R Basics and the Tidyverse

Key Differences From Python

Indexing begins at 1.

Assignment is made with the assignment operator:

```
> x <- 7
```

(although x = 7 also works)

Everything is a **vector** (or vector-like).

Lists and Vectors in R

Vector: homogeneous (all entries must be the same datatype).

x <- c(1,2,3)

You can also assign names to the elements of a vector:

x <- c(“first” = 1, “second” = 2, “third” = 3)

You can access element of a vector either by index (remember, indexing starts at 1) or by name.

x[1] or x[‘first’]

Lists and Vectors in R

List: heterogeneous - can have mixed datatypes.

x <- list(1,2,3)

You can also assign names to the elements of a vector:

x <- list(“first” = 1, “second” = 2, “third” = 3)

You can access element of a vector either by index (remember, indexing starts at 1) or by name. When referencing the name, you can use either brackets or $

x[1] or x[‘first’] or x$first

Factors

Factors: used to represent categorical data. Can be ordered or unordered.

```
> x <- factor(c('one', 'one', 'two', 'three', 'three', 'three'))
> x
[1] one   one   two   three three three
Levels: one three two
```

By default, R orders the levels alphabetically, but you can override this (which comes in handy when you're trying to get a barplot to display in the correct order)

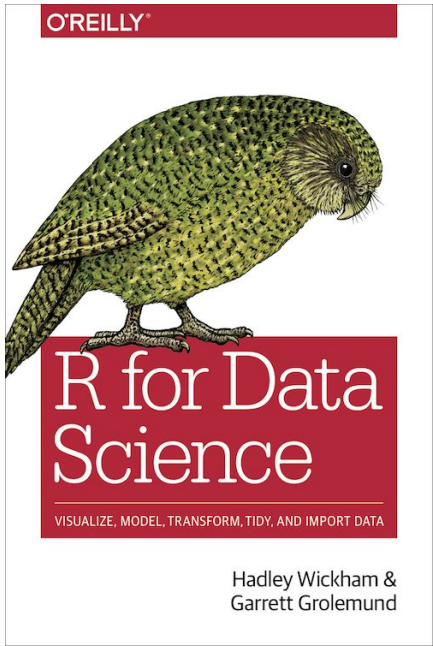
```
> levels(x) <- c('one', 'two', 'three')
> x
[1] one   one   three two   two   two
Levels: one two three
```

What is the Tidyverse?

The tidyverse is an opinionated collection of R packages designed for data science.

All packages share an underlying design philosophy, grammar, and data structures.

Learn more about the tidyverse:
R for Data Science
https://r4ds.had.co.nz/



Tibbles

The basic data structure of the tidyverse is the *tibble*.

A tibble is a slightly improved version of R's data frame.

```
> flights
# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl> <chr>
 1  2013     1     1      517            515         2      830            819        11 UA
 2  2013     1     1      533            529         4      850            830        20 UA
 3  2013     1     1      542            540         2      923            850        33 AA
 4  2013     1     1      544            545        -1     1004           1022       -18 B6
 5  2013     1     1      554            600        -6      812            837       -25 DL
 6  2013     1     1      554            558        -4      740            728        12 UA
 7  2013     1     1      555            600        -5      913            854        19 B6
 8  2013     1     1      557            600        -3      709            723       -14 EV
 9  2013     1     1      557            600        -3      838            846        -8 B6
10  2013     1     1      558            600        -2      753            745         8 AA
# ... with 336,766 more rows, and 9 more variables: flight <int>, tailnum <chr>, origin <chr>,
#   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Tidy Data

Tidy data is data structured for **exploration** and **analysis.**

Three guidelines:

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

References:
http://vita.had.co.nz/papers/tidy-data.pdf
https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html

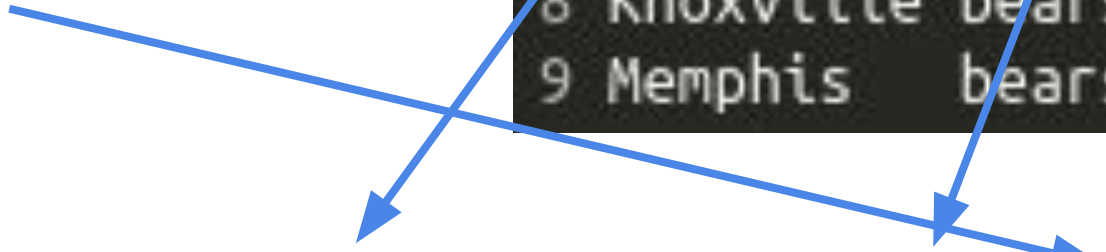
Not Tidy

```
> zoo_df
# A tibble: 3 x 4
  zoo       lions tigers bears
  <chr>     <dbl>  <dbl> <dbl>
1 Nashville     6      4     6
2 Knoxville     4      4     8
3 Memphis       2      4     4
> |
```

Tidy

```
# A tibble: 9 x 3
  zoo       animal count
  <chr>     <chr>  <dbl>
1 Nashville lions      6
2 Knoxville lions      4
3 Memphis   lions      2
4 Nashville tigers     4
5 Knoxville tigers     4
6 Memphis   tigers     4
7 Nashville bears      6
8 Knoxville bears      8
9 Memphis   bears      4
```

Exclude the zoo column



```
zoo_df %>% gather(key = 'animal', value = 'count', -zoo)
```

Pipes

%>%

Ctrl + shift + m
Cmd + shift + m

Pipe operators

Unix pipe: | (1973)

F# pipe: |> (2005)

R pipe: %>% (2014)

- introduced in November 2014
- **magrittr** package – Stefan Bache & Hadley Wickham

The Tidy Tools Manifesto:

1. Reuse existing data structures.
2. **Compose simple functions with the pipe.**
3. Embrace functional programming.
4. **Design for humans.**

https://cran.r-project.org/web/packages/tidyverse/vignettes/manifesto.html

```
my_data <- do_one_last_thing(
        do_something_else(
          do_something(data_in)
            )
          )
```

```
temp_data_1<- do_something(data_in)
temp_data_2 <-do_something_else(temp_data_1)
my_data <- do_one_last_thing(temp_data_2)
```

The pipe operator indicates that **an object is "piped in"** to a function or expression as its first argument:

```
object  %>%  function()
```

R functions are designed (mostly) with data as the first argument - which works nicely with pipes:

```
my_data <- data_in  %>%
        do_something()  %>%
        do_something_else()  %>%
        do_one_last_thing()
```