Claire's Exam 2

Missing Data

Types	Description
MCAR	The probability of missing data for a variable is the same for all cases. Implies that causes of the missing data are unrelated to the data.
MAR	The probability of missing data is related to observed variables but unrelated to unobserved information.
MNAR	The probability of missing data is related to unobserved variables (and probably observed variables too).

- Identifying Missing Data: is.na(), skim(), vis_miss()
- Handling Missing Data: drop_na(), replace_na()

Explicit missing data	Implicit missing data
Data that is explicitly marked as missing (NA)	Data that is missing but not explicitly marked as such.

Use the function complete() to create that new row and plug in values of NA for the missing data.

Functions

```
# Defining a function
function_name <- function(input_name_1, input_name_2 = default_value_2){

  # function body - code that does something

  return(output)
}

# Calling a function (all valid ways)
function_name(input_name_1 = 2, input_name_2 = 4)
function_name(2, 4)
function_name(input_name_2 = 4, input_name_1 = 2)
function_name(2)
```

Example for if-else condition:

```
convert_temp <- function(temp, unit) {
    if (unit=="F") {
        (temp - 32) * 5/9
    } else if (unit=="C") {
        (temp * 9/5) + 32
    }
}

convert_temp(0, unit = "C")
convert_temp(32, unit = "F")
```

Base R

Vectors

```
num_vec <- vector("numeric", length = 2) #empty vector: Zeros
log_vec <- vector("logical", length = 3) #empty vector: FALSE
chr_vec <- vector("character", length = 4) #empty vector: empty strings
named_vec <- c('name1' = 1, 'name2' = 2) # Named numeric vector
```

Lists

```
ex_list <- list(a = 1:3, b = c("a", "b", "c"), c = matrix(1:6, nrow = 2)
```

Array, Matrices, Data Frame

```
ary <- array(NA, dim = c(2,3,4))
m <- matrix(NA, nrow = 2, ncol = 3)
mod_df <- tibble(x = 1:10, y = 1:10 + rnorm(10))
```

Subset with [], and single elements can be subsetted with $.

Loops+Iter

Iteration

```
# seq_len() iterates over the rows of a data frame, seq_along() would it
for (i in seq_along(groups)) {
    print(groups[i])
}

for (i in seq_len(length(groups))) {
    print(groups[i])
}
```

Memory Allocation (Purrr, map)

```
# Example 1: No storage container; append new value to existing vector
x <- integer()
bench::mark(
  for (i in 1:1e5) {
    x <- c(x, i)
  }
)

# Example 2: Using a storage container!
#Notice median (time to run), mem_alloc (how much memory you are using),
x <- vector('integer', length = 1e5)
bench::mark(
  for (i in 1:1e5) {
    x[i] <- i
  }
)

# Example 3: Vectorized function
#Notice median (time to run), mem_alloc (how much memory you are using),
bench::mark(
  x <- seq(1, 1e5, by = 1)
)

# Example 4: purrr::map()
results <- bench::mark(purrr::map_int(1:1e5, ~.x))
```

- map() returns a list
- map_chr() returns a character vector
- map_lgl() returns a logical vector
- map_int() returns an integer vector
- map_dbl() returns a numeric vector
- map_vec() returns a vector of a different (non-atomic) type (like dates)

map2 or pmap

For iterating across rows of a data frame!

```
# save randomly generated data in tibble with simulation parameters
sim_data <- args |>
  mutate(rand_data = pmap(args, rnorm))
# specify variations in some arguments but leave some arguments constant
pmap_chr(string_data, str_replace_all, string = "ppp nnn hhh")
```

walk

No need to save an output for each iteration...

```
# Iterate over paths and plots to save the export images
walk2(
  by_clarity$path,
  by_clarity$plot,
  function(path, plot){ ggsave(path, plot, width = 6, height = 6)}
)
```

APIs

URLs

Base URL	Scheme	Host Name	File Path	Query String
https://api.census.gov	http://	api.census.gov	data/2019/acs/acs1	?get = NAME, B02015_009E, B02015_009M &for=state:*

Wrapper Packages

```
tidycensus::get_acs(
    year = 2020,
    state = "MN",
    geography = "tract",
    variables = c("B01003_001", "B19013_001"),
    output = "wide",
    geometry = TRUE
)
```

Example

```
req <- request("https://boardgamegeek.com/xmlapi2") |>
    req_url_path_append("search") |>
    req_url_query(query = I("mystery+curse"), type = I("boardgame,boardg
```

Then, req_perform(), to see Content-Type that the format of the response is something called XML. We can navigate to the request URL to see the structure of this output. Use resp_body_xml() to read in the XML as an R object. * xml_find_all(), xml_attr() * resp_body_json(simplifyVector = TRUE)

Web Scraping

robots.txt find what can be scraped

```
nih <- read_html("https://www.nih.gov/news-events/news-releases")
# Retrieve and inspect article titles
article_titles <- nih |>
    html_elements(".thumbnail-teaser__heading") |>
    html_text()
```

Database and SQL

1. DBI::dbConnect(duckdb::duckdb())
2. dbWriteTable(con, "flights", nycflights13::flights)
3. flights <- tbl(con, "flights")

with *dplyr*

- show_query()

Composing Query

- SELECT tells SQL that a query is coming.
- TOP(100) only returns the first 100 rows.
- Id, Title, Score, Body, Tags determines what columns are included in the result
- FROM Posts determines the source dataset.
- ORDER BY ??? DESC syntax is similar to R's arrange()
- GROUP BY
- JOIN keyword must go in between the two tables we want to join. Each table must be named.

Effective and Interactive Visualization

- Motivation & Context
- Orientation (axes, color legend, etc.)
- Highlights (one or two interesting data points)
- Big Picture (overall takeaways)

Plotly and Shiny (Ui and Server, App)

Code Quality Checklist

- [ ] Readability
- [ ] Meaningful Names
- [ ] Testing / Test Cases
- [ ] Efficiency
- [ ] Reproducibility
- [ ] Simple Tasks in Functions

Data Quality Checklist

- [ ] Data Parsing (lubridate, stringr)
- [ ] Data Joining (correct joins ie. list_rbind(), list_cbind())