ISOM 3390: Business Programming in R

Instructor: Bingjie Qian (PhD candidate)

Summer 2020, HKUST

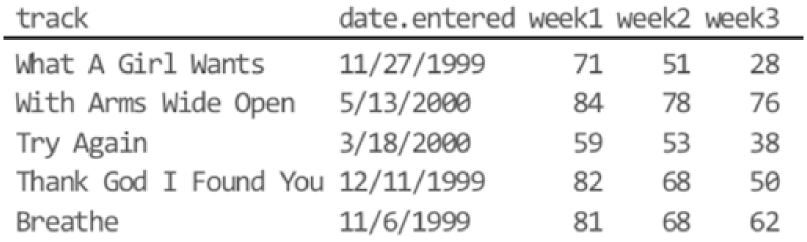
Topic 6: Data Wrangling: A Tidy Data Approach

In the first 5 topics, we have learned the basic skills of R programming. Now we are ready to use R to tackle data science and business analytics problems.

6.1 An Overview of Tidy Data

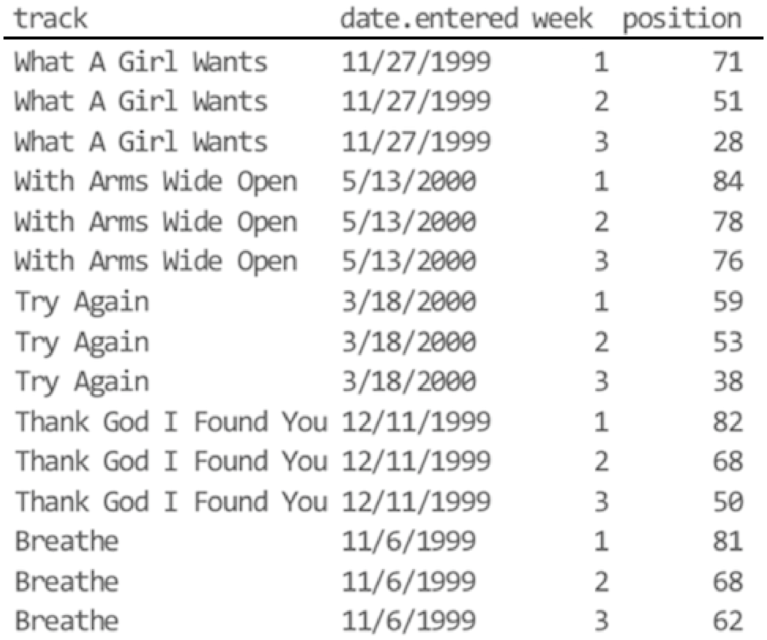
In practice, there are multiple ways to store and present a data set, but they are not equally useful.

Suppose we the data on 5 songs and their weekly ranks in the "Billboard Top 100" over 3 weeks. We can store the data in the *wide format* or in the *long format* as follows.



track	date.entered	week1	week2	week3
What A Girl Wants	11/27/1999	71	51	28
With Arms Wide Open	5/13/2000	84	78	76
Try Again	3/18/2000	59	53	38
Thank God I Found You	12/11/1999	82	68	50
Breathe	11/6/1999	81	68	62

Table 1. Wide Format



track	date.entered	week	position
What A Girl Wants	11/27/1999	1	71
What A Girl Wants	11/27/1999	2	51
What A Girl Wants	11/27/1999	3	28
With Arms Wide Open	5/13/2000	1	84
With Arms Wide Open	5/13/2000	2	78
With Arms Wide Open	5/13/2000	3	76
Try Again	3/18/2000	1	59
Try Again	3/18/2000	2	53
Try Again	3/18/2000	3	38
Thank God I Found You	12/11/1999	1	82
Thank God I Found You	12/11/1999	2	68
Thank God I Found You	12/11/1999	3	50
Breathe	11/6/1999	1	81
Breathe	11/6/1999	2	68
Breathe	11/6/1999	3	62

Table 2. Long Format

- In the wide format, each row represents *a specific song*.
- In the long format, each row represents *a specific song in a specific week*.

If we want to generate a plot that shows each song's ranking variation from the first to the last week, we need to use week number as the variable for x-axis. It is easy to do so with the long format, but with the wide format, week number is not a variable.

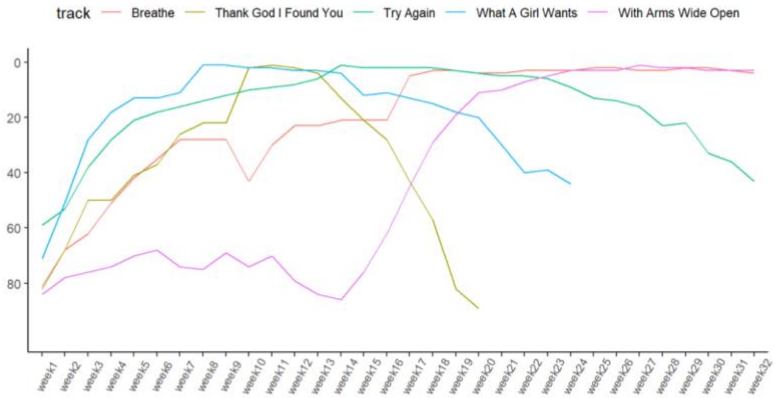


Fig 1. Ranking Plot

Besides, if some songs did not stay in the Billboard for all the weeks, you may find a lot of missing values (`NA`) with the wide format.

The dataset `billboard.csv` (download from Canvas) records weekly ranks of 317 songs appearing in the Billboard Top 100 over a certain time period.

```
billboard <- read.csv("billboard.csv")
head(billboard)   # wide format
```

```
##                 artist                                  track time genre
## 1     Destiny's Child              Independent Women Part I 3:38  Rock
## 2             Santana                          Maria, Maria 4:18  Rock
## 3       Savage Garden                    I Knew I Loved You 4:07  Rock
## 4             Madonna                                 Music 3:45  Rock
## 5 Aguilera, Christina Come On Over Baby (All I Want Is You) 3:38  Rock
## 6               Janet                 Doesn't Really Matter 4:17  Rock
##   date.entered date.peaked week1 week2 week3 week4 week5 week6 week7 week8
## 1    9/23/2000  11/18/2000    78    63    49    33    23    15     7     5
## 2    2/12/2000    4/8/2000    15     8     6     5     2     3     2     2
## 3   10/23/1999   1/29/2000    71    48    43    31    20    13     7     6
## 4    8/12/2000   9/16/2000    41    23    18    14     2     1     1     1
## 5     8/5/2000  10/14/2000    57    47    45    29    23    18    11     9
## 6    6/17/2000   8/26/2000    59    52    43    30    29    22    15    10
##   week9 week10 week11 week12 week13 week14 week15 week16 week17 week18
week19
## 1     1      1      1      1      1      1      1      1      1      1
1
## 2     1      1      1      1      1      1      1      1      1      1
8
## 3     4      4      4      6      4      2      1      1      1      2
1
## 4     1      2      2      2      2      2      4      8     11     16
20
## 5     9     11      1      1      1      1      4      8     12     22
23
## 6    10      5      1      1      1      2      2      3      3      7
8
##   week20 week21 week22 week23 week24 week25 week26 week27 week28 week29
week30
## 1      2      3      7     10     12     15     22     29     31     NA
NA
## 2     15     19     21     26     36     48     47     NA     NA     NA
NA
## 3      2      4      8      8     12     14     17     21     24     30
34
## 4     25     27     27     29     44     NA     NA     NA     NA     NA
NA
## 5     43     44     NA     NA     NA     NA     NA     NA     NA     NA
NA
## 6     20     25     37     40     41     NA     NA     NA     NA     NA
NA
##   week31 week32 week33 week34 week35 week36 week37 week38 week39 week40
week41
## 1     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
NA
## 2     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
```

```
NA
## 3     37     46     47     NA     NA     NA     NA     NA     NA     NA
NA
## 4     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
NA
## 5     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
NA
## 6     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
NA
##   week42 week43 week44 week45 week46 week47 week48 week49 week50 week51
week52
## 1     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
NA
## 2     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
NA
## 3     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
NA
## 4     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
NA
## 5     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
NA
## 6     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
NA
##   week53 week54 week55 week56 week57 week58 week59 week60 week61 week62
week63
## 1     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
NA
## 2     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
NA
## 3     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
NA
## 4     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
NA
## 5     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
NA
## 6     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
NA
##   week64
## 1     NA
## 2     NA
## 3     NA
## 4     NA
## 5     NA
## 6     NA
```

Common Causes of Data Messiness

Some common causes of messiness in data:

- Values are stored in column headers. E.g., in `billboard`, column headers are `week1`, `week2`, ..., `week64`, which stores the values of week number (1 to 64).

- Multiple types of observations in one table. E.g., the `billboard` dataset contains two types of observations: the song's meta information (`artist`, `time`, `genre`, `date.entered`, `date.peaked`) and its rank in each week.

- Multiple variables are stored in one column.

- One type in multiple tables.

"Tidy datasets are all alike, but every messy dataset is messy in its own way." - Hadley Wickham

Tidy data

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

- The following three interrelated rules make a dataset tidy:
  - Each variable forms a column.
  - Each observation forms a row.
  - Each type of observation forms a table.

- Two main advantages of tidy data:
  - Enforcing a consistent data structure makes it easier to learn the tools that work with it because of the **underlying uniformity**.
  - Placing variables in columns allows R's **vectorized** nature to shine.

- A popular set of "tidy" tools known as the **`tidyverse`** allows us to transition smoothly from different stages of data analysis.

`tidyverse`: R Packages for Data Science

`tidyverse` is a collection of packages that share an underlying design philosophy, grammar and data structures, and are designed to work together naturally.

Install the complete `tidyverse` packages with:

```
# install.packages("tidyverse")
library(tidyverse)

## ── Attaching packages ─────────────────────────────────────── tidyverse
1.3.0 ──

## ✓ ggplot2 3.3.2     ✓ purrr   0.3.4
## ✓ tibble  3.0.1     ✓ dplyr   1.0.0
## ✓ tidyr   1.1.0     ✓ stringr 1.4.0
## ✓ readr   1.3.1     ✓ forcats 0.5.0

## ── Conflicts ──────────────────────────────────────────
tidyverse_conflicts() ──
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

`install.packages("tidyverse")` installs the complete collection of packages in `tidyverse`.

`library(tidyverse)` loads the core `tidyverse` packages: `ggplot2`, `dplyr`, `tidyr`, `readr`, `purrr`, `tibble`, `stringr` and `forcats`.

Other packages with more specialized usage in `tidyverse` (e.g., `rvest`, `lubridate`, etc.) need to be loaded with their own `library()` calls.

6.2 Tibbles

One of the unifying features of `tidyverse` is a data structure called **tibble**.

Tibbles are a modern remake of the `data.frame` and encapsulate best practices for data frames.

Use `tibble()` to create a new tibble from column vectors:

```
tb <- tibble(x = 1:5, y = 1, z = x^2 + y)    # 1 is recycled
tb

## # A tibble: 5 x 3
##       x     y     z
##   <int> <dbl> <dbl>
## 1     1     1     2
## 2     2     1     5
## 3     3     1    10
## 4     4     1    17
## 5     5     1    26

str(tb)

## tibble [5 × 3] (S3: tbl_df/tbl/data.frame)
##  $ x: int [1:5] 1 2 3 4 5
##  $ y: num [1:5] 1 1 1 1 1
##  $ z: num [1:5] 2 5 10 17 26
```

It evaluates the arguments *sequentially* so that we can refer to variables just created, i.e., refer to x and y when creating z.

Differences between Tibbles and Data Frames

- It has a refined print method. It shows only the first 10 rows, fits all the columns on screen, and reports the type of each column along the name.

```
billboard_tbl <- as_tibble(billboard)
billboard_tbl

## # A tibble: 317 x 70
##    artist track time  genre date.entered date.peaked week1 week2 week3
week4
##    <chr>  <chr> <chr> <chr> <chr>        <chr>       <int> <int> <int>
<int>
##  1 Desti… Inde… 3:38  Rock  9/23/2000    11/18/2000     78    63    49
33
##  2 Santa… Mari… 4:18  Rock  2/12/2000    4/8/2000       15     8     6
5
##  3 Savag… I Kn… 4:07  Rock  10/23/1999   1/29/2000      71    48    43
31
##  4 Madon… Music 3:45  Rock  8/12/2000    9/16/2000      41    23    18
14
##  5 Aguil… Come… 3:38  Rock  8/5/2000     10/14/2000     57    47    45
29
##  6 Janet  Does… 4:17  Rock  6/17/2000    8/26/2000      59    52    43
30
##  7 Desti… Say … 4:31  Rock  12/25/1999   3/18/2000      83    83    44
38
##  8 Igles… Be W… 3:36  Latin 4/1/2000     6/24/2000      63    45    34
23
##  9 Sisqo  Inco… 3:52  Rock  6/24/2000    8/12/2000      77    66    61
61
## 10 Lones… Amaz… 4:25  Coun… 6/5/1999     3/4/2000       81    54    44
39
## # … with 307 more rows, and 60 more variables: week5 <int>, week6 <int>,
## #   week7 <int>, week8 <int>, week9 <int>, week10 <int>, week11 <int>,
## #   week12 <int>, week13 <int>, week14 <int>, week15 <int>, week16 <int>,
## #   week17 <int>, week18 <int>, week19 <int>, week20 <int>, week21 <int>,
## #   week22 <int>, week23 <int>, week24 <int>, week25 <int>, week26 <int>,
## #   week27 <int>, week28 <int>, week29 <int>, week30 <int>, week31 <int>,
## #   week32 <int>, week33 <int>, week34 <int>, week35 <int>, week36 <int>,
## #   week37 <int>, week38 <int>, week39 <int>, week40 <int>, week41 <int>,
## #   week42 <int>, week43 <int>, week44 <int>, week45 <int>, week46 <int>,
## #   week47 <int>, week48 <int>, week49 <int>, week50 <int>, week51 <int>,
## #   week52 <int>, week53 <int>, week54 <int>, week55 <int>, week56 <int>,
## #   week57 <int>, week58 <int>, week59 <int>, week60 <int>, week61 <int>,
## #   week62 <int>, week63 <int>, week64 <int>
```

- It never changes an input's type. With data frames, strings are often coerced to factors, and you have to set `stringsAsFactors = FALSE` in order to avoid this kind of coercion.

- It never changes the names of variables.

```
names(data.frame("crazy name" = 1))    # data frame changes space to dot
```

```
## [1] "crazy.name"

names(tibble("crazy name" = 1))       # tibble does not change the names

## [1] "crazy name"
```

- It does not allow setting `row.names`. Column names can be set using `names`.

```
row.names(tb) <- letters[1:5]

## Warning: Setting row names on a tibble is deprecated.
```

- It clearly distinguishes between `[` and `[[`. With tibbles, `[` always returns another *tibble*.

```
# data frame:
billboard[1:3, "week1"]       # return a vector

## [1] 78 15 71

billboard[1:3, "week1", drop = FALSE]   # return a data frame

##   week1
## 1    78
## 2    15
## 3    71

# tibble:
billboard_tbl[1:3, "week1"]   # return a tibble

## # A tibble: 3 x 1
##   week1
##   <int>
## 1    78
## 2    15
## 3    71
```

Remark: In the subsequent lectures, we will use the term *tibble* and *data frame* interchcangeably when referring to tibbles.

6.3 `tidyr` for Data Tidying

`tidyr` functions help us *get data into the tidy data format*.

Two fundamental operations for data tidying:

- Converting data between wide and long formats.
  - `pivot_longer()` makes "wide" data longer.
  - `pivot_wider()` makes "long" data wider.
- Splitting and combining character columns.

- separate() splits a single column (that represents multiple variables) into multiple columns.

- unite() combines multiple columns into a single column.

6.3.1 pivot_longer(): Make a Wide Data Longer

In the following example, we will use a simpler dataset billboard2.csv, which only has 5 rows and 5 columns.

```
billboard <- read.csv("billboard2.csv")
billboard_tbl <- as_tibble(billboard)
billboard_tbl   # wide format
```

```
## # A tibble: 5 x 5
##   track                 date.entered week1 week2 week3
##   <chr>                 <chr>        <int> <int> <int>
## 1 What A Girl Wants     11/27/1999      71    51    28
## 2 With Arms Wide Open   5/13/2000       84    78    76
## 3 Try Again             3/18/2000       59    53    38
## 4 Thank God I Found You 12/11/1999      82    68    50
## 5 Breathe               11/6/1999       81    68    62
```

This data set is stored in the *wide* format.

Use the function pivot_longer() to turn it into a *long* format:

```
billboard_tbl_l <- pivot_longer(billboard_tbl, cols = week1:week3, names_to =
"week", values_to = "position")
billboard_tbl_l   # long format
```

```
## # A tibble: 15 x 4
##    track                 date.entered week  position
##    <chr>                 <chr>        <chr>    <int>
##  1 What A Girl Wants     11/27/1999   week1       71
##  2 What A Girl Wants     11/27/1999   week2       51
##  3 What A Girl Wants     11/27/1999   week3       28
##  4 With Arms Wide Open   5/13/2000    week1       84
##  5 With Arms Wide Open   5/13/2000    week2       78
##  6 With Arms Wide Open   5/13/2000    week3       76
##  7 Try Again             3/18/2000    week1       59
##  8 Try Again             3/18/2000    week2       53
##  9 Try Again             3/18/2000    week3       38
## 10 Thank God I Found You 12/11/1999   week1       82
## 11 Thank God I Found You 12/11/1999   week2       68
## 12 Thank God I Found You 12/11/1999   week3       50
## 13 Breathe               11/6/1999    week1       81
## 14 Breathe               11/6/1999    week2       68
## 15 Breathe               11/6/1999    week3       62
```

Usage:

```
pivot_longer(
  data,
  cols,
  names_to = "name",
  values_to = "value",
  values_drop_na = FALSE,
  ...
)
```

- pivot_longer() lengthens data, increasing the number of rows and decreasing the number of columns.

- data: the data frame to pivot.

- cols: the columns to pivot into longer format. The specification syntax is flexible, e.g., select all columns between x and z with x:z (inclusive), exclude y with -y.

- names_to: the name of the new column(s) created from the data stored in the column names of data.

- values_to: the name of the new column created from the data stored as the cell values of data.

- values_drop_na: whether missing values to be removed.

6.3.2 pivot_wider(): Make a Long Data Wider

Use pivot_wider() to turn billboard_tbl_l (*long* format) into a *wide* format:

```
billboard_tbl_l

## # A tibble: 15 x 4
##    track                date.entered week  position
##    <chr>                <chr>        <chr>    <int>
##  1 What A Girl Wants    11/27/1999   week1       71
##  2 What A Girl Wants    11/27/1999   week2       51
##  3 What A Girl Wants    11/27/1999   week3       28
##  4 With Arms Wide Open  5/13/2000    week1       84
##  5 With Arms Wide Open  5/13/2000    week2       78
##  6 With Arms Wide Open  5/13/2000    week3       76
##  7 Try Again            3/18/2000    week1       59
##  8 Try Again            3/18/2000    week2       53
##  9 Try Again            3/18/2000    week3       38
## 10 Thank God I Found You 12/11/1999   week1       82
## 11 Thank God I Found You 12/11/1999   week2       68
## 12 Thank God I Found You 12/11/1999   week3       50
## 13 Breathe              11/6/1999    week1       81
## 14 Breathe              11/6/1999    week2       68
## 15 Breathe              11/6/1999    week3       62
```

```
billboard_tbl_w <- pivot_wider(billboard_tbl_l, names_from = week,
values_from = position)
billboard_tbl_w
```

```
## # A tibble: 5 x 5
##   track                 date.entered week1 week2 week3
##   <chr>                 <chr>        <int> <int> <int>
## 1 What A Girl Wants     11/27/1999      71    51    28
## 2 With Arms Wide Open   5/13/2000       84    78    76
## 3 Try Again             3/18/2000       59    53    38
## 4 Thank God I Found You 12/11/1999      82    68    50
## 5 Breathe               11/6/1999       81    68    62
```

Usage:

```
pivot_wider(
  data,
  names_from = name,
  values_from = value,
  values_fill = NULL,
  ...
)
```

- `pivot_wider()` widens data, increasing the number of columns and decreasing the number of rows.
- `data`: a data frame to pivot.
- `names_from`: the column(s) to get the name of the output columns.
- `values_from`: the column(s) we want to use to populate the output columns.
- `values_fill`: what values to substitute if there are combinations that don't exist.

[Task 1: Converting Data between Wide and Long Formats]

Downlaod the dataset `weeklyprice.csv` from Canvas, which contains time series data for stock prices for three large Internet companies (Facebook, Google, and Amazon) from Oct. 30, 2012 to Oct. 30, 2017.

Load the dataset to create a tibble named `weekly` using `read_csv()` from the `readr` package (the data import package in the `tidyverse`; return a tibble).

```
weekly <- read_csv("weeklyprice.csv", col_names = TRUE, col_types = cols())
weekly
```

```
## # A tibble: 3 x 263
##   company `2012-10-29` `2012-11-05` `2012-11-12` `2012-11-19` `2012-11-26`
##   <chr>          <dbl>        <dbl>        <dbl>        <dbl>        <dbl>
## 1 FB              21.2         19.2         23.6           24           28
## 2 GOOG          342.         329.         321.          332.         347.
## 3 AMZN          232.         226.         225.          240.         252.
## # … with 257 more variables: `2012-12-03` <dbl>, `2012-12-10` <dbl>,
## #   `2012-12-17` <dbl>, `2012-12-24` <dbl>, `2012-12-31` <dbl>,
## #   `2013-01-07` <dbl>, `2013-01-14` <dbl>, `2013-01-21` <dbl>,
## #   `2013-01-28` <dbl>, `2013-02-04` <dbl>, `2013-02-11` <dbl>,
## #   `2013-02-18` <dbl>, `2013-02-25` <dbl>, `2013-03-04` <dbl>,
## #   `2013-03-11` <dbl>, `2013-03-18` <dbl>, `2013-03-25` <dbl>,
## #   `2013-04-01` <dbl>, `2013-04-08` <dbl>, `2013-04-15` <dbl>,
## #   `2013-04-22` <dbl>, `2013-04-29` <dbl>, `2013-05-06` <dbl>,
## #   `2013-05-13` <dbl>, `2013-05-20` <dbl>, `2013-05-27` <dbl>,
## #   `2013-06-03` <dbl>, `2013-06-10` <dbl>, `2013-06-17` <dbl>,
## #   `2013-06-24` <dbl>, `2013-07-01` <dbl>, `2013-07-08` <dbl>,
## #   `2013-07-15` <dbl>, `2013-07-22` <dbl>, `2013-07-29` <dbl>,
## #   `2013-08-05` <dbl>, `2013-08-12` <dbl>, `2013-08-19` <dbl>,
## #   `2013-08-26` <dbl>, `2013-09-02` <dbl>, `2013-09-09` <dbl>,
## #   `2013-09-16` <dbl>, `2013-09-23` <dbl>, `2013-09-30` <dbl>,
## #   `2013-10-07` <dbl>, `2013-10-14` <dbl>, `2013-10-21` <dbl>,
## #   `2013-10-28` <dbl>, `2013-11-04` <dbl>, `2013-11-11` <dbl>,
## #   `2013-11-18` <dbl>, `2013-11-25` <dbl>, `2013-12-02` <dbl>,
## #   `2013-12-09` <dbl>, `2013-12-16` <dbl>, `2013-12-23` <dbl>,
## #   `2013-12-30` <dbl>, `2014-01-06` <dbl>, `2014-01-13` <dbl>,
## #   `2014-01-20` <dbl>, `2014-01-27` <dbl>, `2014-02-03` <dbl>,
## #   `2014-02-10` <dbl>, `2014-02-17` <dbl>, `2014-02-24` <dbl>,
## #   `2014-03-03` <dbl>, `2014-03-10` <dbl>, `2014-03-17` <dbl>,
## #   `2014-03-24` <dbl>, `2014-03-31` <dbl>, `2014-04-07` <dbl>,
## #   `2014-04-14` <dbl>, `2014-04-21` <dbl>, `2014-04-28` <dbl>,
## #   `2014-05-05` <dbl>, `2014-05-12` <dbl>, `2014-05-19` <dbl>,
## #   `2014-05-26` <dbl>, `2014-06-02` <dbl>, `2014-06-09` <dbl>,
## #   `2014-06-16` <dbl>, `2014-06-23` <dbl>, `2014-06-30` <dbl>,
## #   `2014-07-07` <dbl>, `2014-07-14` <dbl>, `2014-07-21` <dbl>,
## #   `2014-07-28` <dbl>, `2014-08-04` <dbl>, `2014-08-11` <dbl>,
## #   `2014-08-18` <dbl>, `2014-08-25` <dbl>, `2014-09-01` <dbl>,
## #   `2014-09-08` <dbl>, `2014-09-15` <dbl>, `2014-09-22` <dbl>,
## #   `2014-09-29` <dbl>, `2014-10-06` <dbl>, `2014-10-13` <dbl>,
## #   `2014-10-20` <dbl>, `2014-10-27` <dbl>, …
```

It is a wide format with 3 rows and 263 columns. Each row stores weekly closing prices for a stock during the aforementioned period.

(a) If we are interested in investigating the fluctuation of weekly stock prices, we need to transform the data from the wide format to the long format. Save the resulting tibble as `weekly_long`, which should have 3 columns, `company`, `date`, and `price`.

(b) Transform the data from the long format (`weekly_long`) back to the wide format. But this time, use company names as the column headers. The resulting tibble should have 4 columns, `date`, `FB`, `GOOG`, and `AMZN`.

[End of Task 1]

6.3.3 `separate()`: Separate a Character Column into Multiple Columns

Use the function `separate()` to separate the column `date.entered` in `billboard_tbl_l` into three columns `month`, day and year:

```
billboard_tbl_l   # one column: `date.entered`

## # A tibble: 15 x 4
##    track                 date.entered week  position
##    <chr>                 <chr>        <chr>    <int>
##  1 What A Girl Wants     11/27/1999   week1       71
##  2 What A Girl Wants     11/27/1999   week2       51
##  3 What A Girl Wants     11/27/1999   week3       28
##  4 With Arms Wide Open   5/13/2000    week1       84
##  5 With Arms Wide Open   5/13/2000    week2       78
##  6 With Arms Wide Open   5/13/2000    week3       76
##  7 Try Again             3/18/2000    week1       59
##  8 Try Again             3/18/2000    week2       53
##  9 Try Again             3/18/2000    week3       38
## 10 Thank God I Found You 12/11/1999   week1       82
## 11 Thank God I Found You 12/11/1999   week2       68
## 12 Thank God I Found You 12/11/1999   week3       50
## 13 Breathe               11/6/1999    week1       81
## 14 Breathe               11/6/1999    week2       68
## 15 Breathe               11/6/1999    week3       62

billboard_tbl_ls <- separate(billboard_tbl_l, col = date.entered, into =
c("month", "day", "year"))
billboard_tbl_ls  # three columns: `month`, `day` and `year`

## # A tibble: 15 x 6
##    track                 month day   year  week  position
##    <chr>                 <chr> <chr> <chr> <chr>    <int>
##  1 What A Girl Wants     11    27    1999  week1       71
##  2 What A Girl Wants     11    27    1999  week2       51
##  3 What A Girl Wants     11    27    1999  week3       28
##  4 With Arms Wide Open   5     13    2000  week1       84
##  5 With Arms Wide Open   5     13    2000  week2       78
##  6 With Arms Wide Open   5     13    2000  week3       76
##  7 Try Again             3     18    2000  week1       59
##  8 Try Again             3     18    2000  week2       53
##  9 Try Again             3     18    2000  week3       38
## 10 Thank God I Found You 12    11    1999  week1       82
## 11 Thank God I Found You 12    11    1999  week2       68
## 12 Thank God I Found You 12    11    1999  week3       50
## 13 Breathe               11    6     1999  week1       81
## 14 Breathe               11    6     1999  week2       68
## 15 Breathe               11    6     1999  week3       62
```

Usage:

```
separate(
  data,
  col,
  into,
  sep = "[^[:alnum:]]+",
  ...
)
```

- `col`: the column to be split apart.
- `into`: the names of new variables to create as character vectors.
- `sep`: the separator between columns.
  - specified either by a *regular expression* (more on it later) or by *position* (e.g., `sep = c(3, -5)`);
  - If character, `sep` is interpreted as a *regular expression*. The default value is a regular expression that matches *non-alphanumeric characters*.
  - If numeric, `sep` is interpreted as *character positions* to split at. Positive values start at 1 at the far-left of the string; negative value start at -1 at the far-right of the string.

6.3.4 `unite()`: Unite Multiple Columns into One by Pasting Strings Together

```
billboard_tbl_ls    # three columns: `month`, `day` and `year`

## # A tibble: 15 x 6
##    track                 month day   year  week  position
##    <chr>                 <chr> <chr> <chr> <chr>    <int>
##  1 What A Girl Wants     11    27    1999  week1       71
##  2 What A Girl Wants     11    27    1999  week2       51
##  3 What A Girl Wants     11    27    1999  week3       28
##  4 With Arms Wide Open   5     13    2000  week1       84
##  5 With Arms Wide Open   5     13    2000  week2       78
##  6 With Arms Wide Open   5     13    2000  week3       76
##  7 Try Again             3     18    2000  week1       59
##  8 Try Again             3     18    2000  week2       53
##  9 Try Again             3     18    2000  week3       38
## 10 Thank God I Found You 12    11    1999  week1       82
## 11 Thank God I Found You 12    11    1999  week2       68
## 12 Thank God I Found You 12    11    1999  week3       50
## 13 Breathe               11    6     1999  week1       81
## 14 Breathe               11    6     1999  week2       68
## 15 Breathe               11    6     1999  week3       62

billboard_tbl_lu <- unite(billboard_tbl_ls, col = date.entered, day, month,
year, sep = "/")
billboard_tbl_lu    # one column: `date.entered`
```

```
## # A tibble: 15 x 4
##    track                  date.entered week  position
##    <chr>                  <chr>        <chr>    <int>
##  1 What A Girl Wants      27/11/1999   week1       71
##  2 What A Girl Wants      27/11/1999   week2       51
##  3 What A Girl Wants      27/11/1999   week3       28
##  4 With Arms Wide Open    13/5/2000    week1       84
##  5 With Arms Wide Open    13/5/2000    week2       78
##  6 With Arms Wide Open    13/5/2000    week3       76
##  7 Try Again              18/3/2000    week1       59
##  8 Try Again              18/3/2000    week2       53
##  9 Try Again              18/3/2000    week3       38
## 10 Thank God I Found You 11/12/1999   week1       82
## 11 Thank God I Found You 11/12/1999   week2       68
## 12 Thank God I Found You 11/12/1999   week3       50
## 13 Breathe                6/11/1999    week1       81
## 14 Breathe                6/11/1999    week2       68
## 15 Breathe                6/11/1999    week3       62
```

Usage:

```
unite(data, col, ..., sep = "_", remove = TRUE, na.rm = FALSE)
```

- `col`: the name of the new column, as a string or symbol.
- `...`: columns to unite.
- sep: separator to use between values.

[Task 2: Splitting and Combining Character Columns]

Continue the weekly stock prices problem in Task 1.

(a) Suppose we want to calculate monthly stock prices by averaging weekly prices. To do so, we need to first separate different parts of date values and store them as different variables. Work on `weekly_long` to get the following tibble and name it `monthly_long`:

```
# A tibble: 786 x 3
   company month   price
   <chr>   <chr>   <dbl>
 1 FB      2012-10  21.2
 2 FB      2012-11  19.2
 3 FB      2012-11  23.6
 4 FB      2012-11  24
 5 FB      2012-11  28
 6 FB      2012-12  27.5
 7 FB      2012-12  26.8
 8 FB      2012-12  26.3
 9 FB      2012-12  25.9
```

```
10 FB      2012-12  28.8
# . with 776 more rows
```

Tips: split a date value by "-", drop the day part, and combine the year and month parts.

(b) Transform monthly_long from the long format to the wide format. call pivot_wider() again as you did in Task 1(b). But this time, include values_fn = list(price = mean) in the argument list.

[End of Task 2]

6.4 The Pipe Operator %>%

The magrittr package provides the pipe operator (%>%), which passes the result of one step (the left-hand side) as input for the next step (the right-hand side).

```
x %>% f         # equivalent to f(x)
```

First-argument rule:

```
x %>% f(y)      # equivalent to f(x, y)
```

The **argument placeholder** specifies where the piped input should land:

```
x %>% f(y, .)      # equivalent to f(y, x)
x %>% f(y, z=.)    # equivalent to f(y, z=x)
```

When the placeholder only appears in a nested expressions, the first-argument rule still applies:

```
x %>% f(y = nrow(.), z = ncol(.))    # equivalent to f(x, y = nrow(x), z =
ncol(x))
```

In the above code, x is passed as the first argument of f(), and also to the place specified by ..

The behavior can be overruled by enclosing the right-hand side in braces:

```
x %>% {f(y = nrow(.), z = ncol(.))}   # equivalent to f(y = nrow(x), z =
ncol(x))
```

With %>%, we can process a data-object using *a sequence of operations* rather than *nested function calls*.

```
x %>% f %>% g %>% h   # equivalent to h(g(f(x)))
```

The intended aim of pipe operators is to *increase human readability* of written code.

It helps avoid nested function calls, minimize the need for local variables and function definitions, and add steps anywhere in the sequence of operations.

`%>%` is used throughout `tidyverse` and loaded automatically when using packages in it.

```
billboard_tbl %>% pivot_longer(cols = week1:week3, names_to = "week",
values_to = "position") %>% separate(date.entered, into = c("year", "month",
"day")) %>% subset(week == "week1") %>% .$position %>% mean

## [1] 75.4
```

Saving Pipelines

The input to the pipeline can itself be a placeholder:

```
num_unique <- . %>% unique %>% length
```

In this case, the pipeline describes a *function chain* that can be saved and re-used. It also has a different print method:

```
num_unique     # functional sequence

## Functional sequence with the following components:
##
##  1. unique(.)
##  2. length(.)
##
## Use 'functions' to extract the individual functions.
```

Apply the functional sequence to a data object:

```
x <- rep(1:5, times = c(2, 3, 2, 5, 2))
x %>% num_unique

## [1] 5

x %>% unique %>% length

## [1] 5

length(unique(x))

## [1] 5
```

[Task 3: Pipes to Base R]

For each of the following code blocks, which are written with pipes, write equivalent code in base R.

(a)

Pipes:

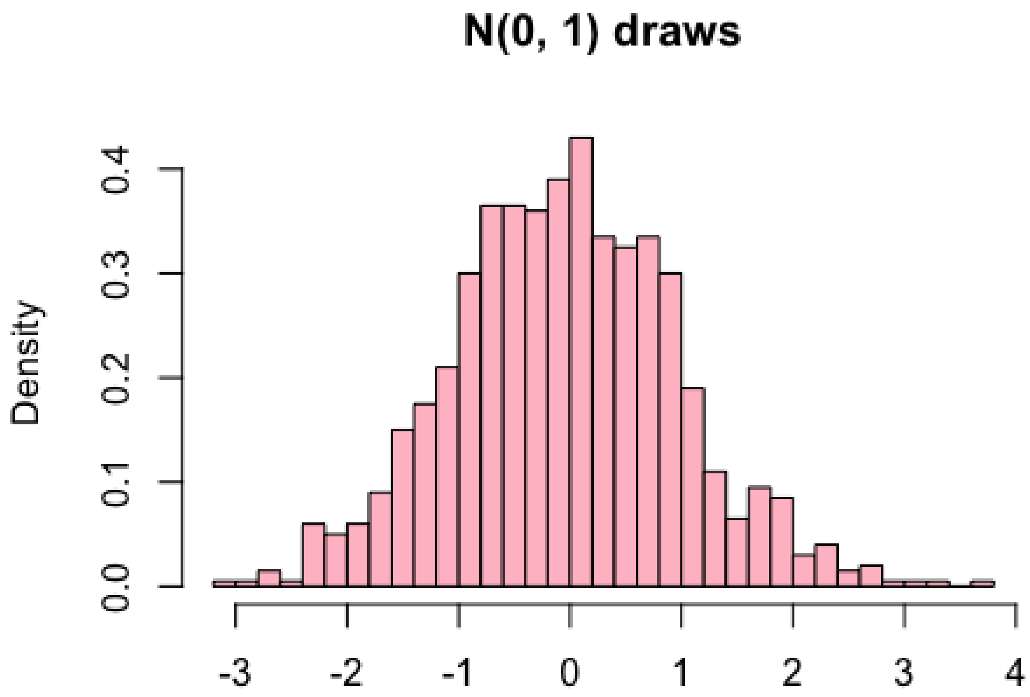
```
# `letters` is a built-in constant in R; ?letters
# `toupper` is a built-in function in R; ?toupper
# it is always good to build up your vocabulary of built-in functions
letters %>% toupper %>% paste(collapse="+")

## [1] "A+B+C+D+E+F+G+H+I+J+K+L+M+N+O+P+Q+R+S+T+U+V+W+X+Y+Z"
```

Base R:

(b)

Pipes:

```
rnorm(1000) %>% hist(breaks=30, main="N(0, 1) draws", col="pink", prob=TRUE)
```



.

Base R:

(c)

Pipes:

```
rnorm(1000) %>%  hist(breaks=30, plot=FALSE) %>% .$density %>% max

## [1] 0.405
```

Base R:

[End of Task 3]

[Task 4: Base R to Pipes]

For each of the following code blocks, which are written in base R, write equivalent code with pipes (to do the same thing).

(a) Tips: use the dot `.` as seen in the lecture notes.

Base R:

```
paste("Your grade is", sample(c("A", "B", "C", "D", "F"), size = 1))

## [1] "Your grade is C"
```

Pipes:

(b) Tips: use the dot `.` again, in order to index `state.name` directly in the last pipe command.

Base R:

```
# state.name and state.x77 are built-in datasets; ?state.x77; ?state.name;
?which.max
state.name[which.max(state.x77[, "Illiteracy"])]

## [1] "Louisiana"
```

Pipes:

[End of Task 4]

6.5 `dplyr` for Data Manipulation

The package `dplyr` provides a consistent set of verbs that help streamline the data manipulation process.

- Single-table verbs:
  - `select(data, variables)`: pick variables based on their names.
  - `filter(data, conditions)`: pick observations based on conditions.
  - `arrange(data, variables)`: reorder observations according to variables.
  - `mutate(data, newvar = function)`: add new variables or transform existing variables.
  - `summarize(data, newvar = function)`: collapse many values down to a single summary.

  - `group_by(data, variables)`: group observations by variables.

- Two-table verbs: e.g., `inner_join(data1, data2, variables)`.

- Some other functions such as `slice()`, `rename()`, `transmute()`, `sample_n()` and `sample_frac()`, all of which we may find useful.

6.5.1 Select & Reorder Columns with `select()`

Let's first generate a data frame (tibble):

```
billboard <- as_tibble(read.csv("billboard.csv"))
billboard_l <- pivot_longer(billboard, starts_with("week"), names_to =
"week", values_to = "position", values_drop_na = TRUE)
billboard_l

## # A tibble: 5,306 x 8
##    artist      track         time  genre date.entered date.peaked week
position
##    <chr>       <chr>         <chr> <chr> <chr>        <chr>       <chr>
<int>
##  1 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week1
78
##  2 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week2
63
##  3 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week3
49
##  4 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week4
33
##  5 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week5
23
##  6 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week6
15
##  7 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week7
7
##  8 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week8
5
##  9 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week9
1
## 10 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week…
1
## # … with 5,296 more rows
```

We can use `select()` to select (and optionally rename) variables in the data frame.

Usage:

```
select(.data, ...)
```

- `.data`: a data frame

- `...`: one or more unquoted expressions separated by commas. Variable names can be used as if they were positions in the data frame, so expressions like `x:y` can be used to select a range of variables.

Example: From the tibble `billboard_1`, select the columns `track`, `week1` through `week3`, and `data.entered`, with the last being renamed to `date`:

```
billboard %>% select(track, week1:week3, date = date.entered)

## # A tibble: 317 x 5
##    track                                 week1 week2 week3 date
##    <chr>                                 <int> <int> <int> <chr>
##  1 Independent Women Part I                 78    63    49 9/23/2000
##  2 Maria, Maria                             15     8     6 2/12/2000
##  3 I Knew I Loved You                       71    48    43 10/23/1999
##  4 Music                                    41    23    18 8/12/2000
##  5 Come On Over Baby (All I Want Is You)    57    47    45 8/5/2000
##  6 Doesn't Really Matter                    59    52    43 6/17/2000
##  7 Say My Name                              83    83    44 12/25/1999
##  8 Be With You                              63    45    34 4/1/2000
##  9 Incomplete                               77    66    61 6/24/2000
## 10 Amazed                                   81    54    44 6/5/1999
## # … with 307 more rows
```

Alternatively, we can replace `week1:week3` using a *selection helper* `num_range`:

```
billboard %>% select(track, num_range("week", 1:3), date = date.entered)

## # A tibble: 317 x 5
##    track                                 week1 week2 week3 date
##    <chr>                                 <int> <int> <int> <chr>
##  1 Independent Women Part I                 78    63    49 9/23/2000
##  2 Maria, Maria                             15     8     6 2/12/2000
##  3 I Knew I Loved You                       71    48    43 10/23/1999
##  4 Music                                    41    23    18 8/12/2000
##  5 Come On Over Baby (All I Want Is You)    57    47    45 8/5/2000
##  6 Doesn't Really Matter                    59    52    43 6/17/2000
##  7 Say My Name                              83    83    44 12/25/1999
##  8 Be With You                              63    45    34 4/1/2000
##  9 Incomplete                               77    66    61 6/24/2000
## 10 Amazed                                   81    54    44 6/5/1999
## # … with 307 more rows
```

A variety of ***helper functions*** can be used to select variables based on their names.

- `starts_with("abc")` matches names that begin with "abc".

- `ends_with("abc")` matches names that end with "abc".

- `contains("abc")` matches names that contain "abc".

- `matches("(.)\\1")` selects variables that match a *regular expression* (more on this in Topic 9). This one matches any variables that contain repeated characters.

- `num_range("abc", 1:3)` matches `abc1`, `abc2` and `abc3`.

- `everything()`: all variables that haven't been specified.

```
billboard %>% select(track, date = date.entered, everything())  # reorder and
rename columns

## # A tibble: 317 x 70
##    track date  artist time  genre date.peaked week1 week2 week3 week4
week5
##    <chr> <chr> <chr>  <chr> <chr> <chr>       <int> <int> <int> <int>
<int>
##  1 Inde… 9/23… Desti… 3:38  Rock  11/18/2000     78    63    49    33
23
##  2 Mari… 2/12… Santa… 4:18  Rock  4/8/2000       15     8     6     5
2
##  3 I Kn… 10/2… Savag… 4:07  Rock  1/29/2000      71    48    43    31
20
##  4 Music 8/12… Madon… 3:45  Rock  9/16/2000      41    23    18    14
2
##  5 Come… 8/5/… Aguil… 3:38  Rock  10/14/2000     57    47    45    29
23
##  6 Does… 6/17… Janet  4:17  Rock  8/26/2000      59    52    43    30
29
##  7 Say … 12/2… Desti… 4:31  Rock  3/18/2000      83    83    44    38
16
##  8 Be W… 4/1/… Igles… 3:36  Latin 6/24/2000      63    45    34    23
17
##  9 Inco… 6/24… Sisqo  3:52  Rock  8/12/2000      77    66    61    61
61
## 10 Amaz… 6/5/… Lones… 4:25  Coun… 3/4/2000       81    54    44    39
38
## # … with 307 more rows, and 59 more variables: week6 <int>, week7 <int>,
## #   week8 <int>, week9 <int>, week10 <int>, week11 <int>, week12 <int>,
## #   week13 <int>, week14 <int>, week15 <int>, week16 <int>, week17 <int>,
## #   week18 <int>, week19 <int>, week20 <int>, week21 <int>, week22 <int>,
## #   week23 <int>, week24 <int>, week25 <int>, week26 <int>, week27 <int>,
## #   week28 <int>, week29 <int>, week30 <int>, week31 <int>, week32 <int>,
## #   week33 <int>, week34 <int>, week35 <int>, week36 <int>, week37 <int>,
## #   week38 <int>, week39 <int>, week40 <int>, week41 <int>, week42 <int>,
## #   week43 <int>, week44 <int>, week45 <int>, week46 <int>, week47 <int>,
## #   week48 <int>, week49 <int>, week50 <int>, week51 <int>, week52 <int>,
## #   week53 <int>, week54 <int>, week55 <int>, week56 <int>, week57 <int>,
## #   week58 <int>, week59 <int>, week60 <int>, week61 <int>, week62 <int>,
## #   week63 <int>, week64 <int>
```

- `last_col()`: last variable, possibly with an offset.

```
billboard %>% select(track, last_col())

## # A tibble: 317 x 2
##    track                                   week64
##    <chr>                                    <int>
##  1 Independent Women Part I                    NA
##  2 Maria, Maria                                NA
##  3 I Knew I Loved You                          NA
##  4 Music                                       NA
##  5 Come On Over Baby (All I Want Is You)       NA
##  6 Doesn't Really Matter                       NA
##  7 Say My Name                                 NA
##  8 Be With You                                 NA
##  9 Incomplete                                  NA
## 10 Amazed                                      50
## # … with 307 more rows

billboard %>% select(track, last_col(offset = 2))  #  Set offset=n to select
the nth var from the end

## # A tibble: 317 x 2
##    track                                   week62
##    <chr>                                    <int>
##  1 Independent Women Part I                    NA
##  2 Maria, Maria                                NA
##  3 I Knew I Loved You                          NA
##  4 Music                                       NA
##  5 Come On Over Baby (All I Want Is You)       NA
##  6 Doesn't Really Matter                       NA
##  7 Say My Name                                 NA
##  8 Be With You                                 NA
##  9 Incomplete                                  NA
## 10 Amazed                                      42
## # … with 307 more rows
```

6.5.2 Subset Rows with `filter()`

Usage:

```
filter(.data, ..., .preserve = FALSE)
```

- `...`: Expressions that return a logical value, and are defined in terms of the variables in `.data`. If there are multiple expressions separated by commas, they are combined with the & operator.

Example: Look at the weekly ranks of a subset of songs (by the first 10 artists), and only look at the weeks when they rank higher than the average rank of all songs:

```
billboard_l %>% filter(artist %in% unique(artist)[1:10] & position <
mean(position))
```

```
## # A tibble: 499 x 8
##    artist      track          time  genre date.entered date.peaked week 
position
##    <chr>       <chr>          <chr> <chr> <chr>        <chr>       <chr>
<int>
##  1 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week3
49
##  2 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week4
33
##  3 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week5
23
##  4 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week6
15
##  5 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week7
7
##  6 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week8
5
##  7 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week9
1
##  8 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week…
1
##  9 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week…
1
## 10 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week…
1
## # … with 489 more rows
```

The operator `%in%` is a binary operator, indicating if there is a match or not for its left operand. `?'%in%'` for more information.

6.5.3 Rearrange Rows with `arrange()`

Usage:

```
arrange(.data, ..., .by_group = FALSE)
```

- `arrange()` orders the rows of a data frame by the values of selected columns.
- Use `desc()` to sort a variable in descending order.

Example: Sort weekly ranking positions first by `artist`, second by `track`, and last by `position` (descending order):

```
billboard_l %>% arrange(artist, track, desc(position))
```

```
## # A tibble: 5,306 x 8
##    artist  track              time  genre date.entered date.peaked week 
position
##    <chr>   <chr>              <chr> <chr> <chr>        <chr>       <chr>
<int>
##  1 2 Pac   Baby Don't Cry (… 4:22  Rap   2/26/2000    3/11/2000   week7
99
##  2 2 Pac   Baby Don't Cry (… 4:22  Rap   2/26/2000    3/11/2000   week6
94
##  3 2 Pac   Baby Don't Cry (… 4:22  Rap   2/26/2000    3/11/2000   week1
87
##  4 2 Pac   Baby Don't Cry (… 4:22  Rap   2/26/2000    3/11/2000   week5
87
##  5 2 Pac   Baby Don't Cry (… 4:22  Rap   2/26/2000    3/11/2000   week2
82
##  6 2 Pac   Baby Don't Cry (… 4:22  Rap   2/26/2000    3/11/2000   week4
77
##  7 2 Pac   Baby Don't Cry (… 4:22  Rap   2/26/2000    3/11/2000   week3
72
##  8 2Ge+her The Hardest Part… 3:15  R&B   9/2/2000     9/9/2000    week3
92
##  9 2Ge+her The Hardest Part… 3:15  R&B   9/2/2000     9/9/2000    week1
91
## 10 2Ge+her The Hardest Part… 3:15  R&B   9/2/2000     9/9/2000    week2
87
## # … with 5,296 more rows
```

6.5.4 Add, Modify, and Delete Columns with `mutate()` and `transmute()`

- `mutate()` adds new variables and *preserves* existing ones.
- `transmute()` adds new variables and *drops* existing ones.
- New variables overwrite existing variables of the same name.
- Variables can be removed by setting their value to `NULL`.

Example: Calculate changes in ranking positions between two adjacent weeks with `mutate()`:

```
billboard_l %>% filter(track=="Maria, Maria") %>% .[c("track", "week",
"position")] %>% mutate(current = position, previous = lag(position), change
= previous - current, week = as.integer(str_extract(week, "[\\d]+")),
position = NULL)
```

```
## # A tibble: 26 x 5
##    track         week current previous change
##    <chr>        <int>   <int>    <int>  <int>
##  1 Maria, Maria     1      15       NA     NA
##  2 Maria, Maria     2       8       15      7
##  3 Maria, Maria     3       6        8      2
##  4 Maria, Maria     4       5        6      1
##  5 Maria, Maria     5       2        5      3
##  6 Maria, Maria     6       3        2     -1
##  7 Maria, Maria     7       2        3      1
##  8 Maria, Maria     8       2        2      0
##  9 Maria, Maria     9       1        2      1
## 10 Maria, Maria    10       1        1      0
## # … with 16 more rows
```

Alternatively, we can use `transmute()`, no need to set `positiion = NULL`:

```
billboard_l %>% filter(track=="Maria, Maria") %>% .[c("track", "week",
"position")] %>% transmute(current = position, previous = lag(position),
change = previous - position, week = as.integer(str_extract(week, "[\\d]+")))
```

```
## # A tibble: 26 x 4
##    current previous change  week
##      <int>    <int>  <int> <int>
##  1      15       NA     NA     1
##  2       8       15      7     2
##  3       6        8      2     3
##  4       5        6      1     4
##  5       2        5      3     5
##  6       3        2     -1     6
##  7       2        3      1     7
##  8       2        2      0     8
##  9       1        2      1     9
## 10       1        1      0    10
## # … with 16 more rows
```

In the above codes, `lag()` returns the *previous* values in a vector. Another useful function is `lead()` which returns the *next* values in a vector:

```
x <- 1:5
tibble(x_behind = lag(x), x, x_ahead = lead(x))
```

```
## # A tibble: 5 x 3
##   x_behind     x x_ahead
##      <int> <int>   <int>
## 1       NA     1       2
## 2        1     2       3
## 3        2     3       4
## 4        3     4       5
## 5        4     5      NA
```

`log()` and `lead()` are two examples of **window functions**.

A window function takes `n` inputs and returns `n` values. Examples include:

- Ranking and ordering functions: `row_number()`, `min_rank()`, `dense_rank()`, `cume_dist()`, `percent_rank()`, and `ntile()`.

- Offsets `lead()` and `lag()` allow us to compute differences and trends.

- Cumulative aggregates: `cumsum()`, `cummin()`, `cummax()` from base R; `cumall()`, `cumany()`, and `cummean()` from `dplyr`.

6.5.5 Group Data by `group_by()`

- Most data operations are done on *groups* defined by variables.
- `group_by()` takes an existing `tbl` and converts it into a grouped `tbl` where *operations are performed "by group"*.
- `ungroup()` removes grouping.

Example: Group weekly ranking positions first by `week` and then by `artist`:

```
billboard_l %>% group_by(week, artist)
```

```
## # A tibble: 5,306 x 8
## # Groups:   week, artist [3,988]
##    artist      track          time  genre date.entered date.peaked week
position
##    <chr>       <chr>          <chr> <chr> <chr>        <chr>       <chr>
<int>
##  1 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week1
78
##  2 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week2
63
##  3 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week3
49
##  4 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week4
33
##  5 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week5
23
##  6 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week6
15
##  7 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week7
7
##  8 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week8
5
##  9 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week9
1
## 10 Destiny's … Independent … 3:38  Rock  9/23/2000    11/18/2000  week…
1
## # … with 5,296 more rows
```

Compared to the original tbl, the grouped tbl shows "Groups: week, artist [3,988]".

```
billboard_l %>% group_by(week, artist) %>% groups      # return a list of
group names
```

```
## [[1]]
## week
##
## [[2]]
## artist
```

```
billboard_l %>% group_by(week, artist) %>% group_vars  # return a character
vector of group names

## [1] "week"   "artist"
```

By default, `group_by()` overrides existing grouping:

```
billboard_l %>% group_by(week, artist) %>% group_by(track) %>% group_vars

## [1] "track"
```

Set `.add = TRUE` to instead append grouping:

```
billboard_l %>% group_by(week, artist) %>% group_by(track, .add = TRUE) %>%
group_vars

## [1] "week"   "artist" "track"
```

Use `ungroup()` to removing grouping:

```
billboard_l %>% group_by(week, artist) %>% ungroup %>% group_vars

## character(0)
```

[Task 5: Fastest 100m sprint times]

We read in a data frame `sprint.m.df` containing of the fastest times ever recorded for the 100m sprint in men's track.

```
sprint.m.df <- read_tsv("sprint.m.dat", col_names = TRUE)

## Parsed with column specification:
## cols(
##   Rank = col_double(),
##   Time = col_double(),
##   Wind = col_double(),
##   Name = col_character(),
##   Country = col_character(),
##   Birthdate = col_character(),
##   City = col_character(),
##   Date = col_character()
## )

sprint.m.df

## # A tibble: 2,988 x 8
##     Rank  Time  Wind Name         Country Birthdate City          Date
##    <dbl> <dbl> <dbl> <chr>        <chr>   <chr>     <chr>         <chr>
##  1     1  9.58   0.9 Usain Bolt   JAM     21.08.86  Berlin
16.08.2009
##  2     2  9.63   1.5 Usain Bolt   JAM     21.08.86  London
```

```
05.08.2012
##  3     3  9.69   0   Usain Bolt    JAM     21.08.86  Beijing
16.08.2008
##  4     3  9.69   2   Tyson Gay     USA     09.08.82  Shanghai
20.09.2009
##  5     3  9.69  -0.1 Yohan Blake   JAM     26.12.89  Lausanne
23.08.2012
##  6     6  9.71   0.9 Tyson Gay     USA     09.08.82  Berlin
16.08.2009
##  7     7  9.72   1.7 Usain Bolt    JAM     21.08.86  New York City
31.05.2008
##  8     7  9.72   0.2 Asafa Powell  JAM     23.11.82  Lausanne
02.09.2008
##  9     9  9.74   1.7 Asafa Powell  JAM     23.11.82  Rieti
09.09.2007
## 10     9  9.74   0.9 Justin Gatlin USA     10.02.82  Ad-Dawhah
15.05.2015
## # … with 2,978 more rows
```

Compute, for each country, the quadruple: (Name, City, Country, Time) corresponding to the athlete with the fastest time among athletes from that country, and display the first 10 results ordered by increasing time. If there are ties, then show all the results that correspond to the fastest time.

Tips: group by `Country`; find all the results with the fastest time for each country; and arrange the results by increasing time.

[End of Task 5]

6.5.6 Split-and-Apply Data Analysis with `group_by()` and `summarise()`

Form a summary table showing the number of weeks on chart, average ranking, and peak position of *each track*:

```
billboard_l %>% group_by(track) %>% summarise("weeks on chart" = n(),
"average position" = mean(position), "peak position" = min(position))

## `summarise()` ungrouping output (override with `.groups` argument)

## # A tibble: 316 x 4
##    track                           `weeks on chart` `average positio… `peak
position`
##    <chr>                                      <int>             <dbl>
<int>
##  1 (Hot S**t) Country Grammar                    34              30.9
7
##  2 3 Little Words                                 9              94.4
89
##  3 911                                           19              60
```

```
38
##  4 A Country Boy Can Survive                 3              86.7
75
##  5 A Little Gasoline                         6              89.8
75
##  6 A Puro Dolor (Purest Of P…               26              49.5
26
##  7 Aaron's Party (Come Get I…               15              66.3
35
##  8 Absolutely (Story Of A Gi…               27              26.5
6
##  9 All Good?                                 3              97.3
96
## 10 All The Small Things                     23              33.3
6
## # … with 306 more rows
```

Calculate *every artist's* best *weekly* ranking (an artist may have multiple tracks):

```
billboard_l %>% group_by(week, artist) %>% summarise("peak position" =
min(position))

## `summarise()` regrouping output by 'week' (override with `.groups`
argument)

## # A tibble: 3,988 x 3
## # Groups:   week [64]
##    week  artist              `peak position`
##    <chr> <chr>                         <int>
##  1 week1 2 Pac                            87
##  2 week1 2Ge+her                          91
##  3 week1 3 Doors Down                     76
##  4 week1 504 Boyz                         57
##  5 week1 98?                              51
##  6 week1 A*Teens                          97
##  7 week1 Aaliyah                          59
##  8 week1 Adams, Yolanda                   76
##  9 week1 Adkins, Trace                    84
## 10 week1 Aguilera, Christina              50
## # … with 3,978 more rows
```

Each call to summarise() removes a layer of grouping.

```
billboard_l %>% group_by(week, artist) %>% summarise("peak position" =
min(position)) %>% group_vars()

## `summarise()` regrouping output by 'week' (override with `.groups`
argument)

## [1] "week"
```

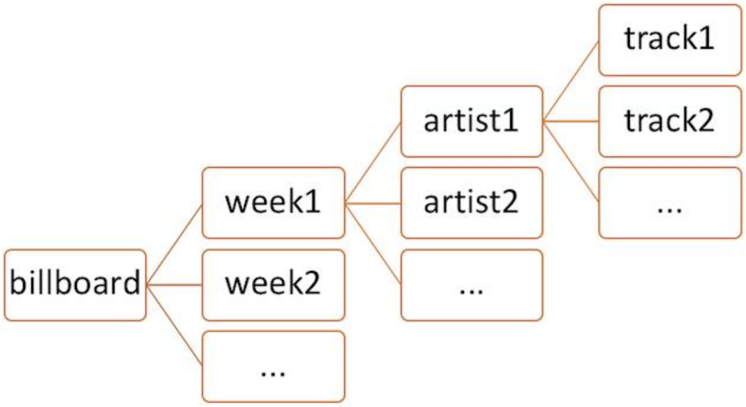


Fig 2. Group and Summarise

In the above codes, `n()`, `mean()`, and `min()` are examples of **aggregation functions**.

An aggregation function takes `n` inputs and return a single value. Examples inlcude:

- `min()`, `max()`, ..., `median()`, etc. from base R.
- `n()`: the number of observations in the current group.
- `n_distinct(x)` :the number of unique values in x.
- `first(x)`, `last(x)` and `nth(x, n)`
    - work similarly to `x[1]`, `x[length(x)]`, and `x[n]` but give more control over the result.

6.5.7 By-Group Operations

All `dplyr` verbs discussed so far can combine naturally with `group_by()` to enable "by group" operations.

*** Grouped `arrange()`***

By default, it ignores groups:

```
billboard_l %>% group_by(track) %>% .[c("artist", "track", "week",
"position")] %>% arrange(desc(position))
```

```
## # A tibble: 5,306 x 4
## # Groups:   track [316]
##    artist               track                      week   position
##    <chr>                <chr>                      <chr>     <int>
##  1 Nelly                (Hot S**t) Country Grammar week1       100
##  2 Martin, Ricky        She Bangs                  week18      100
##  3 Lil Bow Wow          Bounce With Me             week22      100
##  4 Spears, Britney      Lucky                      week11      100
##  5 Hoku                 Another Dumb Blonde        week14      100
##  6 Sugar Ray            Falls Apart                  week20      100
##  7 Backstreet Boys, The The One                      week15      100
##  8 Carter, Aaron        Aaron's Party (Come Get It)  week15      100
##  9 No Doubt             Simple Kind Of Life          week12      100
## 10 Lawrence, Tracy      Lessons Learned              week20      100
## # … with 5,296 more rows
```

Unless we explicitly set `.by_group = TRUE`:

```
billboard_l %>% group_by(track) %>% .[c("artist", "track", "week",
"position")] %>% arrange(desc(position), .by_group = TRUE)
```

```
## # A tibble: 5,306 x 4
## # Groups:   track [316]
##    artist track                      week   position
##    <chr>  <chr>                      <chr>     <int>
##  1 Nelly  (Hot S**t) Country Grammar week1       100
##  2 Nelly  (Hot S**t) Country Grammar week2        99
##  3 Nelly  (Hot S**t) Country Grammar week3        96
##  4 Nelly  (Hot S**t) Country Grammar week4        76
##  5 Nelly  (Hot S**t) Country Grammar week5        55
##  6 Nelly  (Hot S**t) Country Grammar week34       49
##  7 Nelly  (Hot S**t) Country Grammar week6        37
##  8 Nelly  (Hot S**t) Country Grammar week11       37
##  9 Nelly  (Hot S**t) Country Grammar week32       37
## 10 Nelly  (Hot S**t) Country Grammar week10       36
## # … with 5,296 more rows
```

*** Grouped `mutate()` and `filter()`***

Most useful in conjunction with aggregation and window functions:

```
billboard_l %>% group_by(track) %>% .[c("track", "week", "position")] %>%
mutate("highest position" = cummin(position)) %>% ungroup %>% filter(track
%in% c("This Time Around", "American Pie")) %>% head(n = 20)
```

```
## # A tibble: 16 x 4
##    track            week  position `highest position`
##    <chr>            <chr>    <int>                <int>
##  1 This Time Around week1       22                   22
##  2 This Time Around week2       22                   22
##  3 This Time Around week3       20                   20
##  4 This Time Around week4       45                   20
##  5 This Time Around week5       87                   20
##  6 This Time Around week6       71                   20
##  7 This Time Around week7       95                   20
##  8 American Pie     week1       43                   43
##  9 American Pie     week2       35                   35
## 10 American Pie     week3       29                   29
## 11 American Pie     week4       29                   29
## 12 American Pie     week5       33                   29
## 13 American Pie     week6       32                  29
## 14 American Pie     week7       40                  29
## 15 American Pie     week8       58                  29
## 16 American Pie     week9       88                  29
```

```
billboard_l %>% group_by(track) %>% .[c("track", "week", "position")] %>%
filter(position < lag(position)) %>% ungroup %>% filter(track %in% c("This
Time Around", "American Pie")) %>% head(n = 20)
```

```
## # A tibble: 5 x 3
##   track            week  position
##   <chr>            <chr>    <int>
## 1 This Time Around week3       20
## 2 This Time Around week6       71
## 3 American Pie     week2       35
## 4 American Pie     week3       29
## 5 American Pie     week6       32
```

[Task 6: Practicing the `dplyr` verbs with flights data]

Read in the data from `flights.csv` and `airports.csv` to create two tibbles named `flights` and `airports`:

```
flights <- read_csv("flights.csv", col_names = TRUE, col_types = cols())
airports <- read_csv("airports.csv", col_names = TRUE, col_types = cols())
glimpse(flights)   # glimpse() is a transposed version of print
```

```
## Rows: 336,776
## Columns: 19
## $ year           <dbl> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013,
2013, …
## $ month          <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, …
## $ day            <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, …
## $ dep_time       <dbl> 517, 533, 542, 544, 554, 554, 555, 557, 557, 558,
558,…
## $ sched_dep_time <dbl> 515, 529, 540, 545, 600, 558, 600, 600, 600, 600,
600,…
## $ dep_delay      <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, -2, -2, -2, -2, -
2, -…
## $ arr_time       <dbl> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 753,
849…
## $ sched_arr_time <dbl> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 745,
851…
## $ arr_delay      <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2, -3,
7, -…
## $ carrier        <chr> "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV",
"B6", …
```

```
## $ flight         <dbl> 1545, 1714, 1141, 725, 461, 1696, 507, 5708, 79,
301, …
## $ tailnum        <chr> "N14228", "N24211", "N619AA", "N804JB", "N668DN",
"N39…
## $ origin         <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR",
"LGA"…
## $ dest           <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL",
"IAD"…
## $ air_time       <dbl> 227, 227, 160, 183, 116, 150, 158, 53, 140, 138,
149, …
## $ distance       <dbl> 1400, 1416, 1089, 1576, 762, 719, 1065, 229, 944,
733,…
## $ hour           <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6,
6, …
## $ minute         <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, 0,
59, …
## $ time_hour      <dttm> 2013-01-01 05:00:00, 2013-01-01 05:00:00, 2013-01-
01 …
```

```
summary(flights)
```

```
##       year          month             day           dep_time   
sched_dep_time
##  Min.   :2013   Min.   : 1.000   Min.   : 1.00   Min.   :   1   Min.   :
106
##  1st Qu.:2013   1st Qu.: 4.000   1st Qu.: 8.00   1st Qu.: 907   1st Qu.:
906
##  Median :2013   Median : 7.000   Median :16.00   Median :1401   Median
:1359
##  Mean   :2013   Mean   : 6.549   Mean   :15.71   Mean   :1349   Mean
:1344
##  3rd Qu.:2013   3rd Qu.:10.000   3rd Qu.:23.00   3rd Qu.:1744   3rd
Qu.:1729
##  Max.   :2013   Max.   :12.000   Max.   :31.00   Max.   :2400   Max.
:2359
##                                                  NA's   :8255
##    dep_delay          arr_time    sched_arr_time   arr_delay
##  Min.   : -43.00   Min.   :   1   Min.   :   1   Min.   : -86.000
##  1st Qu.:  -5.00   1st Qu.:1104   1st Qu.:1124   1st Qu.: -17.000
##  Median :  -2.00   Median :1535   Median :1556   Median :  -5.000
##  Mean   :  12.64   Mean   :1502   Mean   :1536   Mean   :   6.895
##  3rd Qu.:  11.00   3rd Qu.:1940   3rd Qu.:1945   3rd Qu.:  14.000
##  Max.   :1301.00   Max.   :2400   Max.   :2359   Max.   :1272.000
##  NA's   :8255      NA's   :8713                  NA's   :9430
##    carrier              flight       tailnum             origin
##  Length:336776      Min.   :   1   Length:336776      Length:336776
##  Class :character   1st Qu.: 553   Class :character   Class :character
##  Mode  :character   Median :1496   Mode  :character   Mode  :character
##                     Mean   :1972
##                     3rd Qu.:3465
##                     Max.   :8500
##
##      dest              air_time        distance         hour
##  Length:336776      Min.   : 20.0   Min.   :  17   Min.   : 1.00
##  Class :character   1st Qu.: 82.0   1st Qu.: 502   1st Qu.: 9.00
##  Mode  :character   Median :129.0   Median : 872   Median :13.00
##                     Mean   :150.7   Mean   :1040   Mean   :13.18
##                     3rd Qu.:192.0   3rd Qu.:1389   3rd Qu.:17.00
##                     Max.   :695.0   Max.   :4983   Max.   :23.00
##                     NA's   :9430
##      minute        time_hour
##  Min.   : 0.00   Min.   :2013-01-01 05:00:00
##  1st Qu.: 8.00   1st Qu.:2013-04-04 13:00:00
##  Median :29.00   Median :2013-07-03 10:00:00
##  Mean   :26.23   Mean   :2013-07-03 05:02:36
##  3rd Qu.:44.00   3rd Qu.:2013-10-01 07:00:00
##  Max.   :59.00   Max.   :2013-12-31 23:00:00
##
```

The `flights` dataset is about all the flights that departed from New Yord City (i.e. airports JFK, LGA or EWR) in 2013. In particular, the interest lies in the following variables:

- `hour`, `minute`: the hour and minute of the departure
- `arr_delay`: the arrival delay of the incoming plane (in minutes)
- `dest`: the destination

Note that several variables have **missing values**.

(a) Creating new variables

Create a new variable which encodes a given `hour` and `minute` as one decimal number, i.e. time in hours; that is, for example, 2 hours 45 minutes should be coded to 2.75 hours, because 45 minutes is 45 minutes * (1 hour / 60 minutes) = 0.75 hours. Name this new variable `time` in `flights`.

(b) Plotting the destinations.

(b-1) Calculate the average value of the arrival delay (`arr_delay`) and the number of departing flights (`n`) for each destination (`dest`) and name the resulting data frame `delay.per.dest`.

Tips: group observations by `dest`; use the summary function `n()` to find the count for each group; when calculating the mean, you should be aware of the existence of the missing values.

(b-2) The `airports` dataset contains the coordinates (`lon`, `lat`) of the 1,458 airports. `faa` stands for FAA airport code.

```
airports
```

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

Merge the tibble `delay.per.dest` and `airports` in order to add the coordinates (`lon`, `lat`) of the airports to `delay.per.dest` using `left_join()` in `dplyr` (type `?left_join` to see how it works).

(b-3) Once you have `delay.per.dest` ready, run the plotting function below to create a scatter plot of the latitude against the longitude and scale the points according to the number of departing planes.

```
plot(lat ~ lon, data = delay.per.dest, pch = 19, cex = n / 6000)
```

Explain what values are represented by the size of the bubbles.

[End of Task 6]