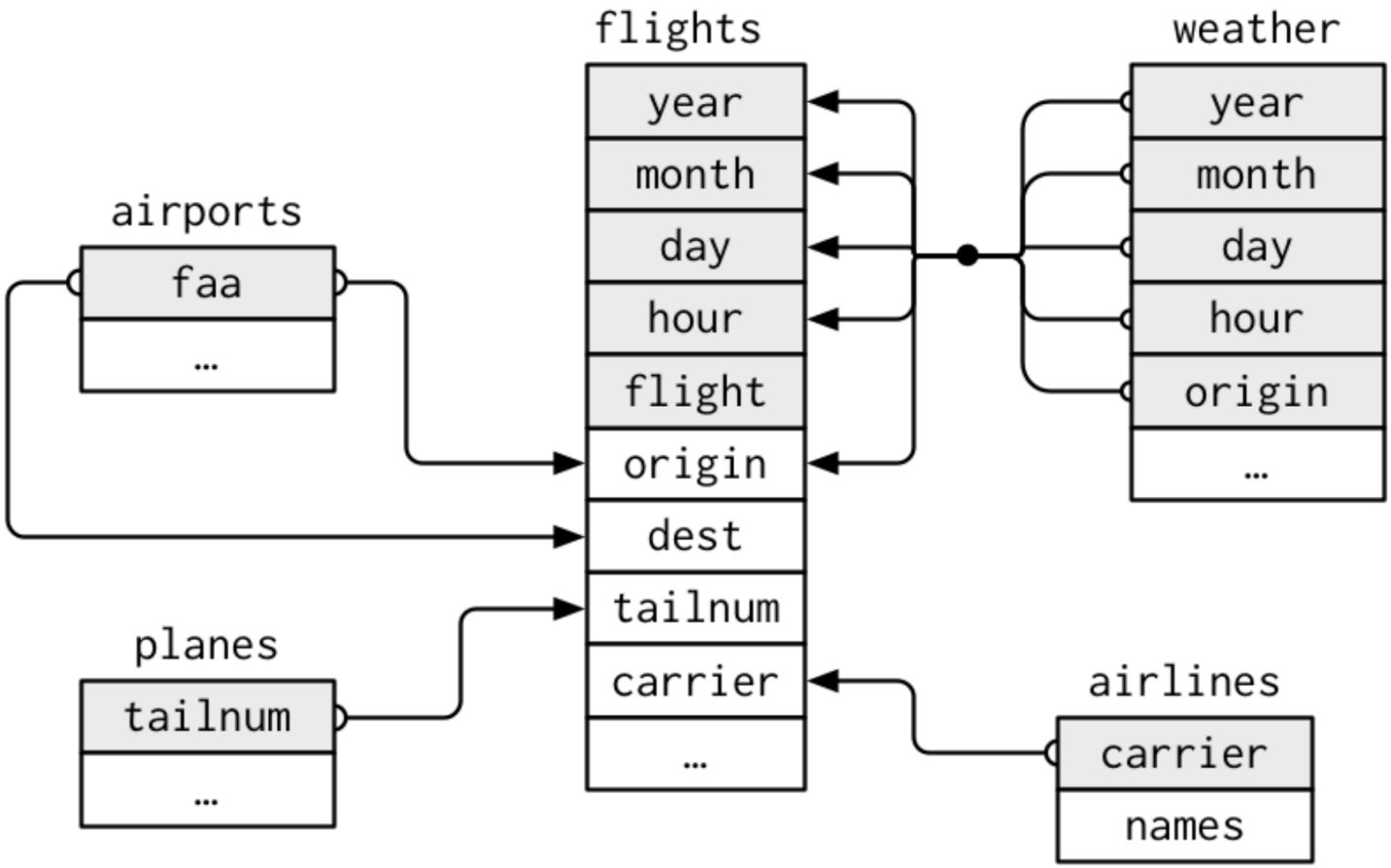
```
## # A tibble: 3,322 x 9
##    tailnum  year type                manufacturer model engi
##    <chr>   <int> <chr>               <chr>        <chr>   <i
##  1 N10156   2004 Fixed wing multi~ EMBRAER      EMB-~
##  2 N102UW   1998 Fixed wing multi~ AIRBUS INDU~ A320~
##  3 N103US   1999 Fixed wing multi~ AIRBUS INDU~ A320~
##  4 N104UW   1999 Fixed wing multi~ AIRBUS INDU~ A320~
##  5 N10575   2002 Fixed wing multi~ EMBRAER      EMB-~
##  6 N105UW   1999 Fixed wing multi~ AIRBUS INDU~ A320~
##  7 N107US   1999 Fixed wing multi~ AIRBUS INDU~ A320~
##  8 N108UW   1999 Fixed wing multi~ AIRBUS INDU~ A320~
##  9 N109UW   1999 Fixed wing multi~ AIRBUS INDU~ A320~
## 10 N110UW   1999 Fixed wing multi~ AIRBUS INDU~ A320~
## # i 3,312 more rows
```

# nycflights13:weather

```
weather
```

```
## # A tibble: 26,115 x 15
##    origin  year month   day  hour  temp  dewp humid wind_di
##    <chr>  <int> <int> <int> <int> <dbl> <dbl> <dbl>    <dbl
##  1 EWR     2013     1     1     1  39.0  26.1  59.4       27
##  2 EWR     2013     1     1     2  39.0  27.0  61.6       25
##  3 EWR     2013     1     1     3  39.0  28.0  64.4       24
##  4 EWR     2013     1     1     4  39.9  28.0  62.2       25
##  5 EWR     2013     1     1     5  39.0  28.0  64.4       26
##  6 EWR     2013     1     1     6  37.9  28.0  67.2       24
##  7 EWR     2013     1     1     7  39.0  28.0  64.4       24
##  8 EWR     2013     1     1     8  39.9  28.0  62.2       25
##  9 EWR     2013     1     1     9  39.9  28.0  62.2       26
## 10 EWR     2013     1     1    10  41    28.0  59.6       26
## # i 26,105 more rows
## # i 5 more variables: wind_gust <dbl>, precip <dbl>, pressu
```

# Relations in nycflights13 data



# Key variables

- The variables used to connect each pair of tables are called **keys**. A key is a variable (or set of variables) that uniquely identifies an observation.
  - **Primary** Key: uniquely identifies an observation in its **own** table
  - **Foreign** Key: uniquely identifies an observation in **another** table

```
head(planes$tailnum) # primary key in `planes` table
```

```
## [1] "N10156" "N102UW" "N103US" "N104UW" "N10575" "N105UW"
```

```
head(flights$tailnum) # foreign key in `flights` table
```

```
## [1] "N14228" "N24211" "N619AA" "N804JB" "N668DN" "N39463"
```

# Key variables

- Verify that key variables do indeed uniquely identify each observation.
  - count() the primary keys and look for entries where n is greater than one

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

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

# Key variables

- Sometimes you can

```
weather %>%
  count(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
```

# Join(merge) tables

- A primary key and the corresponding foreign key in another table form a **relation**.
  - 1 : M(any)
  - 1 : 1
  - M : M

# Join(merge) tables

- **mutating join**: It first matches observations by their keys, then copies across variables from one table to the other.
- Aim: add the full `airline` name to the `flight2`

```
flights2 <- flights %>%
  select(year:day,hour,tailnum,carrier,origin)
```

# Join(merge) tables

- Aim: add the full `airline` name to the `flight2`

```
flights2  %>%
  left_join(airlines, by = "carrier")
```

```
## # A tibble: 336,776 x 8
##     year month   day  hour tailnum carrier origin name
##    <int> <int> <int> <dbl> <chr>   <chr>   <chr>  <chr>
##  1  2013     1     1     5 N14228  UA      EWR    United Air Lines Inc.
##  2  2013     1     1     5 N24211  UA      LGA    United Air Lines Inc.
##  3  2013     1     1     5 N619AA  AA      JFK    American Airlines Inc.
##  4  2013     1     1     5 N804JB  B6      JFK    JetBlue Airways
##  5  2013     1     1     6 N668DN  DL      LGA    Delta Air Lines Inc.
##  6  2013     1     1     5 N39463  UA      EWR    United Air Lines Inc.
##  7  2013     1     1     6 N516JB  B6      EWR    JetBlue Airways
##  8  2013     1     1     6 N829AS  EV      LGA    ExpressJet Airlines In
##  9  2013     1     1     6 N593JB  B6      JFK    JetBlue Airways
## 10  2013     1     1     6 N3ALAA  AA      LGA    American Airlines Inc.
## # i 336,766 more rows
```

# Join(merge) tables

```
planes %>%
  select(tailnum,type) %>%
  right_join(flights,by = "tailnum")
```

```
## # A tibble: 336,776 x 20
##    tailnum type        year month   day dep_time sched_dep_ti
##    <chr>   <chr>      <int> <int> <int>    <int>          <in
##  1 N10156  Fixed w~  2013     1    10      626              6
##  2 N10156  Fixed w~  2013     1    10     1120             10
##  3 N10156  Fixed w~  2013     1    10     1619             15
##  4 N10156  Fixed w~  2013     1    11      632              6
##  5 N10156  Fixed w~  2013     1    11     1116             11
##  6 N10156  Fixed w~  2013     1    11     1845             18
##  7 N10156  Fixed w~  2013     1    12      830              8
##  8 N10156  Fixed w~  2013     1    12     1410             13
##  9 N10156  Fixed w~  2013     1    13     1551             15
```

# Top 10 most popular destinations

```
## # A tibble: 10 x 2
##    dest      n
##    <chr> <int>
##  1 ORD   17283
##  2 ATL   17215
##  3 LAX   16174
##  4 BOS   15508
##  5 MCO   14082
##  6 CLT   14064
##  7 SFO   13331
##  8 FLL   12055
##  9 MIA   11728
## 10 DCA    9705
```

# Top 10 most popular destinations

- find each flight that went to one of those destinations.

```
## # A tibble: 141,145 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_ar
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      542            540         2      923
##  2  2013     1     1      554            600        -6      812
##  3  2013     1     1      554            558        -4      740
##  4  2013     1     1      555            600        -5      913
##  5  2013     1     1      557            600        -3      838
##  6  2013     1     1      558            600        -2      753
##  7  2013     1     1      558            600        -2      924
##  8  2013     1     1      558            600        -2      923
##  9  2013     1     1      559            559         0      702
## 10  2013     1     1      600            600         0      851
## # i 141,135 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <d
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

# Top 10 most popular destinations

- difficult to extend that approach to multiple variables.

```
## Joining with `by = join_by(dest)`

## # A tibble: 141,145 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_ar
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      542            540         2      923
##  2  2013     1     1      554            600        -6      812
##  3  2013     1     1      554            558        -4      740
##  4  2013     1     1      555            600        -5      913
##  5  2013     1     1      557            600        -3      838
##  6  2013     1     1      558            600        -2      753
##  7  2013     1     1      558            600        -2      924
##  8  2013     1     1      558            600        -2      923
##  9  2013     1     1      559            559         0      702
## 10  2013     1     1      600            600         0      851
## # i 141,135 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <d
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

# anti_join

- Whether are there `flights` that do not have a match in `planes`

```
## # A tibble: 722 x 2
##    tailnum     n
##    <chr>   <int>
##  1 <NA>     2512
##  2 N725MQ    575
##  3 N722MQ    513
##  4 N723MQ    507
##  5 N713MQ    483
##  6 N735MQ    396
##  7 NOEGMQ    371
##  8 N534MQ    364
##  9 N542MQ    363
## 10 N531MQ    349
## # i 712 more rows
```

# Some Notes for join data

1. Start by identifying the variables that form the primary key in each table.
2. Check that none of the variables in the primary key are missing and whether is duplicated(for inner join).
3. Check that your foreign keys match primary keys in another table.

# Topics in the Next Lecture

- Missing Values and Duplicates
- ggplot2
- Rmarkdown

# Reference

1. Grolemund & Wickham(2017), R for Data Science
2. DataCamp(2018), Introduction to Tidyverse