Day 2

Tidyverse

- Not a single package but has a lot of them
- Works based on piping operator %>% (from package magrittr)
  - There is an alternative in base R - |>
- Idea is to build pipelines

Example code:

```
Data <- data[data$name %in% “lol”, ]

Data <- data %>% filter(name %in% “lol”)

Basically filter(data, expression)
```

```
Data <- filter(...)
Data <- dplyr::filter(...)
```

```
> library(tidyverse)
-- Attaching packages ---------------------------------------- tidyverse 1.2.1 --
v ggplot2 3.2.0     v purrr   0.3.2
v tibble  2.1.3     v dplyr   0.8.1
v tidyr   0.8.3     v stringr 1.4.0
v readr   1.3.1     v forcats 0.4.0
-- Conflicts ------------------------------------------- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
>
```

dplyr

- **A Grammar of Data Manipulation**
  - A fast, consistent tool for working with data frame like objects, both in memory and out of memory.
  - This will be of focus in the tutorial you are going to read

Map function to understand purrr

The map functions transform their input by applying a function to each element of a list or atomic vector and returning an object of the same length as the input.

- map() always returns a list. See the modify() family for versions that return an object of the same type as the input.
- map_int(), map_dbl() and map_chr() return an atomic vector of the indicated type (or die trying). For these functions, .f must return a length-1 vector of the appropriate type.
- map_vec() simplifies to the common type of the output. It works with most types of simple vectors like Date, POSIXct, factors, etc.

purrr enhances R's functional programming (FP) toolkit by providing a complete and consistent set of tools for working with functions and vectors.

split a data frame into pieces, fit a model to each piece, compute the summary, then extract the R^2

```
mtcars |>
  split(mtcars$cyl) |>  # from base R - 3 dataframes
  map(\(df) lm(mpg ~ wt, data = df)) |>  lapply(split_out, lm, mpg ~ wt) -> list(3)
  map(summary) %>%
  map_dbl("r.squared")
```

readr

The goal of 'readr' is to provide a fast and friendly way to read rectangular data (like 'csv', 'tsv', and 'fwf'). It is designed to flexibly parse many types of data found in the wild, while still cleanly failing when data unexpectedly changes.

tidyr

The goal of tidyr is to help you create **tidy data**. Tidy data is data where:

1. Every column is variable.
2. Every row is an observation.
3. Every cell is a single value.

Example of using tidyr

table4a

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

```
pivot_longer(table4a,
cols=2:3, names_to=”year”,
value_to=”cases” )
```

stringr

stringr is built to provide fast, correct implementations of common string manipulations.

stringr focuses on the most important and commonly used string manipulation functions whereas stringi provides a comprehensive set covering almost anything you can imagine

Example of stringr

```
x <- c("why", "video", "cross", "extra", "deal", "authority")
str_length(x)
#> [1] 3 5 5 5 4 9
str_c(x, collapse = ", ")
#> [1] "why, video, cross, extra, deal, authority"
str_sub(x, 1, 2)
#> [1] "wh" "vi" "cr" "ex" "de" "au"
```

forcats

The goal of the **forcats** is to provide a suite of tools that solve common problems with factors

- `fct_reorder()`: Reordering a factor by another variable.
- `fct_infreq()`: Reordering a factor by the frequency of values.
- `fct_relevel()`: Changing the order of a factor by hand.
- `fct_lump()`: Collapsing the least/most frequent values of a factor into “other”.

tibble

- Provides a 'tbl_df' class (the 'tibble') with stricter checking and better formatting than the traditional data frame.
- The tbl_df class is a special case of the base data.frame
- Create a tibble: tibble(), as_tibble(), tribble(), enframe()
- Inspect a tibble: print.tbl(), glimpse()

ggplot2()

- A package to visualize data - but much better than base R plotting
- The base idea is to create layers of plots
- The first layer is to create an empty plot with data and aesthetics in it
- Later we define the type of plot
- Additionally we can add lines, text, statistics etc

Example:

Plot <- ggplot(data, aes(x=x, y=y, color=color)) + geom_box(..., aes(shape=shape))

Note that aes() inputs are not strings but symbols. If you want to convert text you can use rlang::sym(“x”)=x where x is the character input. Rlang is for some other time.

Your tasks

- https://bookdown.org/yih_huynh/Guide-to-R-Book/tidyverse.html - Read chapters 4-7
- http://r-statistics.co/Complete-Ggplot2-Tutorial-Part1-With-R-Code.html - go through ggplot tutorial on this website
- https://github.com/rstudio/cheatsheets - read the cheatsheets and save them for the packages mentioned in the presentation to get a better sense of the packages

Solutions to pattern questions

https://www.javatpoint.com/star-program-in-c#Plus-Star