tidyverse

Matt McDonald

inspired by
https://github.com/michaellevy/tidyverse_talk/blob/master/tidyverse.md

What is the tidyverse?

- R packages for data science
- The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Tidy data

Put data in *data frames*

- Each type of observation gets a data frame
- Each variable gets a column
- Each observation gets a row

Tidy APIs

Functions should be consistent and easily (human) readable

- Take one step at a time
- Connect simple steps with the pipe
- Referential transparency

Okay but really, what is it?

- Suite of ~20 packages that provide consistent, user-friendly, smart-default tools to do most of what most people do in R.
- **https://www.tidyverse.org/packages/**
- `install.packages(tidyverse)` installs all of the above packages.
- `library(tidyverse)` attaches only the core packages.

Why tidyverse?

- Consistency
  - e.g. All `stringr` functions take string first
  - e.g. Many functions take data.frame first -> piping
    - Faster to write
    - Easier to read
  - Tidy data: Imposes good practices
  - Type specificity
- Implements simple solutions to common problems (e.g. `purrr::transpose`)
- Smarter defaults
  - e.g. `utils::write.csv(row.names = FALSE)` = `readr::write_csv()`
- Runs fast (thanks to `Rcpp`)

- Interfaces well with other tools (e.g. Spark with `dplyr` via `sparklyr`)

Data Types in R

- **Numeric**: This is the default data type for numbers in R. It includes real numbers (floating-point values) and integers. For example, 42, 3.14.
- **Integer**: Specifically for integer values. While numeric data can include integers, if you specifically want to declare an integer, you append L to the number, like 42L.
- **Logical**: This type represents boolean values and can either be TRUE or FALSE.
- **Character**: This type represents strings. Text and characters are enclosed in quotes. For example, “Hello, World!”.
- **Factor**: A data type used for categorical data. Factors can be ordered or unordered and are very useful in statistical modeling and graphics. They are stored as integers but each integer value corresponds to a label.
- Other: Complex, Raw and Date.

Data Structures in R

- **Vectors**: An ordered collection of elements of the same basic data type.
- **Matrices**: Two-dimensional, rectangular layouts of elements of the same basic data type.
- **Arrays**: Similar to matrices but can have more than two dimensions.
- **Data frames**: A table or a two-dimensional array-like structure where each column can contain different types of data (numeric, character, factor, etc.). It's one of the most important data types in R for data analysis.
- **Lists**: An ordered collection of objects (components). A list in R can contain objects of different types including numbers, strings, vectors, and even other lists.

tibble

Tibbles are a modern re-imagining of data frames.

Tibbles print politely.

```
tdf = tibble(x = 1:1e4, y = rnorm(1e4))
tdf
```

```
# A tibble: 10,000 × 2
       x       y
   <int>   <dbl>
 1     1  0.165
 2     2  0.666
 3     3  0.671
 4     4 -0.675
 5     5  0.271
 6     6  0.759
 7     7 -0.0394
 8     8  1.10
 9     9  0.960
10    10  0.717
# ℹ 9,990 more rows
```

- Can customize print methods with `print(tdf,`

```
n = rows, width =
cols)
```

- Set default with

```
options(tibble.print_max
= rows, tibble.width
= cols)
```

Tibble defaults

Tibbles have some convenient and consistent defaults that are different from base R data.frames.

type consistency

```
dfs = list(
  df = data.frame(abc = letters[1:3], xyz = letters[24:2
  tbl = tibble(abc = letters[1:3], xyz = letters[24:26])
)

sapply(dfs, function(d) class(d[, "abc"]))
```

```
$df
[1] "character"

$tbl
[1] "tbl_df"     "tbl"        "data.frame"
```

Note that tidyverse import functions (e.g. `readr::read_csv`) default to tibbles and that *this can break existing code*.

List-columns!

```
a <- tibble(ints = 1:5,
            powers = lapply(1:5, function(x) x^(1:x)))

a[[5,2]]
```

```
[[1]]
[1]    5   25  125  625 3125
```

```
a
```

```
# A tibble: 5 × 2
   ints powers
  <int> <list>
1     1 <dbl [1]>
2     2 <dbl [2]>
3     3 <dbl [3]>
4     4 <dbl [4]>
5     5 <dbl [5]>
```

The pipe %>%

Sends the output of the LHS function to the first argument of the RHS function.

```
1:8 %>%
  sum() %>%
  sqrt()
```

```
[1] 6
```

```
sqrt(sum(1:8))
```

```
[1] 6
```

`dplyr`

Common data(frame) manipulation tasks.
Four core “verbs”:

- filter
- select
- arrange
- group_by + summarize

We will cover in more detail in further classes

joins

`dplyr` also does multi-table joins and can connect to various types of databases.

```
    t1 = tibble(alpha = c(letters[2:7],letters[11]), num = c(2:7,11))
    t2 = tibble(alpha = letters[4:10], num = 4:10)
    t3 <- full_join(t1, t2, by = "alpha", suffix = c("_t1", "_t2"))

    arrange(t3, alpha)
```

```
# A tibble: 10 × 3
   alpha num_t1 num_t2
   <chr>  <dbl>  <int>
 1 b          2     NA
 2 c          3     NA
 3 d          4      4
 4 e          5      5
 5 f          6      6
 6 g          7      7
 7 h         NA      8
 8 i         NA      9
 9 j         NA     10
10 k         11     NA
```

Other Useful Packages

- *ggplot2* - data visualization
- *tidyr* - data reshaping
- *stringr* - manipulating strings
- *purrr* - programming + working with lists

What does “un-tidy” data mean?

```
who <- read_csv(here::here('Lecture1', 'who.csv'))

who
```

```
# A tibble: 9,137 × 46
   country  iso2  iso3   year new_sp_m014 new_sp_m1524 new_sp_m2534 new_sp_m3544
   <chr>    <chr> <chr> <dbl>       <dbl>         <dbl>         <dbl>         <dbl>
 1 Afghani… AF    AFG    1980          NA            NA            NA            NA
 2 Afghani… AF    AFG    1981          NA            NA            NA            NA
 3 Afghani… AF    AFG    1982          NA            NA            NA            NA
 4 Afghani… AF    AFG    1983          NA            NA            NA            NA
 5 Afghani… AF    AFG    1984          NA            NA            NA            NA
 6 Afghani… AF    AFG    1985          NA            NA            NA            NA
 7 Afghani… AF    AFG    1986          NA            NA            NA            NA
 8 Afghani… AF    AFG    1987          NA            NA            NA            NA
 9 Afghani… AF    AFG    1988          NA            NA            NA            NA
10 Afghani… AF    AFG    1989          NA            NA            NA            NA
# ℹ 9,127 more rows
# ℹ 38 more variables: new_sp_m4554 <dbl>, new_sp_m5564 <dbl>,
#   new_sp_m65 <dbl>, new_sp_f014 <dbl>, new_sp_f1524 <dbl>,
#   new_sp_f2534 <dbl>, new_sp_f3544 <dbl>, new_sp_f4554 <dbl>,
#   new_sp_f5564 <dbl>, new_sp_f65 <dbl>, new_sn_m014 <dbl>,
#   new_sn_m1524 <dbl>, new_sn_m2534 <dbl>, new_sn_m3544 <dbl>,
#   new_sn_m4554 <dbl>, new_sn_m5564 <dbl>, new_sn_m65 <dbl>, …
```

Using tidyverse to work with “un-tidy” data

```
who %>%
  select(-iso2, -iso3) %>%
  gather(group, cases, -country, -year ) %>%
  mutate(group = str_replace(group, "new_*", ""),
         method = str_extract(group, "[a-z]+"),
         gender = str_sub(str_extract(group, "_[a-z]"), 2, 2),
         age = str_extract(group, "[0-9]+"),
         age = ifelse(str_length(age) > 2,
                      str_c(str_sub(age, 1, -3), str_sub(age, -2, -1), sep = "-")
                      str_c(age, "+"))) %>%
  group_by(year, gender, age, method) %>%
  summarize(total_cases = sum(cases, na.rm = TRUE), .groups='drop') %>%
  ggplot(aes(x = year, y = total_cases, linetype = gender)) +
  geom_line() +
  facet_grid(method ~ age,
             labeller = labeller(.rows = label_both, .cols = label_both)) +
  scale_y_log10() +
  theme_light() +
  theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1))
```

Using tidyverse to work with “un-tidy” data