Data transformation

Harlley Lima & Manel Slokom

December 5, 2018

Overview

1. Pipes
2. Data transformation
   - Data
   - Primary functions
3. R project

Prerequisites

Pipes requires:

```
install.packages("magrittr")
library(magrittr)
```

- Packages in the tidyverse load Pipes for you automatically
- You don't usually load magrittr explicitly, just load `library(tidyverse)`

Pipes

- Pipes are a powerful tool for clearly expressing a sequence of multiple operations
- Pipe helps to write code in a way that is easier to read and understand
- Pipe operator %>%: provides the ability to string multiple functions together

Basic piping

- x %>% f is equivalent to $f(x)$
  - pi %>% sin
- x %>% f(y) is equivalent to $f(x, y)$
  - "Hello" %>% cat(", world!")
- x %>% f %>% g %>% h is equivalent to $h(g(f(x)))$
  - `pi %>% sin %>% cos`

Pipe - Example

Using the diamonds dataset, calculate the average price for each cut of "I" colored diamonds:

1. Filter diamonds to only keep observations where the color is rated as "I"
2. Group the filtered diamonds data frame by cut
3. Summarize the grouped and filtered diamonds by calculating the average prices

Intermediate steps

```
diamonds_1 <- filter(diamonds, color == "I")
diamonds_2 <- group_by(diamonds_1, cut)
diamonds_3 <- summarize(diamonds_2, price = mean(price))
```

Using pipes

```
diamonds %>%
    filter(color == "I") %>%
    group_by(cut) %>%
    summarize(price = mean(price))
```

When not to use the pipe

You should reach for another tool when

- Your pipes are longer than (say) ten steps
- If there are two or more variables being combined together, don't use the pipe

Data transformation

Data transformation

How to transform the data?

- To get the data in the right form
- Create new variables or summaries
- To rename the variables or reorder the observations

```
library(nycflights13)
library(tidyverse)
library(dplyr) # a package that transforms data
```

nycflights13

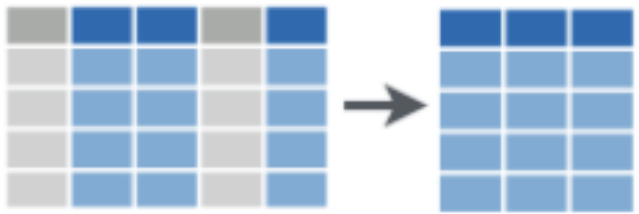
- nycflights13::flights

```
> flights
# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# ... with 336,766 more rows, and 11 more variables: 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>
```

Primary functions

1. **select()**: to select variables of concern
2. **filter()**: to filter values based on conditions
3. **group_by()**: to group data by categorical levels
4. **summarise()**: to change unit of analysis
5. **arrange()**: to reorder the rows
6. **mutate()**: to create new variables
7. **join()**: to combine separate data sets

select( ) function



select( ) function

Objective: to reduce dataframe size to desired variables for current task.
Description: When working with a sizable dataframe, often we desire to only assess specific variables.
The `select()` function allows to select specific variables.

- Function: `select(data, ...)`
- same as:
```
data %>%
    select(...)
```

select( ) function

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

```
> 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
```

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

Special functions within select( )

- `starts_with("abc")`: matches names that begin with "abc".
- `ends_with("xyz")`: matches names that end with "xyz".
- `contains("ijk")`: matches names that contain "ijk".
- `num_range("x", 1:3)`: matches x1, x2 and x3.

Special functions within select( )

```
# select all variables that start with "d":
flights %>%
   select(starts_with("d"))
```

```
> flights %>%
+   select(starts_with("d"))
# A tibble: 336,776 x 5
     day dep_time dep_delay dest  distance
   <int>    <int>     <dbl> <chr>    <dbl>
 1     1      517         2 IAH       1400
 2     1      533         4 IAH       1416
 3     1      542         2 MIA       1089
 4     1      544        -1 BQN       1576
 5     1      554        -6 ATL        762
 6     1      554        -4 ORD        719
 7     1      555        -5 FLL       1065
 8     1      557        -3 IAD        229
 9     1      557        -3 MCO        944
10     1      558        -2 ORD        733
# ... with 336,766 more rows
```

Special functions within select( )

```
# select all variables that contain "dep":
flights %>%
   select(contains("dep"))
```

```
> flights %>%
+   select(contains("dep"))
# A tibble: 336,776 x 3
   dep_time sched_dep_time dep_delay
      <int>          <int>     <dbl>
 1      517            515         2
 2      533            529         4
 3      542            540         2
 4      544            545        -1
 5      554            600        -6
 6      554            558        -4
 7      555            600        -5
 8      557            600        -3
 9      557            600        -3
10      558            600        -2
# ... with 336,766 more rows
```

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

filter( ) function

filter( ) function

Objective: Reduce rows/observations with matching conditions.
Description: Filtering data is a common task to identify/select observations in which a particular variable matches a specific value/condition.

- Function: `filter(data, ...)`
- Same as: `data %>% filter(...)`

- The first argument is the name of the data frame.
- The second and subsequent arguments are the expressions that filter the data frame.

Filter rows

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

```
> filter(flights, month == 1, day == 1)
# A tibble: 842 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# ... with 832 more rows, and 11 more variables: 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>
```

```
df <- filter(flights, month == 1, day == 1)
# Similarly
df <- flights %>%
  filter(month == 1, day == 1)
```

Logic rules in filter()

We can apply multiple logic rules in the `filter()` function such as:

- < less than, > greater than
- == equal to, != not equal to
- <= less than or equal to, >= greater or equal to
- `is.na` is NA
- `!is.na` is not NA

Logic rules in filter()

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

```
> filter(flights, month == 11 | month == 12)
# A tibble: 55,403 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
 1  2013    11     1        5           2359         6      352            345         7
 2  2013    11     1       35           2250       105      123           2356        87
 3  2013    11     1      455            500        -5      641            651       -10
 4  2013    11     1      539            545        -6      856            827        29
 5  2013    11     1      542            545        -3      831            855       -24
 6  2013    11     1      549            600       -11      912            923       -11
 7  2013    11     1      550            600       -10      705            659         6
 8  2013    11     1      554            600        -6      659            701        -2
 9  2013    11     1      554            600        -6      826            827        -1
10  2013    11     1      554            600        -6      749            751        -2
# ... with 55,393 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>
```

Logic rules in filter( )

```
# remove duplicate rows
flights %>%
  distinct()
```

```
> flights %>%
+   distinct()
# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
 1  2013     1     1      517            515         2      830            819        11
 2  2013     1     1      533            529         4      850            830        20
 3  2013     1     1      542            540         2      923            850        33
 4  2013     1     1      544            545        -1     1004           1022       -18
 5  2013     1     1      554            600        -6      812            837       -25
 6  2013     1     1      554            558        -4      740            728        12
 7  2013     1     1      555            600        -5      913            854        19
 8  2013     1     1      557            600        -3      709            723       -14
 9  2013     1     1      557            600        -3      838            846        -8
10  2013     1     1      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>
```

Logic rules in filter( )

```
# select rows by position: row =3, 4 and 5
flights %>%
  slice(3:5)
```

```
> flights %>%
+  slice(3:5)
# A tibble: 3 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
  <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
1  2013     1     1      542            540         2      923            850        33
2  2013     1     1      544            545        -1     1004           1022       -18
3  2013     1     1      554            600        -6      812            837       -25
# ... with 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>
```

group_by() function

group_by( ) function

Objective: Group data by categorical variables
Description: Often, observations are nested within groups or categories. The `group_by()` function allows to create these categorical groupings.

- Function: `group_by(data, ...)`
- Same as:
```
data %>%
    group_by(...)
```

- data: data frame
- ...: variables to group_by

group_by( ) function

```
flights %>%
    group_by(year, month, day)
```

```
> flights %>%
+  group_by(year, month, day)
# A tibble: 336,776 x 19
# Groups:   year, month, day [365]
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
 1  2013     1     1      517            515         2      830            819        11
 2  2013     1     1      533            529         4      850            830        20
 3  2013     1     1      542            540         2      923            850        33
 4  2013     1     1      544            545        -1     1004           1022       -18
 5  2013     1     1      554            600        -6      812            837       -25
 6  2013     1     1      554            558        -4      740            728        12
 7  2013     1     1      555            600        -5      913            854        19
 8  2013     1     1      557            600        -3      709            723       -14
 9  2013     1     1      557            600        -3      838            846        -8
10  2013     1     1      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>
```

Use `ungroup(x)` to remove groups.

summarise() function

summarise() function

Objective: Perform summary statistics on variables
Description: The `summarise()` function allows us to perform the majority of the initial summary statistics when performing exploratory data analysis.

- Function: `summarise(data, ...)`
- Same as:
```
data %>%
    summarise(...)
```

- data: data frame
- ... : name-value pairs of summary functions like min(), mean(), max() etc.

summarise() function

```
flights %>% summarise(Min = min(distance),
          Max = max(distance),
          Mean = mean(distance),
          SD = sd(distance),
          N = n()
)
```

```
# A tibble: 1 x 5
    Min   Max  Mean    SD      N
  <dbl> <dbl> <dbl> <dbl>  <int>
1    17  4983 1040.  733. 336776
```

summarise() function

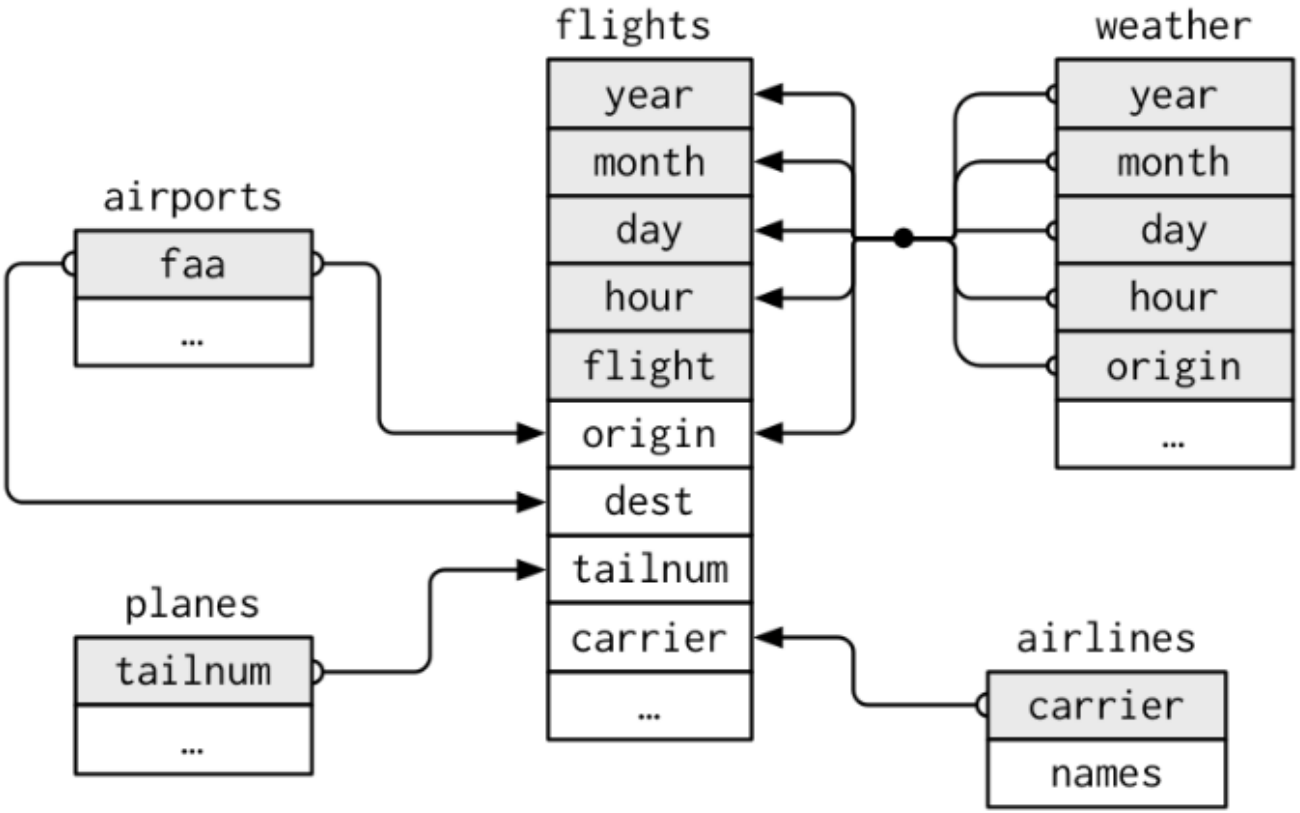
Summary statistics at multiple levels is when you really start to gather some insights:

```
flights %>% group_by(carrier) %>%
            summarise(Min = min(distance),
                Max = max(distance),
                Mean = mean(distance),
                SD = sd(distance),
                N = n()
)
```

```
# A tibble: 16 x 6
   carrier   Min   Max  Mean    SD     N
   <chr>   <dbl> <dbl> <dbl> <dbl> <int>
 1 9E         94  1587  530. 322.  18460
 2 AA        187  2586 1340. 638.  32729
 3 AS       2402  2402 2402    0     714
 4 B6        173  2586 1069. 704.  54635
 5 DL         94  2586 1237. 660.  48110
 6 EV         80  1389  563. 287.  54173
```

nycflights13

How can I get the full name of each airlines?



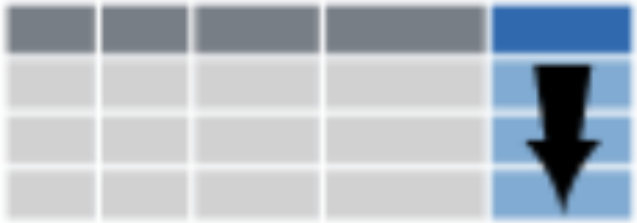
`flights` connects to `airlines` through the `carrier` variable

You can also use `join`:

```
flights %>% group_by(carrier) %>%,
            summarise(Min = min(distance),
                Max = max(distance),
                Mean = mean(distance),
                SD = sd(distance),
                N = n())
            inner_join(airlines, by = "carrier")
```

```
# A tibble: 16 x 7
   carrier   Min   Max  Mean    SD     N name
   <chr>   <dbl> <dbl> <dbl> <dbl> <int> <chr>
 1 9E         94  1587  530. 322.  18460 Endeavor Air Inc.
 2 AA        187  2586 1340. 638.  32729 American Airlines Inc.
 3 AS       2402  2402 2402    0     714 Alaska Airlines Inc.
 4 B6        173  2586 1069. 704.  54635 JetBlue Airways
 5 DL         94  2586 1237. 660.  48110 Delta Air Lines Inc.
 6 EV         80  1389  563. 287.  54173 ExpressJet Airlines Inc.
```

arrange() function



arrange( ) function

Objective: Order variable values
Description: to view observations in a rank order for a particular variable(s).
The `arrange()` function allows us to order data by variables in ascending or descending order.

- `Function: arrange(data, ...)` ‘
- Same as:
```
data %>%
    arrange(...)
```

- data: data frame
- ...: Variable(s) to order
- use `desc(x)` to sort variable in descending order

arrange( ) function

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

```
>  arrange(flights, year, month, day)
# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
 1  2013     1     1      517            515         2      830            819        11
 2  2013     1     1      533            529         4      850            830        20
 3  2013     1     1      542            540         2      923            850        33
 4  2013     1     1      544            545        -1     1004           1022       -18
 5  2013     1     1      554            600        -6      812            837       -25
 6  2013     1     1      554            558        -4      740            728        12
 7  2013     1     1      555            600        -5      913            854        19
 8  2013     1     1      557            600        -3      709            723       -14
 9  2013     1     1      557            600        -3      838            846        -8
10  2013     1     1      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>
```

arrange( ) function

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

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

```
>  arrange(flights, desc(dep_delay))
# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
 1  2013     1     9      641            900      1301     1242           1530      1272
 2  2013     6    15     1432           1935      1137     1607           2120      1127
 3  2013     1    10     1121           1635      1126     1239           1810      1109
 4  2013     9    20     1139           1845      1014     1457           2210      1007
 5  2013     7    22      845           1600      1005     1044           1815       989
 6  2013     4    10     1100           1900       960     1342           2211       931
 7  2013     3    17     2321            810       911      135           1020       915
 8  2013     6    27      959           1900       899     1236           2226       850
 9  2013     7    22     2257            759       898      121           1026       895
10  2013    12     5      756           1700       896     1058           2020       878
# ... 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>
```

mutate() function

Add new variables with mutate( )

Objective: Creates new variables at the end of the data frame
Description: to create a new variable that is a function of the current variables in the dataframe or even just add a new variable.
The `mutate()` function allows us to add new variables while preserving the existing variables.

- Function: `mutate(data, ...)`
- Same as:
```
data %>%
    mutate(...)
```

Add new variables with mutate( )

```
# create a dataset
> flights_sml <- select(flights,
    year:day,
    ends_with("delay"),
    distance,
    air_time
    )
```

```
> flights_sml <- select(flights,
+   year:day,
+   ends_with("delay"),
+   distance,
+   air_time
+ )
>  flights_sml
# A tibble: 336,776 x 7
    year month   day dep_delay arr_delay distance air_time
   <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl>
 1  2013     1     1         2        11     1400      227
 2  2013     1     1         4        20     1416      227
 3  2013     1     1         2        33     1089      160
 4  2013     1     1        -1       -18     1576      183
 5  2013     1     1        -6       -25      762      116
 6  2013     1     1        -4        12      719      150
 7  2013     1     1        -5        19     1065      158
 8  2013     1     1        -3       -14      229       53
 9  2013     1     1        -3        -8      944      140
10  2013     1     1        -2         8      733      138
#   ... with 336,766 more rows
```

Add new variables with mutate( )

```
# mutate
> mutate(flights_sml,
    gain = dep_delay - arr_delay,
    speed = distance / air_time * 60
    )
```

```
> 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.
 9  2013     1     1        -3        -8      944      140     5  405.
10  2013     1     1        -2         8      733      138   -10  319.
# ... with 336,766 more rows
```

Add new variables with 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 gain_per_hour
   <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl> <dbl>          <dbl>
 1  2013     1     1         2        11     1400      227    -9 3.78           -2.38
 2  2013     1     1         4        20     1416      227   -16 3.78           -4.23
 3  2013     1     1         2        33     1089      160   -31 2.67          -11.6
 4  2013     1     1        -1       -18     1576      183    17 3.05            5.57
 5  2013     1     1        -6       -25      762      116    19 1.93            9.83
 6  2013     1     1        -4        12      719      150   -16 2.5            -6.4
 7  2013     1     1        -5        19     1065      158   -24 2.63           -9.11
 8  2013     1     1        -3       -14      229       53    11 0.883          12.5
 9  2013     1     1        -3        -8      944      140     5 2.33            2.14
10  2013     1     1        -2         8      733      138   -10 2.3            -4.35
# ... with 336,766 more rows
```

transmute( )

To only keep the new variables, use `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
```

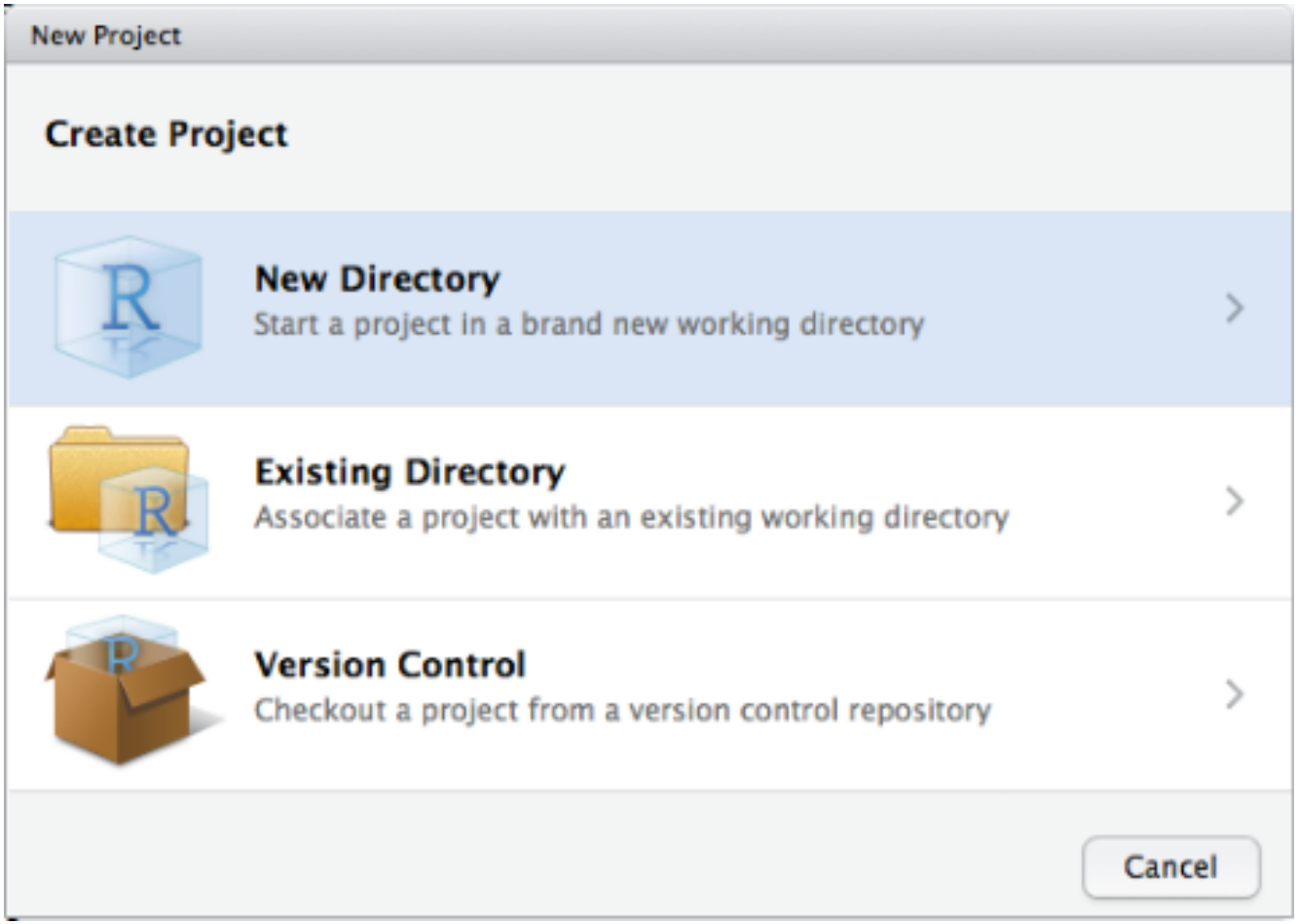
R project

Create an R project

- A folder in R-Studio for all your work on one project
  - R experts keep all the files associated with a project together – input data, R scripts, analytical results, figures
  - R will load from and save to here

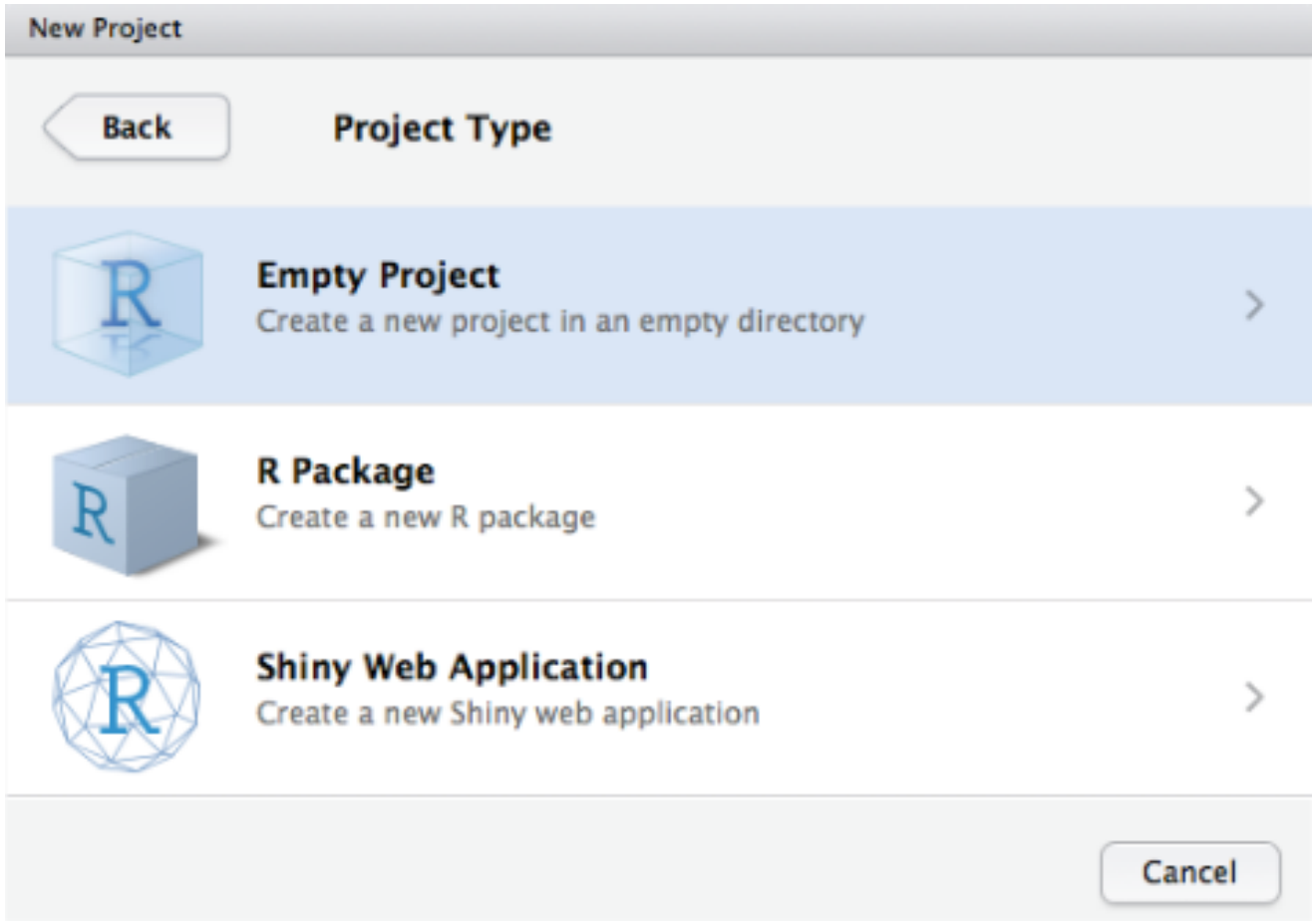
RStudio projects

Click `File > New Project`



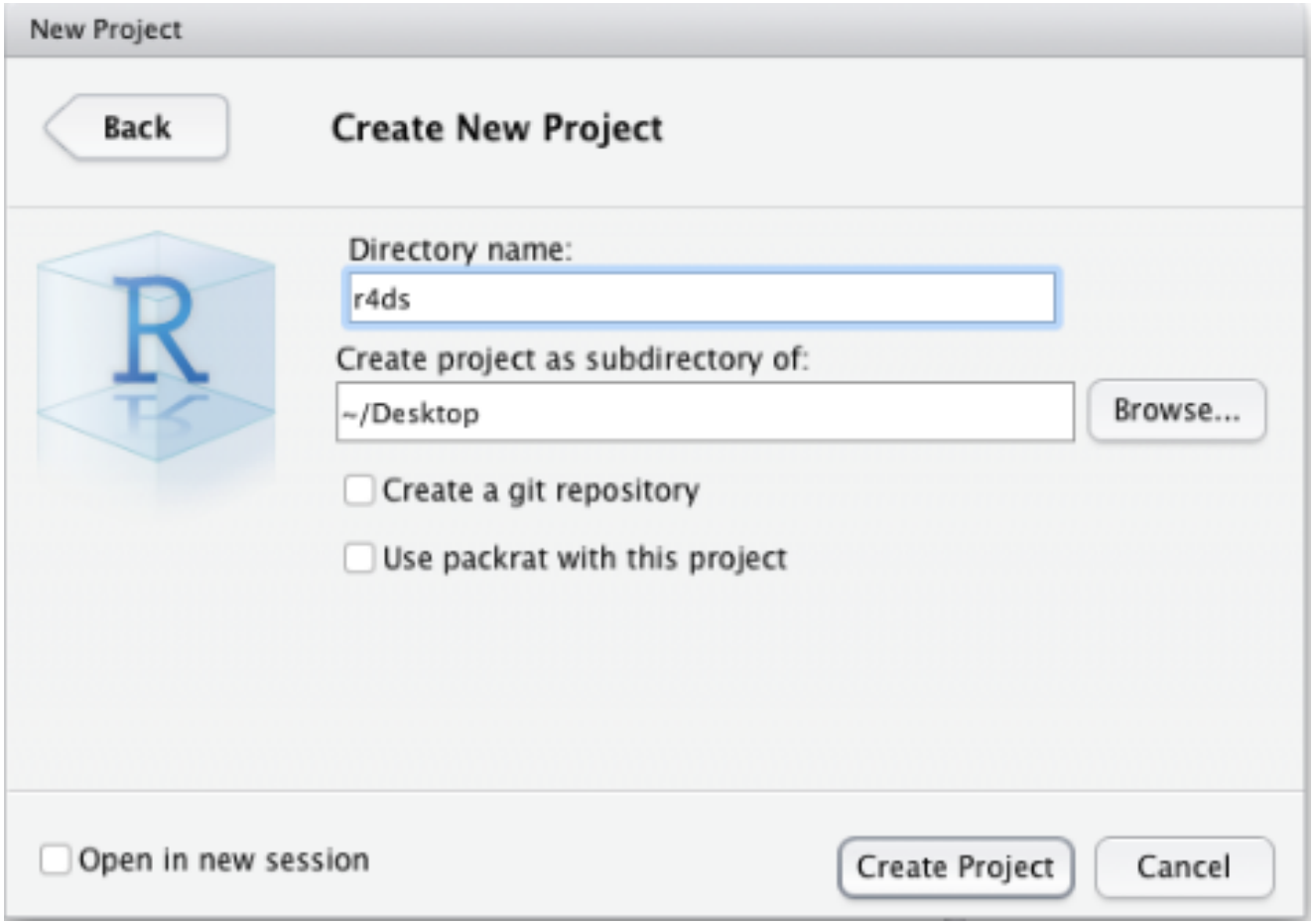
RStudio projects

Click `File > New Project`



RStudio projects

Click `File > New Project`



Working Directory

- type `getwd()` in the console to see your working directory
  - `[1] "/home/harlley/Projects/FinalProject"`
- RStudio automatically sets the directory to the folder containing your R project
- Whenever you refer to a file with a **relative path** it will look for it in the working directory
  - **Relative paths** you take as reference the current working directory
  - While an **absolute path** specifies the location of a file or directory from the root directory
- You should **never** use absolute paths in your scripts, because they hinder sharing: no one else will have exactly the same directory configuration as you

References

Hadley Wickham & Garrett Grolemund (2017)

R for data science: import, tidy, transform, visualize, and model data

O'Reilly.

Transforming Your Data with dplyr

AFIT Data Science Lab R Programming Guide

https://afit-r.github.io/dplyr

Brad Boehmke (2015)

Data wrangling in R

http://rpubs.com/bradleyboehmke/data_processing

garrettgman (2018)

RStudio Cheat Sheets

https://github.com/rstudio/cheatsheets

The End