Lecture 7: Strings, Factors, and Dates

UNIVERSITY OF
SAN FRANCISCO

Abbie M. Popa

BSDS 100 - Intro to Data Science with R

Questions: Tibbles versus Tribbles

- We make data structures using **constructors**, which are special functions that return a data structure.

Constructor	Output
`c( )`	vector
`matrix( )`	matrix
`array( )`	array
`list( )`	list
`data.frame()`	(generic) data frame
`tibble( )`	tibble
`tribble( )`	tribble

- Note: Both `tibble` and `tribble` construct a tibble, they are two different ways to make the same thing!

Tibbles versus Dataframes

- Tibbles are a friendly "species" of data frame, like a golden retriever is a species of dog
- In general, use whichever you prefer (unless specifically stated in the question)
- Exception! If your employer or interviewer requires you to only use "base" `R` you will have to use generic data frames.

Outline

- Factors
- Strings
- Dates

Part I: Factors

Factors

- Factors are less human-intuitive than strings, but useful for computing
- A factor is a vector of elements from a discrete set, and is used to store categorical (ordinal or nominal) data
- Discrete: the values do not overlap (e.g., you can be a freshman or a sophomore but not a freshmore)
- Categorical: Values like real numbers are continuous, values that fall into bins are categorical (though you can make a continuous value categorical by binning, e.g., "even" vs "odd" numbers)
  - Ordinal categories have intrinsic order (e.g., small dogs, medium dogs, big dogs) though there may not be equal distance between
  - Nominal categories have no intrinsic order (e.g., apple, microsoft, linux)

Factors: Under the Hood

- Factors are built on top of integer vectors using two attributes:
  1. The `class()` 'factor': makes them behave differently from regular integer vectors
  2. The `levels()`: define the discrete set of permissible values
  3. We also have a useful argument `labels()` that we can use to give more human readable names to factors

Nominal vs Ordinal Factors

- Although we (intelligent humans) have an inherent ability to understand the ordering of the ordinal categories below, `R` does not, and unless told, will treat them as nominal categorical variables
- Nominal (unordered) factors are sorted automatically by `R`, e.g., alphabetically, numerically, etc.
- **Note:** The terms *ordered* and *sorted* are **not** synonymous here

Factor Comparisons

- R can "sort" nominal factors (e.g., alphabetically), so what does it mean that they are un-ordered?
- Operations like $<$ will fail on nominal factors
- These will work on ordinal (ordered) factors
- If we omit the "levels" field R will choose it's own ordering

Example: Try This

Use the following code to create the variable `myCyl` using the dataset `mtcars`

```
myCyl <- mtcars$cyl
```

1. Create an ordered factor from `myCyl`, mapping the levels to 'Small', 'Medium' and 'Large'
2. How many observations have cylinders <= 'Medium'?

A Caution with factors

Be careful when adding items to factors, as items that don't belong to the original set of levels may cause errors

Part II: Strings

Strings

- We have already encountered strings, which are character objects we build using ""
- You can make strings with single quotes `'this is a string'` or double quotes `"this is a string"`
- If you forget to close your quotes, `R` thinks you are still writing a string and will give you the `+` continuation character
- You can put almost anything in a string, for special characters you will need to use the backslash
- When importing data, you will frequently see "\n" for new line and "\t" for tab.

String Manipulation

- There are more manipulations you can do to strings with **regular expressions**, but you can go far with a few basic functions
- The library `stringr` provides an intuitive package for working with strings
- The function `str_length()` will tell you how long a string is
- The function `str_c()` will combine multiple strings into one string
- The function `str_sub()` takes a subset of a string, based on character index
- The function `str_to_lower()` and `str_to_upper` change capitalization

Finding Strings

- The function `str_detect()` allows you to find strings that contain a particular substring
- We can use this to add columns to a data frame

New Day

- Today we will finish strings
- If we have time, we will cover dates (if not, dates on Thursday)
- Introduction of Case Study 1, and some time to form groups and work

Extracting Numbers from Strings

- Another very common act is to extract numbers from a string
- The library `readr` provides the function `parse_number()` for this purpose

Splitting Strings

- The function `str_split` can be used to split strings on some character
- Can be used to split a sentence into words

Finding a letter in a word

- We've seen how `str_detect` can find if a pattern is present, what if we need to know where a pattern is in a word?
- The library `stringi` provides us with two functions for this purpose
- One, `stri_locate` is "lazy" or "eager," it will return the first match it finds
- The second, `stri_locate_all` is "greedy," it will return all matches
- These are useful in conjunction with `str_sub`

White space

- If you are processing text input from users, you may find people use differing amounts of white space
- The function `str_trim` will remove whitespace from the beginning and end of a string
- The function `str_squish` will remove extra white spaces from within a string.

Counting

- You can count how many times something occurs in a string with `str_count`

Data Quality

- How much text manipulation you will need to do will depend on the quality of the data you start with
- If you start with well organized data, you may avoid text manipulation entirely!
- Consistency and planning ahead will save you time

Part III: Dates

dates

- Dates are exceedingly common in datasets (birthdays, date of data collection)
- Unfortunately, inconsistent date formatting (e.g., May 18, 1951, 5-18-51, 1951/05/18) makes dealing with dates a pain
- Fortunately, there is a handy library, `lubridate` that can save us lots of time and trouble

Getting Dates

- See cheatsheet (!) on github for many lubridate functions
- Lubridate provides several functions for entering a date and returning a generic date object, pick the one that follows your day, month, year order
- There is also an option to get today by typing `today()`

Comparing Dates

- You can subtract two dates to find the number of days between them `today() - mdy("7-25-2015")`
- Getting other units is a little more awkward but still easier than without lubridate

```
bday <- mdy("7-25-15")
interval(bday, today())/years(1)
interval(bday, today())/months(1)
```

- There are many more functions of lubridate that you can find if they are useful to you including timezones, look into these if you end up at an international company!

A brief note on time

- There are similar functions should you need to measure "time" (e.g., hours or minutes of battery life)
- Most commonly, `ymd_hms()` can be used if you have a date with a time and `hms()` can be used if you have a time.
- Similar to `today()`, `now()` gets the current time.
- Should you work with times for a final project or a job, look out for issues caused by time zones and daylight savings time!