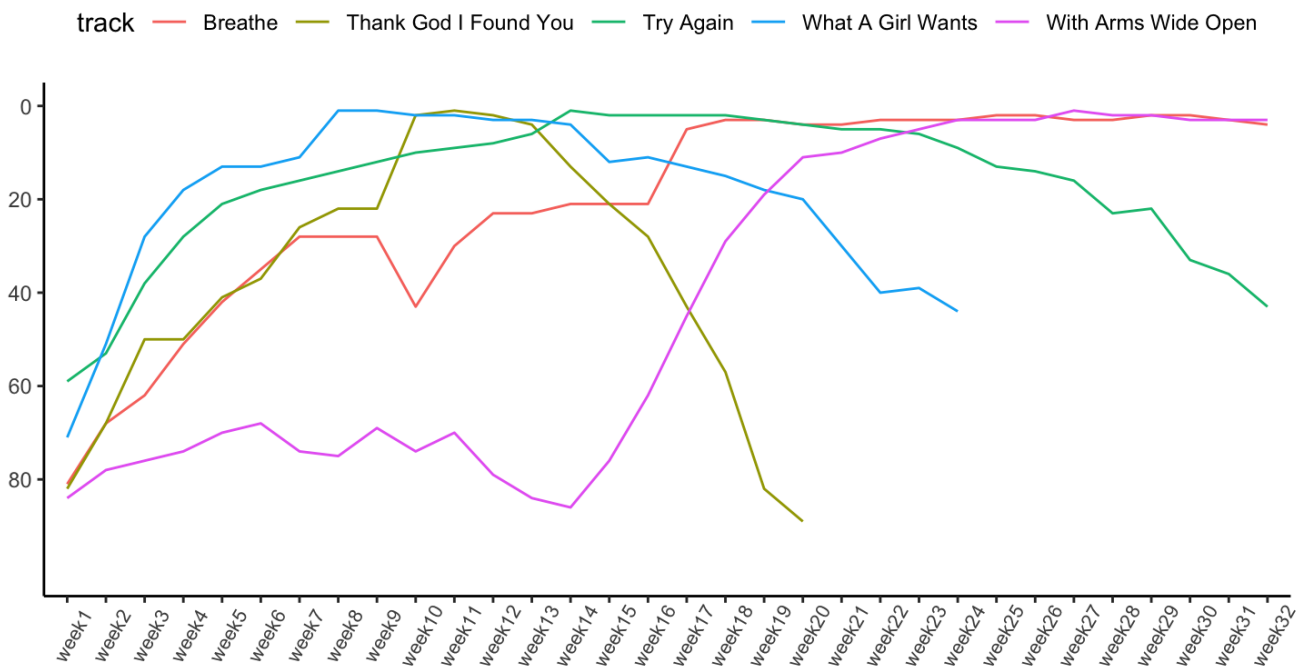
Topic 6: Data Wrangling with R: A Tidy Data Approach

ISOM3390: Business Programming in R

Motivation: Billboard Top Hits

artist	track	time	genre	date.entered	date.peaked	week1	week2	week3	...	week64
Destiny's Child	Independent Women Part I	3:38	Rock	2000/9/23	2000/11/18	78	63	49		na
Santana	Maria, Maria	4:18	Rock	2000/2/12	2000/4/8	15	8	6		na
Savage Garden	I Knew I Loved You	4:07	Rock	1999/10/23	2000/1/29	71	48	43		na
Madonna	Music	3:45	Rock	2000/8/12	2000/9/16	41	23	18		na
Aguilera, Christina	Come On Over Baby (All I Want Is You)	3:38	Rock	2000/8/5	2000/10/14	57	47	45		na
Janet	Doesn't Really Matter	4:17	Rock	2000/6/17	2000/8/26	59	52	43	...	na
Destiny's Child	Say My Name	4:31	Rock	1999/12/25	2000/3/18	83	83	44		na
Iglesias, Enrique	Be With You	3:36	Latin	2000/4/1	2000/6/24	63	45	34		na
Sisqo	Incomplete	3:52	Rock	2000/6/24	2000/8/12	77	66	61		na
Lonestar	Amazed	4:25	Country	1999/6/5	2000/3/4	81	54	44		50

Weekly Track Rankings



Looking at Data in Different Ways

The same underlying data can be represented in multiple ways. But they are not equally easy to use.

artist	track	week1	week2	week3	...	week64
Santana	Maria, Maria	15	8	6	...	na
Lonestar	Amazed	81	54	44		50

artist	track	week	position
Santana	Maria, Maria	week1	15
Lonestar	Amazed	week1	81
Santana	Maria, Maria	week2	8
Lonestar	Amazed	week2	54
Santana	Maria, Maria	week3	6
Lonestar	Amazed	week3	44
	...		
Santana	Maria, Maria	week64	na
Lonestar	Amazed	week64	50

Common Causes of Messiness

- Column headers are values, not variable names
- Multiple variables stored in one column

country	year	m014	m1524	m2534	m3544	m4554	m5564	m65	mu	f014
AD	2000	0	0	1	0	0	0	0	—	—
AE	2000	2	4	4	6	5	12	10	—	3
AF	2000	52	228	183	149	129	94	80	—	93
AG	2000	0	0	0	0	0	0	1	—	1
AL	2000	2	19	21	14	24	19	16	—	3
AM	2000	2	152	130	131	63	26	21	—	1
AN	2000	0	0	1	2	0	0	0	—	0
AO	2000	186	999	1003	912	482	312	194	—	247
AR	2000	97	278	594	402	419	368	330	—	121
AS	2000	—	—	—	—	1	1	—	—	—

- Variables in both rows and columns

country	year	m014	m1524	m2534	m3544	m4554	m5564	m65	mu	f014
AD	2000	0	0	1	0	0	0	0	—	—
AE	2000	2	4	4	6	5	12	10	—	3
AF	2000	52	228	183	149	129	94	80	—	93
AG	2000	0	0	0	0	0	0	1	—	1
AL	2000	2	19	21	14	24	19	16	—	3
AM	2000	2	152	130	131	63	26	21	—	1
AN	2000	0	0	1	2	0	0	0	—	0
AO	2000	186	999	1003	912	482	312	194	—	247
AR	2000	97	278	594	402	419	368	330	—	121
AS	2000	—	—	—	—	1	1	—	—	—

- Multiple types in one table
  - E.g., the Billboard dataset contains two types of observational units: the song and its rank in each week
- One type in multiple tables

Tidy Data

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

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**:

1. Each variable forms a column.
2. Each observation forms a row.
3. Each type of observational unit forms a table.

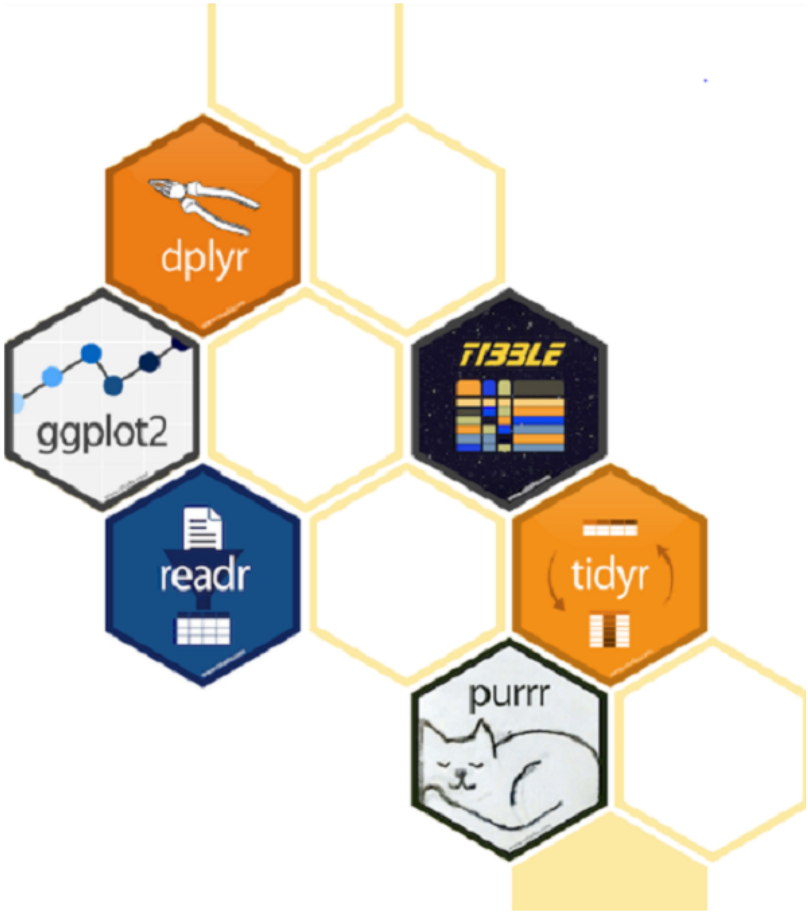
Why Ensure Tidy Data?

Two main advantages:

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

`tidyverse`: R Packages for Data Science



All packages in the `tidyverse` share an underlying design philosophy, grammar, and data structures, and are designed to work together naturally.

Install the complete tidyverse with:

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

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

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

Tibbles

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

Tibbles are a modern re-imagining of the `data.frame` and encapsulates best practices for data frames.

A new tibble can be created from column vectors with `tibble()`:

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

```
## # 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
```

```
# It only recycle inputs of length 1 It evaluates its arguments sequentially
# so that we can refer to variables just created, i.e.  x and y
```

```
glimpse(tb)  # Get a glimpse of the data
```

```
## Observations: 5
## Variables: 3
## $ x <int> 1, 2, 3, 4, 5
## $ y <dbl> 1, 1, 1, 1, 1
## $ z <dbl> 2, 5, 10, 17, 26
```

How Do Tibbles Differ from Data Frames?

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

```
# Regular data frames can be coerced to a tibble with as_tibble().
(billboard_tbl <- as_tibble(billboard))
```

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

- It never changes an input's type. No more `stringsAsFactors = FALSE`!
- It never changes the names of variables.

```
names(data.frame(`crazy name` = 1))
```

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

```
names(tibble(`crazy name` = 1))
```

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

- It doesn't allow setting row names.

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

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

```
billboard[1:3, "week1"]
```

```
## [1] 78 15 71
```

```
billboard[1:3, "week1", drop = FALSE]
```

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

- It clearly distinguish between `[` and `[[`: `[` always returns another tibble. No more `drop = FALSE`!

```
billboard_tbl[1:3, "week1"]
```

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

- ...

To work with some older functions that do not support tibbles, use `as.data.frame()` to turn a tibble back to a data frame.

Revisting the Two Representations

artist	track	week1	week2	week3	...	week64
Santana	Maria, Maria	15	8	6		na
Lonestar	Amazed	81	54	44	...	50

artist	track	week	position
Santana	Maria, Maria	week1	15
Lonestar	Amazed	week1	81
Santana	Maria, Maria	week2	8
Lonestar	Amazed	week2	54
Santana	Maria, Maria	week3	6
Lonestar	Amazed	week3	44
	...		
Santana	Maria, Maria	week64	na
Lonestar	Amazed	week64	50

> *“ One variable is spread across multiple columns. ”*

`tidyr` for Data Tidying

`tidyr` provides a set of functions that help us get to tidy data.

- Two fundamental verbs of data tidying:
  - `gather()`: takes multiple columns and gathers them into key-value pairs: it makes "wide" data longer.
  - `spread()`: takes two columns (key & value) and spreads in to multiple columns, it makes "long" data wider.
- `separate()` and `extract()` (later) pull apart a column that represents multiple variables.
  - `unite()`, the complement to `separate()`

Wide to Long: Gathering

```
billboard_tbl_l <- gather(billboard_tbl, week, position, "week1":"week3")
```

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

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

`gather()` in `tidyr`

```
gather(billboard_tbl, week, position, "week1":"week3")
```

`gather()` pushes data that is currently in columns into rows:

```
str(gather)
```

```
## function (data, key = "key", value = "value", ..., na.rm = FALSE,
##     convert = FALSE, factor_key = FALSE)
```

- `key`: the name of the new "naming" variable
- `value`: the column name representing variable values
- `na.rm`: whether missing values to be removed
- `...`: a selection of columns.
  - If empty, all variables are selected.
  - Select all variables between `x` and `z` with `x:z` (inclusive), exclude `y` with `-y`.

Long to Wide: Spreading

```
spread(billboard_tbl_l, week, position)
```

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

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

`spread()` in `tidyr`

```
spread(billboard_tbl_l, week, position)
```

`spread()`, which is the opposite of `gather()`, pulls the values into their own columns:

```
str(spread)
```

```
## function (data, key, value, fill = NA, convert = FALSE, drop = TRUE,
##     sep = NULL)
```

- `key`: the column we want to split apart
- `value`: the column we want to use to populate the new columns
- `fill`: what to substitute if there are combinations that don't exist
- `sep`: whether to name the new columns in format of "<key_name><sep><key_value>"

Separating Different Variables into Different Columns

```
billboard_tbl_ls <- separate(billboard_tbl_l, date.entered, into = c("year",
    "month", "date"))
```

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

`separate()` in `tidyr`

```
separate(billboard_tbl_l, date.entered, into = c("year", "month", "day"))
```

`separate()` pulls apart one column into multiple columns, by splitting wherever a separator appears.

```
str(separate)
```

```
## function (data, col, into, sep = "[^[:alnum:]]+", remove = TRUE, convert = FALSE,
##     extra = "warn", fill = "warn", ...)
```

- `col`: the column to be split apart
- `into`: names of new variables to create as character vector
- `sep`: separator between columns
  - specified either by a **regular expression** (more on **regular expressions** later) or a vector of character positions
  - defaults to a non-alphanumeric character

When `sep` is a character vector:

- `extra`: what happens when there are pieces more than the number of new variables; one value from `"warn"`, `"drop"`, and `"merge"`
- `fill`: what happens when the resulting pieces are fewer than the number of new variables; one value from `"warn"`, `"left"`, and `"right"`.

```
tibble(x = c("a", "a b", "a b c")) %>% separate(x, c("a", "b"), extra = "merge",
    fill = "left")
```

```
## # A tibble: 3 x 2
##   a     b
##   <chr> <chr>
## 1 <NA>  a
## 2 a     b
## 3 a     b c
```

Combining Multiple Columns into One Variable

```
unite(billboard_tbl_ls, date.entered, day, month, year, sep = "/")
```

```
## # 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 With Arms Wide Open  13/5/2000    week1       84
##  3 Try Again            18/3/2000    week1       59
##  4 Thank God I Found You 11/12/1999  week1       82
##  5 Breathe              6/11/1999    week1       81
##  6 What A Girl Wants    27/11/1999   week2       51
##  7 With Arms Wide Open  13/5/2000    week2       78
##  8 Try Again            18/3/2000    week2       53
##  9 Thank God I Found You 11/12/1999  week2       68
## 10 Breathe              6/11/1999    week2       68
## 11 What A Girl Wants    27/11/1999   week3       28
## 12 With Arms Wide Open  13/5/2000    week3       76
## 13 Try Again            18/3/2000    week3       38
## 14 Thank God I Found You 11/12/1999  week3       50
## 15 Breathe              6/11/1999    week3       62
```

`unite()` in `tidyr`

```
unite(billboard_tbl_ls, date.entered, day, month, year, sep = "/")
```

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

```
str(unite)
```

```
## function (data, col, ..., sep = "_", remove = TRUE)
```

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

In One Step with the Pipe Operator `%>%`

We can use the `magrittr` package's pipe operator (`%>%`) to do this all in one step:

```
gather(billboard_tbl, week, position, "week1":"week3") %>% separate(date.entered,
    into = c("year", "month", "day"))
```

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

The Pipe Operator %>%


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

It can be used to couple several function calls to express a **sequence of multiple operations** clearly :

```
read.csv("/path/to/data/file.csv") %>% gather(week, position, "week1":"week3") %>%
    separate(date.entered, into = c("year", "month", "day")) %>% subset(week ==
    "week1") %>% .$position %>% mean
```

```
## [1] 75.4
```

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 the `tidyverse` and loaded automatically when using packages in it.

The Argument Placeholder

Placeholder says 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))
```

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))
```

`dplyr` for Data Manipulation

`dplyr` introduces basic verbs that help streamline the data manipulation process.

- Single table verbs:
  - `select(data, variables)`: pick variables based on their names.
  - `filter(data, conditions)`: pick obervations based on conditions.
  - `arrange(data, variables)`: reorder cases 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 cases by variables.
- A variety of two-table verbs: e.g., `inner_join(data1, data2, variables)`.

It also has several other functions such as `slice()`, `rename()`, `transmute()`, `sample_n()` and `sample_frac()`, all of which we may find useful.

Selecting & Reordering Columns with `select()`

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 2000/9/23
##  2 Maria, Maria                            15     8     6 2000/2/12
##  3 I Knew I Loved You                      71    48    43 1999/10/23
##  4 Music                                   41    23    18 2000/8/12
##  5 Come On Over Baby (All I Want Is You)   57    47    45 2000/8/5
##  6 Doesn't Really Matter                   59    52    43 2000/6/17
##  7 Say My Name                             83    83    44 1999/12/25
##  8 Be With You                             63    45    34 2000/4/1
##  9 Incomplete                              77    66    61 2000/6/24
## 10 Amazed                                  81    54    44 1999/6/5
## # ... with 307 more rows
```

Alternatively, we can use a helper function:

```
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 2000/9/23
##  2 Maria, Maria                            15     8     6 2000/2/12
##  3 I Knew I Loved You                      71    48    43 1999/10/23
##  4 Music                                   41    23    18 2000/8/12
##  5 Come On Over Baby (All I Want Is You)   57    47    45 2000/8/5
##  6 Doesn't Really Matter                   59    52    43 2000/6/17
##  7 Say My Name                             83    83    44 1999/12/25
##  8 Be With You                             63    45    34 2000/4/1
##  9 Incomplete                              77    66    61 2000/6/24
## 10 Amazed                                  81    54    44 1999/6/5
## # ... with 307 more rows
```

Select Helpers

A variety of helper functions can be used as part of a select call:

- `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".
- `matches("(.)\\1")`: selects variables that match a **regular expression**. This one matches any variables that contain repeated characters.
- `num_range("x", 1:3)` matches x1, x2 and x3.
- `everything()`: all variables.

```
billboard %>% select(track, date = date.entered, everything())
```

- `last_col()`: last variable, possibly with an offset, e.g., `last_col(offset = 2)`.

Subsetting Rows with `filter()`

Look at the weekly rankings of songs of 10 artists when they rank higher than the average ranking of all songs:

```
billboard_1 %>% filter(artist %in% billboard[["artist"]][1:10] & position <
    mean(position))
```

```
## # A tibble: 428 x 8
##    artist    track     time  genre date.entered date.peaked week  position
##    <chr>     <chr>     <chr> <chr> <chr>        <chr>       <chr>    <int>
##  1 Santana   Maria, M~ 4:18  Rock  2000/2/12    2000/4/8    week1       15
##  2 Madonna   Music     3:45  Rock  2000/8/12    2000/9/16   week1       41
##  3 Aguilera~ I Turn T~ 4:00  Rock  2000/4/15    2000/7/1    week1       50
##  4 Madonna   American~ 4:30  Rock  2000/2/19    2000/3/4    week1       43
##  5 Santana   Maria, M~ 4:18  Rock  2000/2/12    2000/4/8    week2        8
##  6 Savage G~ I Knew I~ 4:07  Rock  1999/10/23   2000/1/29   week2       48
##  7 Madonna   Music     3:45  Rock  2000/8/12    2000/9/16   week2       23
##  8 Aguilera~ Come On ~ 3:38  Rock  2000/8/5     2000/10/14  week2       47
##  9 Iglesias~ Be With ~ 3:36  Latin 2000/4/1     2000/6/24   week2       45
## 10 Aguilera~ What A G~ 3:18  Rock  1999/11/27   2000/1/15   week2       51
## # ... with 418 more rows
```

Comparision functions: `==`, `>`, `>=`, etc.;

Logical operators: `&` (or multiple comma-separated varialbes), `|`, `!`, `xor()`;

Other functions returning logical values: `is.na()`; `between()`, `near()` (in `dplyr`); etc.

Rearranging Rows with `arrange()`

Sort weekly ranking positions first by `artist`, second by `tack`, and last by `position`:

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

```
## # A tibble: 5,306 x 4
##    artist  track                                          week  position
##    <chr>   <chr>                                          <chr>    <int>
##  1 2 Pac   Baby Don't Cry (Keep Ya Head Up II)             week1       87
##  2 2 Pac   Baby Don't Cry (Keep Ya Head Up II)             week2       82
##  3 2 Pac   Baby Don't Cry (Keep Ya Head Up II)             week3       72
##  4 2 Pac   Baby Don't Cry (Keep Ya Head Up II)             week4       77
##  5 2 Pac   Baby Don't Cry (Keep Ya Head Up II)             week5       87
##  6 2 Pac   Baby Don't Cry (Keep Ya Head Up II)             week6       94
##  7 2 Pac   Baby Don't Cry (Keep Ya Head Up II)             week7       99
##  8 2Ge+her The Hardest Part Of Breaking Up (Is Getting Bac~ week1       91
##  9 2Ge+her The Hardest Part Of Breaking Up (Is Getting Bac~ week2       87
## 10 2Ge+her The Hardest Part Of Breaking Up (Is Getting Bac~ week3       92
## # ... with 5,296 more rows
```

The `desc()` function sorts values of the specified variables in descending order.

Adding New Columns with `mutate()`

Calculate changes in ranking position between two adjacent weeks:

```
billboard_l %>% subset(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
```

Useful functions: `+`, `-`, etc.; `log()`; **window functions** such as `lead()`, `lag()`, `min_rank()`, `cumsum()`; `na_if()`, `coalesce()`, `if_else()`, `recode()`, `case_when()` (in `dplyr`).

A Syntactic Shortcut: `transmute()`

We can directly use `transmute()`, if all we want to keep from `mutate()` are the newly formed variables:

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

```
## # A tibble: 33 x 4
##    current previous change  week
##      <int>    <int>  <int> <int>
##  1      71       NA     NA     1
##  2      48       71     23     2
##  3      43       48      5     3
##  4      31       43     12     4
##  5      20       31     11     5
##  6      13       20      7     6
##  7       7       13      6     7
##  8       6        7      1     8
##  9       4        6      2     9
## 10       4        4      0    10
## # ... with 23 more rows
```

Grouping Data with `group_by()`

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

```
billboard_1 %>% 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~ Independe~ 3:38  Rock  2000/9/23    2000/11/18  week1       78
##  2 Santana  Maria, Ma~ 4:18  Rock  2000/2/12    2000/4/8    week1       15
##  3 Savage ~ I Knew I ~ 4:07  Rock  1999/10/23   2000/1/29   week1       71
##  4 Madonna  Music      3:45  Rock  2000/8/12    2000/9/16   week1       41
##  5 Aguiler~ Come On O~ 3:38  Rock  2000/8/5     2000/10/14  week1       57
##  6 Janet    Doesn't R~ 4:17  Rock  2000/6/17    2000/8/26   week1       59
##  7 Destiny~ Say My Na~ 4:31  Rock  1999/12/25   2000/3/18   week1       83
##  8 Iglesia~ Be With Y~ 3:36  Latin 2000/4/1     2000/6/24   week1       63
##  9 Sisqo    Incomplete 3:52  Rock  2000/6/24    2000/8/12   week1       77
## 10 Lonestar Amazed     4:25  Coun~ 1999/6/5     2000/3/4    week1       81
## # ... with 5,296 more rows
```

```
billboard_1 %>% group_by(week, artist) %>%
  groups # returns a list of symbols
```

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

```
billboard_1 %>% group_by(week, artist) %>%
  group_vars # returns a character vector
```

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

By default, `group_by()` overrides existing grouping

```
billboard_1 %>% group_by(week, artist) %>%
  group_by(track) %>% group_vars
```

```
## [1] "track"
```

Use `add = TRUE` to instead append

```
billboard_1 %>% group_by(week, artist) %>%
  group_by(track, add = TRUE) %>% group_vars
```

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

Use `ungroup()` to removing grouping:

```
billboard_1 %>% group_by(week, artist) %>% ungroup %>% group_vars
```

```
## character(0)
```

Split-Apply-Combine 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_1 %>% group_by(track) %>% summarise(`weeks on chart` = n(), `average position` = mean(position),
    `peak position` = min(position))
```

```
## # A tibble: 316 x 4
##    track                `weeks on chart` `average positio~ `peak position`
##    <chr>                           <int>             <dbl>           <dbl>
##  1 (Hot S**t) Country ~               34              30.9               7
##  2 3 Little Words                      9              94.4              89
##  3 911                                19              60                38
##  4 A Country Boy Can S~                3              86.7              75
##  5 A Little Gasoline                   6              89.8              75
##  6 A Puro Dolor (Pures~               26              49.5              26
##  7 Aaron's Party (Come~               15              66.3              35
##  8 Absolutely (Story O~               27              26.5               6
##  9 All Good?                           3              97.3              96
## 10 All The Small Things               23              33.3               6
## # ... with 306 more rows
```

Summarizing Data with `summarise()`

Calculate every artist's best weekly ranking:

```
billboard_1 %>% group_by(week, artist) %>% summarise(`peak position` = min(position))
```

```
## # A tibble: 3,988 x 3
## # Groups:   week [?]
##    week  artist              `peak position`
##    <chr> <chr>                         <dbl>
##  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_1 %>% group_by(week, artist) %>% summarise(`peak position` = min(position)) %>%
    group_vars()
```

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

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

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_1 %>% 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 Drama          Left, Right, Left         week1      100
##  3 Herndon, Ty    No Mercy                  week1      100
##  4 Lil' Mo        Ta Da                     week1      100
##  5 Larrieux, Amel Get Up                    week1      100
##  6 Jay-Z          Hey Papi                  week2      100
##  7 Hart, Beth     L.A. Song                 week2      100
##  8 Diffie, Joe    The Quittin' Kind         week2      100
##  9 Juvenile       U Understand              week3      100
## 10 Diffie, Joe    The Quittin' Kind         week3      100
## # ... with 5,296 more rows
```

Unless we specifically ask:

```
billboard_1 %>% 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(position = min_rank(position))  # mean(position)
```

```
## # A tibble: 5,306 x 3
## # Groups:   track [316]
##    track                                week  position
##    <chr>                                <chr>    <int>
##  1 Independent Women Part I             week1       28
##  2 Maria, Maria                         week1       19
##  3 I Knew I Loved You                   week1       33
##  4 Music                                week1       23
##  5 Come On Over Baby (All I Want Is You) week1       21
##  6 Doesn't Really Matter                week1       24
##  7 Say My Name                          week1       31
##  8 Be With You                          week1       20
##  9 Incomplete                           week1       26
## 10 Amazed                               week1       55
## # ... with 5,296 more rows
```

```
billboard_l %>% group_by(track) %>% .[c("track", "week", "position")] %>% filter(position <
    lag(position))  # mean(position)
```

```
## # A tibble: 2,120 x 3
## # Groups:   track [298]
##    track                                week  position
##    <chr>                                <chr>    <int>
##  1 Independent Women Part I             week2       63
##  2 Maria, Maria                         week2        8
##  3 I Knew I Loved You                   week2       48
##  4 Music                                week2       23
##  5 Come On Over Baby (All I Want Is You) week2       47
##  6 Doesn't Really Matter                week2       52
##  7 Be With You                          week2       45
##  8 Incomplete                           week2       66
##  9 Amazed                               week2       54
## 10 It's Gonna Be Me                     week2       70
## # ... with 2,110 more rows
```

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()` allows us to compute differences and trends.
- Cumulative aggregates: `cumsum()`, `cummin()`, `cummax()` from base R; `cumall()`, `cumany()`, and `cummean()` from `dplyr`.

Scoped Variants of `dplyr` Verbs

`dplyr` verbs have variants suffixed with `_all`, `_if`, and `_at`.

- `verb_all(tbl, ...)`: apply an operation on all variables.
- `verb_at(tbl, vars, ...)`: apply an operation on a subset of variables specified with the quoting function `vars()`.
- `verb_at(tbl, predicate, ...)`: apply an operation on the subset of variables for which a `predicate` function returns `TRUE`.

```
billboard %>% group_by(artist) %>% .[1:9] %>% summarise_at(vars(matches("week")),
    funs(mean, median), na.rm = TRUE)
```

```
## # A tibble: 228 x 7
##    artist week1_mean week2_mean week3_mean week1_median week2_median
##    <chr>       <dbl>      <dbl>      <dbl>        <dbl>        <dbl>
##  1 2 Pac        87         82         72           87           82
##  2 2Ge+h~       91         87         92           91           87
##  3 3 Doo~       78.5       73         70           78.5         73
##  4 504 B~       57         34         25           57           34
##  5 98?          51         39         34           51           39
##  6 A*Tee~       97         97         96           97           97
##  7 Aaliy~       71.5       57.5       44.5         71.5         57.5
##  8 Adams~       76         76         74           76           76
##  9 Adkin~       84         84         75           84           84
## 10 Aguil~       59.3       45.7       34.3         57           47
## # ... with 218 more rows, and 1 more variable: week3_median <dbl>
```

```
billboard %>% .[1:9] %>% filter_if(is.integer, any_vars(. < 10))
```

```
## # A tibble: 2 x 9
##   artist  track     time  genre date.entered date.peaked week1 week2 week3
##   <chr>   <chr>     <chr> <chr> <chr>        <chr>       <int> <int> <int>
## 1 Santana Maria, M~ 4:18  Rock  2000/2/12    2000/4/8       15     8     6
## 2 Kenny G Auld Lan~ 7:50  Jazz  1999/12/25   2000/1/8       89    89     7
```

Quoting Functions

- `vars()`
  - used for scoped summarising and mutating verbs (`mutate_at()` and `summarise_at()`).
  - accepts helpers like `starts_with()`, e.g., `vars(start_with("week")`.
  - Instea of a `var()` selection, an integerish vector of column positions or a character vector of column names (e.g., `c("artist", "track")`) can also be used.
- `funs()`
  - generates a named list of functions for input to other functions, e.g., `summarise_at()`.
  - Functions can be specified in many ways, e.g., `funs(m1 = mean, m2 = "mean", m3 = mean(., na.rm = TRUE))`, `funs(inches = . / 2.54)`
- `all_vars()` and `any_vars()`
  - instruct scoped filtering verbs (e.g. filter_if() or filter_all()) to apply a predicate expression to all relevant variables. E.g., `filter_at(billboard, vars(starts_with("week")), any_vars(. < 10))`

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

```
rep(1:5, times = c(2, 3, 2, 5, 2)) %>% sample
```

```
##  [1] 4 2 4 2 5 1 1 4 2 3 4 3 5 4
```

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

```
## [1] 5
```