R Reference Manual

Contents

Chapter 1

Introduction

In *R for Data Science*, Garrett Grolemund and Hadley Wickham outline the tools needed to tackle about 80% of the tasks required in a typical data science project ("Introduction"). Those tools, illustrated in the diagram below and the organizing template for this reference manual, are:

- Import
- Tidy
- Transform
- Visualize
- Model
- Communicate
- Program

1.1 References & Resources

- For an introduction to the R programming language, see the R Project for Statistical Computing's "What is R?" and Wikipedia's "R (programming language)."
- To download R, go to r-project.org and choose the cloud CRAN Mirror option.
- To program in the R language on a user-friendly platform, download the RStudio IDE.
- The R Project for Statistical Computing
  - Library of R Packages
  - *Getting Help with R*

  - *The R Manuals*
  - *Frequently Asked Questions*
  - *Books Related to R*
  - *Documentation*

- RStudio

  - *RStudio Cheat Sheets*
  - *Webinars and Videos On Demand*
  - *Online learning*
  - *RStudio Blog*

- Online Manuals

  - *R for Data Science*
  - *Advanced R* by Hadley Wickham + Hadley's second edition draft is available here.
  - *R Packages* by Hadley Wickham
  - *The R Inferno* by Patrick Burns
  - *The tidyverse style guide*
  - *Efficient R Programming*
  - *Free R Reading Material*
  - *What They Forgot to Teach You About R* by Jenny Bryan and Jim Hester

- Other Online Resources

  - DataCamp
  - RDocumentation
  - R Bloggers
    * "Tutorials for learning R"
  - Regular Expressions 101

Chapter 2

Import or Create Data

2.1 Create Data

`{base}`

- `array()`
- `c()`
  - See `base::vector()`.
- `data.frame()`
- `dir.create()`
- `factor()`
- `list()`
- `matrix()`
- `seq()`
- `vector()`
  - Preferable to `base::c()` when creating an empty vector ("Five Tips to Improve Your R Code").

`{stats}`

- `dnorm()`
- `pnorm()`
- `qnorm()`
- `rnorm()`

`{tibble}`

- `add_row()`
- `tibble()`
- `tribble()`

2.2 Local Drive

`{base}`

- `attach()`
  - Allows objects in the database to be accessed by giving their names (e.g., `height` rather than `women$height`).
- `file.choose()`
- `file.size()`
- `load()`
  - Reload datasets saved with `base::save()`.
- `readRDS()`
  - Restore an R object written with `base::saveRDS()`.

`{data.table}`

- `fread()`
  - Similar to `utils::read.table()`, but faster and more convenient for large data sets.

`{foreign}`

- `read.spss()`

`{haven}`

- `read_sas()`

`{readr}`

- `read_csv()`
- `read_csv2()`
- `read_delim()`
  - Remove imported attributes using `%>% .[]` or `attr(df, "spec") <- NULL`.
- `read_tsv()`

`{readxl}`

- `excel_sheets()`
- `read_excel()`
- `read_xls()`
- `read_xlsx()`

`{utils}`

- `data()`
  - Load specified data sets, or list the available data sets.
  - Use this function to load the data sets that accompany R packages, such as `openintro::hsb2`, `openintro::email50`,and `gapminder::gapminder`.
- `read.csv()`
- `read.csv2()`
- `read.delim()`
- `read.delim2()`
- `read.table()`

`{XLConnect}`

- `readWorksheetFromFile()`

2.3 Internet

`{httr}`

- `GET()`
  - Get a URL.

`{jsonlite}`

- `read_json()`

`{readr}`

- `read_csv()`
- `read_csv2()`
- `read_delim()`
- `read_tsv()`

`{rjson}`

- `fromJSON()`
  - Convert JSON to R.

`{utils}`

- `download.file()`
  - See example below.
- `unzip()`
  - See example below.

2.3.1 Examples

`utils::download.file()`:

```
download.file(
  "https://assets.datacamp.com/production/repositories/5028/datasets/a55843f83746968c7
  destfile = paste0(getwd(), "/Snake River Visits.rds")
)

# Option 1:
snake_river_visits <- readRDS(file.choose())

# Option 2:
snake_river_path   <- paste0(getwd(), "/Snake River Visits.rds")
snake_river_visits <- readRDS(snake_river_path)
```

`utils::download.file()` for .Rdata files:

```
# Example 1:
download.file(
  "https://assets.datacamp.com/production/repositories/236/datasets/7f714f993f1ad4c3d2
  destfile = "datacamp_iris_dataset.Rdata"
)

load("datacamp_iris_dataset.Rdata")

# Example 2:
download.file(
  "https://assets.datacamp.com/production/repositories/235/datasets/3b6fc2923b5990585
```

```
  destfile = "chis_2009.Rdata",
  # The documentation for `download.file` indicates that the function will
  # automatically include `mode = "wb"` for .Rdata files. That may have happened
  # in Example 1, but didn't happen in Example 2, which is why I've included it.
  mode = "wb"
)

load("chis_2009.Rdata")
```

`utils::unzip()`:

```
download.file(
  "https://assets.datacamp.com/production/repositories/1069/datasets/578834f5908e3b2fa575429a2875
  destfile = "Data Sets/Countries",
  mode = "wb"
)

unzip("Data Sets/Countries", exdir = "Data Sets")
```

2.4 Database

`{DBI}`

- `dbAppendTable()`
- `dbBind()`
- `dbClearResult()`
- `dbConnect()`
- `dbCreateTable()`
- `dbDataType()`
- `dbDisconnect()`
- `dbExecute()`
- `dbFetch()`
- `dbGetQuery()`
- `dbListTables()`
  - Specify schema with `dbListTables(conn, schema = "schema_name")`.
- `dbReadTable()`
- `dbRemoveTable()`
- `dbSendQuery()`
- `dbSendStatement()`
- `Id()`
  - See example below.

- `SQL()`
  - See example below.

`{dbplyr}`

- `in_schema()`
- `tbl()`

`{odbc}`

- `dbConnect()`
  - See example below for specifying a database.
  - See example below for troubleshooting hang-ups.

2.4.1 Examples

`DBI::Id()` vs `DBI::SQL()`:

The following are comparable methods of accessing `<database>.<schema_name>.<table_name>`:

```
dbCreateTable(
  conn,
  name =
    Id(catalog = "database_name", schema = "schema_name", table = "table_name"),
  fields = sample_data_frame
)

dbCreateTable(
  conn,
  name = SQL("database_name.schema_name.table_name"),
  fields = sample_data_frame
)
```

`odbc::dbConnect()`:

```
# Specify database:
conn_1 <- dbConnect(odbc::odbc(), dsn = "sql") # Default database: Actuary

conn_2 <- dbConnect(odbc::odbc(), dsn = "sql", Database = "Staging")

# Troubleshoot hang-ups
# It appears that RStudio's attempt to load information into the Connection Pane,
# via `odbc::dbConnect`, can sometimes cause the call to hang, indefinitely.
```

```
# Use the following code to access the database in such a situation:

options(connectionObserver = FALSE)
```

2.4.2 References

- "Databases using R"

2.4.3 SQL and R

2.4.3.1 Tips and Troubleshooting Hints

- Temp Tables
  - Use `SET NOCOUNT ON` in the SQL query to prevent R confusing the SQL rowcount return value with the temporary table.
- Try case-sensitive syntax (e.g., when querying a database other than the default).
- Try selecting specific columns, rather than `SELECT *`, as SQL BLOB columns cause trouble.
- Error Messages
  - `Error in env[[name]] <- x : attempt to use zero-length variable name`
    - Problem code: `select max(date) from table`
    - Solution: `select max(date) as max_date from table`
  - Error: "Object does not exist or you do not have permission to create/delete it" in combination with having db_owner rights to the database.
    - Problem code: `CREATE OR ALTER PROCEDURE <non_default_schema>.<procedure_name>`
    - Solution: The default database in the ODBC is not set to the database you wish to modify. You can recreate the error in SSMS by using the same code (e.g. set the database in the drop-down menu to master and run the script, which outputs the same error). Update the ODBC connection to specify the correct database.
- Parameterized Queries
  - Reference: "Databases Using R"
  - See example below.
- Optimization:
  - `WHERE EXISTS` and `WHERE NOT EXISTS` are generally more efficient than `=` and `<>`.
  - `=` is generally more efficient than `<>`.

2.4.3.2 Examples

Parameterized query:

```
param_query <- dbSendQuery(con, "select * from table where column = ?")

dbBind(param_query, input)
dbFetch(param_query)

# As a function:
query_data <- function(input) {

  dbBind(param_query, input)
  dbFetch(param_query)
}

map(as.list(df$input_column), query_data) %>% bind_rows()
```

2.4.3.3 SQL References

- "10 Easy Steps to a Complete Understanding of SQL"
- "Query Planning"
- "Using SQL in RStudio by Irene Steves

2.5 Notes & References

- "Data structures" (Wickham, *Advanced R*)
- `{openintro}` contains data sets useful for practicing and teaching.
- "Vectors" (Grolemund & Wickham, *R for Data Science*)

Chapter 3

Tidy

"Tidying your data means storing it in a consistent form that matches the semantics of the dataset with the way it is stored. In brief, when your data is tidy, each column is a variable and each row is an observation. Tidying data is important because the consistent structure lets you focus your struggle on questions about the data, not fighting to get the data into the right form for different functions."
- Garrett Grolemund & Hadley Wickham, *R for Data Science*

3.1 Explore Raw Data

`{base}`

- `attr()`
  - `attr(x, "names")` is the same as `base::names(x)`.
  - Use `attr(x, "names") <- value` to set attribute values.
- `attribute()`
- `class()`
- `colnames()`
- `dim()`
- `dimnames()`
- `format()`
- `length()`
- `levels()`
  - Levels default to alphabetical order, so be careful when renaming them (i.e., don't accidentally set the "F" level equal to "Male" rather than "Female").

- `mode()`
- `names()`
- `nchar()`
- `order()`
- `rownames()`
- `summary()`
- `typeof()`

`{graphics}`

- `hist()`
- `plot()`

`{skimr}`

- `skim()`
  - Similar to `tibble::glimpse()` and `utils::str()`.

`{tibble}`

- `glimpse()`
  - Similar to `skimr::skim()` and `utils::str()`.

`{utils}`

- `head()`
- `str()`
  - Similar to `skimr::skim()` and `tibble::glimpse()`.
- `tail()`

3.2 Tidy Data

`{base}`

- `droplevels()`
  - Drop unused levels from factors.
  - This function will keep levels that have even 1 or 2 counts. If you want to remove levels with low counts from a data set in order to simplify your analysis, first filter out those rows and then use `droplevels()`.

- `duplicated()`
  - See also `data.table::duplicated()`.
- `unique()`
  - See also `data.table::unique()`.

`{data.table}`

- `anyDuplicated()`
- `duplicated()`
  - See also `base::duplicated()`.
- `unique()`
  - See also `base::unique()`.

`{dplyr}`

- `na_if()`
  - See also `tidyr::replace_na()`.

`{janitor}`

- `clean_names()`
- `get_dupes()`
  - Prefer over `base::duplicated()` and `data.table::duplicated()`.
- `remove_empty()`

`{naniar}`

- `replace_with_na()`
  - Compliment to `tidyr::replace_na()`; consider also `stringr::str_replace(replacement = NA_character_)`.

`{purrr}`

- `transpose()`
  - Turn a list-of-lists inside-out.

`{reshape2}` (superseded by `{tidyr}`)

- `cast()`
  - See `tidyr::pivot_wider()`
- `melt()`
  - See `tidyr::pivot_longer()`

`{splitstackshape}`

- `cSplit()`
  - See also `tidyr::separate()`

`{tibble}`

- `rowid_to_column()`
  - Add a column of sequential row IDs.
  - Useful when a separate ID is required to manipulate rows or columns, such as when using `tidyr::pivot_longe()r`
- `rownames_to_column()`

`{tidyr}`

- `nest()`
  - Nest repeated values in a list-variable.
  - Helpful when separating a data frame in preparation to model the data for each grouping.
- `replace_na()`
  - See also `dplyr::na_if()`.
- `separate()`
- `pivot_longer()`
- `pivot_wider()`
- `unite()`
- `unnest()`

3.3 Prepare Data for Analysis

3.3.1 Coerce

`{base}`

- `as.array()`
- `as.data.frame()`
  - See also`tibble::as_tibble()`.
- `as.Date()`
- `as.factor()`
- `as.list()`
- `as.matrix()`
- `as.POSIX-()`
- `factor()`
  - Rearrange the order of factors by using the `levels` argument. For example, rearrange the order of "Bad,"Good," and "Neutral" using 'levels = c("Bad", "Neutral", "Good").
- `is.na()`
  - Use `is.na <-` to set elements to `NA`.
- `unclass()`

`{methods}`

- `as()`

`{tibble}`

- `as_tibble()`
  - Preferable to `base::as.data.frame()`.
- `enframe()`
  - Preferable to using `tibble::as_tibble()` to coerce a vector to a data frame.

3.3.2 Dates and Datetimes

`{anytime}`

- `anytime()`: Parse POSIXct or Date objects from input data.

`{base}`

- `as.Date()`
- `OlsonNames()`

  - Displays available time zones.

- `as.POSIXct.*()`
- `as.POSIXlt.*()`
- `strptime()`
  - Date-time conversion to and from character.
- `Sys.timezone()`
  - See also `base::OlsonNames()`.

`{fasttime}`

- `fastPOSIXct()`
  - Convert strings into `POSIXct` object (string must be in year, month, day, hour, minute, second format.)

`{hms}`

- `as.hms()`
- `hms()`
  - Store time-of-day values as `hms` class.
- `is.hms()`

`{lubridate}`

- `as_date()`
- `fast_strptime()`
  - Fast C parser of numeric formats only that accepts explicit format arguments (just as `base::strptime()`).
  - Note that the format argument must match the input exactly, including any non-white space characters (such as "T" and "Z").
- `make_date()`
  - Create dates from numeric representations.
- `make_datetime()`
  - Create date-times from numeric representations.
- `parse_date_time()`
  - This function can be slow, because it is designed to be forgiving and flexible. If the dates you are working with are in a consistent format (ideally ISO 8601), use one of the following: `fasttime::fastPOSIXct()` or `lubridate::parse_date_time2()`.

- `parse_date_time2()`
  - Fast C parser of numeric orders.
- `ymd()`: Parse dates with year, month, and day components.
  - Related formats: `ydm()`, `mdy()`, `myd()`, `dmy()`, `dym()`, `yq()`.
- `ymd_hms()`: Parse date-times with year, month, day, hour, minute, and second components.
  - Related formats: `ymd_hm()`, `ymd_h()`, `dmy_hms()`, `dmy_hm()`, `dmy_h()`, `mdy_hms()`, `mdy_hm()`, `mdy_h()`, `ydm_hms()`, `ydm_hm()`, `ydm_h()`.

3.3.3 Filter

`{dplyr}`

- `filter()`

`{purrr}`

- `keep()`

`{stats}`

- `na.omit()`

3.3.4 Strings

`{base}`

- `agrep()`
- `cat()`
  - Concatenate and print.
- `chartr()`
  - Change certain characters.
- `gregexpr()`
- `grep()`
- `grepl()`
- `gsub()`
- `regexec()`

- regexpr()
- sub()
- tolower()
  - See also stringr::str_to_lower().
- toupper(): Convert to uppercase.
  - See also stringr::str_to_upper().

{fuzzyjoin}

- stringdist_join()
- stringdist_anti_join()
- stringdist_full_join()
- stringdist_inner_join()
- stringdist_left_join()
- stringdist_right_join()
- stringdist_semi_join()

{fuzzywuzzyR}

- FuzzMatcher()

{qdap}

- check_spelling()

{qdapDictionaries}

- DICTIONARY()
  - Nettalk Corpus syllable data set.
- GradyAugmented()
  - Augmented list of Grady Ward's *English Words* and Mark Kantrowitz's *Names* List.
  - Mark Kantrowitz's *Names* list is available in full here.

{stringr}

- str_detect
  - Control the pattern argument options with regex() (e.g., str_detect(x, regex(pattern, ignore_case = TRUE)).
- str_remove()
- str_to_lower()
- str_to_title()
- str_trim()
- str_to_upper()

3.3.5 Test

`{base}`

- `all()`
- `any()`
  - Use `any(is.na(data.frame))` to determine if there are any `NA` values in a data frame.
- `anyNA()`
  - Possibly faster implementation of `base::any(is.na(x))`.
- `exists()`
- `is.array()`
- `is.data.frame()`
- `is.matrix()`
- `is.vector()`
- `setequal()`
  - Check two vectors for equality.
- `sum()`
  - Test whether all elements of a vector do or do not meet a certain condition, use as follows: `sum(email$num_char < 0)`.

`{purrr}`

- `every()`

`{stats}`

- `complete.cases()`
  - Find complete cases (i.e., rows without `NA` values).

`{tibble}`

- `is_tibble()`

Chapter 4

Transform

"Transformation includes narrowing in on observations of interest (like all people in one city, or all data from the last year), creating new variables that are functions of existing variables (like computing velocity from speed and time), and calculating a set of summary statistics (like counts or means)."
- Garrett Grolemund & Hadley Wickham, *R for Data Science*

4.1 Arithmetic & Summary Statistics

`{assertive}`

- `is_divisible_by()`

`{base}`

- `abs()`
- `colMeans()`
- `colSums()`
- `diff()`
- `IQR()`
- `max()`
- `mean()`
  - Use `base::mean(variable == value)` to calculate percentage match (accuracy); see example below.
- `median()`
- `min()`
- Mode: Use `table` to view the mode of a data set.

- Operators:
  - Arithmetic Operators: `+`, `-`, `*`, `/`, `%%`
  - Comparison Operators: `<`, `>`, `<=`, `>=`, `==`, `!=`.
    - Use `identical()` and `(all.equal))` rather than `==` and `!=` in tests where a single `TRUE` or `FALSE` is required (such as `if` expressions).
- `range()`
  - Use `diff(range())` to get the range as a measure of variability.
- `round()`
- `rowMeans()`
- `rowSums()`
- `scale()`
- `sd()`
- `signif()`
  - Round values to a specified number of significant digits.
- `sqrt()`
- `sum()`
- `summary()`
- `var()`

`{dplyr}`

- `count()`
- `group_by()`
- `n()`
  - Must be used within `summarize()`, `mutate()`, or `filter()`.
- `n_distinct()`
- `near()`
  - Compare two numeric vectors (safer than using `==`).
- `summarize()`
  - See `dplyr::n_distinct()`.
- `tally()`
- `top_n()`

`{dummies}`

- `dummy.data.frame()`
  - Dummify a data frame (useful when needing to calculate the Jaccard index for categorical data)

`{magrittr}`

- `extract()`
  - `x %>% extract(y)` is equivalent to `x[y]`.
- `extract2()`
- `multiply_by()`
  - `x %>% multiply_by(y)` is equivalent to `x * y`.
- `raise_to_power()`
  - `x %>% raise_to_power(y)` is equivalent to `x^y`.

`{stats}`

- `aggregate()`
  - Compute summary statistics of data subsets.
- `cor()`
- `cov()`
- `cutree()`
  - Pair with `stats::hclust()`.
- `dist()`
- `hclust()`
- `lag()`
- `rnorm()`
- `var()`

4.1.1 Examples

`base::mean()`

```
by_country <-
  votes %>%
  group_by(country) %>%
  summarize(
    total = n(),
    percent_yes = mean(vote == 1))

# Classification Trees: Compute the accuracy on the test dataset:
mean(loans_test$outcome == loans_test$prediction)
```

4.2 Create New Variables or Modify Existing Ones

`countrycode`

- `countrycode()`: Convert country codes into country names.

`dplyr`

- `mutate()`: Add new variables.
  - `mutate()` can also be used to modify existing variables. To change the case of a character variable, for example, do something like:
  - Child function: `transmute()` (drops existing variables).
- `recode()`: Recode values (the numeric alternative to using `if_else` or `case_when()`).

Example: `dplyr::mutate`

```
df <-
  df %>%
  mutate(var_name = str_to_lower(var_name))

# Add a T/F vector, which can then be referenced to highlight certain
# information in a plot:

df <-
  df %>%
  mutate(hilight = attributed_provider_id = 95597)
```

4.3 Dates and Datetimes

- `base::date`
  - Get the current system date and time.
- `base::difftime`
  - Time intervals and differences.
  - `base::difftime` is the function behind the `-` operator when used with dates and datetimes (e.g., `time_1 - time_2` is equivalent to `difftime(time_1, time_2)`). The advantage of using `difftime` over `-`, however, is the `units` argument because it allows you to specify the unit of time in which the difference is calculated.

- `base::months`
  - Extract the month names.
- `base::quarters`
  - Extract the calendar quarters.
- `base::seq.Date`
- `base::Sys.Date`
  - Get the current date in the current time zone.
- `base::Sys.time`
  - Get the absolute date-time value (which can be converted to various time zones and may return different days).
- `base::weekdays`
  - Extract weekday names.
- `lubridate::ceiling_date`
- `lubridate::date`
  - Get or set the date component of a date-time.
- `lubridate::day`
  - Get or set the day component of a datetime.
- `lubridate::floor_date`
- `lubridate::month`
  - Get or set the month component of a datetime.
- `lubridate::now`
  - The current time (as a POSIXct object).
- `lubridate::quarter`
  - Get or set the fiscal quarter or semester component of a datetime.
- `lubridate::round_date`
- Time spans: Duration
  - Use when you are interested in seconds elapsed.
  - `lubridate::ddays`
  - `lubridate::dhours`
  - `lubridate::dminutes`
  - `lubridate::dseconds`
  - `lubridate::dweeks`
  - `lubridate::dyears`
- Time spans: Interval
  - Use when you have a start and end.

  - `%--%`
  - `int_aligns`
  - `int_diff`
  - `int_end`
  - `int_flip`
  - `int_length`
  - `int_overlaps`
  - `int_shift`
  - `int_standardize`
  - `int_start`
  - `interval`
  - `is.interval`

- Time spans: Period

  - Use when you are interested in human units.
  - `lubridate::day`
  - `lubridate::hour`
  - `lubridate::minute`
  - `lubridate::month`
  - `lubridate::second`
  - `lubridate::week`
  - `lubridate::year`

- Time zones:

  - `lubridate::force_tz`
    - Change the time zone without changing the clock time.
  - `lubridate::tz`
    - Extract the time zone from a datetime.
  - `lubridate::with_tz`
    - View the same instant in a different time zone.

- `lubridate::today`
- `lubridate::%m+%` & `%m-%`

  - Add and subtract months to a date without exceeding the last day of the new month.

- `lubridate::%within%`

  - Test whether a date or interval falls within an interval.

- `lubridate::year`

  - Get or set the year component of a datetime.

4.4 Factors

`forcats`

- `fct_drop()`: Drop levels.
- `fct_reorder()`: Reorder levels, based on the value of another variable.
- `fct_rev()`: Reverse levels.

`stats`

- `reorder()`: Reorder levels of a factor.
  - Useful within the `aes()` argument in a `ggplot()` call.

4.5 Merge or Append Data

`base`

- `append()`: Add elements to a vector.
- `cbind()`: Combine objects by column.
- `intersect()`: Combine data shared in common between two datasets.
  - Similar to `dplyr::semi_join()`.
- `merge()`: Merge two data frames.
  - `dplyr::join` functions are an alternative to `merge()`.
- `rbind()`: Combine objects by row.
- `setdiff()`: Find the difference between two vectors.
  - Similar to `dplyr::anti_join()`.
- `union()`: Combine two datasets without duplicating values.

`dplyr`

- `bind()`: Bind multiple data frames by row and column.
  - Child functions: bind_rows(), bind_cols(), combine().
- Join Functions: Join two tables.
  - Filtering Joins:
    - anti_join(): Return all rows from `x` where there are not matching values in `y`, keeping just columns from `x`.

    - semi_join(): Return all rows from `x` where there are matching values in `y`, keeping just columns from `x`. A semi join differs from an inner join because an inner join will return one row of `x` for each matching row of `y`, where a semi join will never duplicate rows of `x`.
    - Mutating Joins:
        - full_join(): Return all rows and all columns from both `x` and `y`. Where there are not matching values, returns `NA` for the one missing.
        - inner_join(): Return all rows from `x` where there are matching values in `y`, and all columns from `x` and `y`. If there are multiple matches between `x` and `y`, all combination of the matches are returned.
        - left_join(): Return all rows from `x`, and all columns from `x` and `y`. Rows in `x` with no match in `y` will have `NA` values in the new columns. If there are multiple matches between `x` and `y`, all combinations of the matches are returned.
        - right_join(): Return all rows from `y`, and all columns from `x` and `y`. Rows in `x` with no match in `y` will have `NA` values in the new columns. If there are multiple matches between `y` and `x`, all combinations of the matches are returned.

`tibble`

- `add_column()`: Add columns to a data frame.
- `add_row()`: Add rows to a data frame.

4.6 Narrow in on Observations of Interest

`{base}`

- `order()`
    - See also: `dplyr::arrange()`.
- `prop.table()`
    - Express table entries as proportions of the marginal table (thus, as the values are proportions of the whole, `sum(prop.table(table_name))` = 1.)
    - The input is a table produced by `base::table()`.
    - Specify conditional proportions on rows or columns by using the `margin` argument.
- `table()`: Build a table of the counts at each combination of factor levels.

  - Use `base::prop.table()` to see the table entries expressed as proportions.

`{dplyr}`

- `arrange()`
- `distinct()`
- `filter()`
  - When using multiple `|` conditions, use `%in%` to save space; see example below.
- `rename()`
- `sample_n()`
- `select()`
  - Use `?dplyr::select_helpers` to see the available helper functions.
- `slice()`
- `transmute()`
  - A combination of `dplyr::select()` and `dplyr::mutate()`.

4.6.1 Examples

`dplyr::filter()`

```
ilo_data %>%
  filter(country %in% c("Sweden", "Switzerland"))

ilo_data %>%
  filter(country == "Sweden" | country == "Switzerland")
```

4.7 Test

`{base}`

- `identical()`
  - See also `dplyr::all_equal()`, `dplyr::near()`.
- `match()`
  - `%in%` is the more intuitive binary operator.
- `setequal()`

  - Check two vectors for equality.
  - See also `dplyr::near()`.

- `which()`
  - This function is often unnecessary, according to "Five Tips to Improve Your R Code".
- `which.max()`
  - Use also to identify the first `TRUE`/`FALSE`.
- `which.min()`
  - Use also to identify the first `TRUE`/`FALSE`.

`{dplyr}`

- `all_equal()`
  - Compare data frames.
- `near()`
  - See also `base::identical()`.

Chapter 5

Visualize

"Visualisation is a fundamentally human activity. A good visualisation will show you things that you did not expect, or raise new questions about the data. A good visualisation might also hint that you're asking the wrong question, or you need to collect different data. Visualisations can surprise you, but don't scale particularly well because they require a human to interpret them."
- Garrett Grolemund & Hadley Wickham, *R for Data Science*

5.1 Interfaces

- `shiny`

5.2 Graphs, Charts, and Plots

`{base}`

- `abline()`
  - Add straight lines to a plot.
- `plot()`
- `points()`

`{diagram}`

`{DiagrammeR}`

- `grViz()`

`{GGally}`

- `ggally_box()`
- `ggpairs()`
  - Scatter plot matrix (SPLOM).
- `ggparcoord()`
  - Parallel coordinate plots.

`{ggbeeswarm}`

- `geom_beeswarm()`

`{ggforce}`

- `facet_wrap_paginate()`

`{ggplot2}`

- `aes()`
  - Arguments include `color`, `fill`, `size`, `labels`, `alpha`, `shape` (1-20 accept color attributes and 21-25 accept color and fill attributes), `linewidth`, `linetype`, and `group`.
  - Use `?pch` to see options for `shape`.
  - Note that `aes` can be called within `geom_*` rather than prior to.
  - The default shape for points does not have a `fill` attribute, which means that mapping a categorical variable onto `fill` won't result in multiple colors.
  - Helper functions to include in the call when needing to modify the data include: `stats::reorder()`.
- `coord_cartesian()`
  - Zoom a plot in or out without changing the underlying data.
- `coord_flip()`
  - Flip the x and y axes.
- `coord_polar()`
  - Use to convert a stacked bar chart to a pie chart.
- `element_blank()`
- `element_line()`
- `element_rect()`
- `element_text()`

- `facet_grid()`
  - Lay out panels in a grid.
  - `ROWS ~ COLS`: When using a tilde function, the variable on the left specifies the rows and the variable on the right specifies the columns. When faceting in only one direction (e.g., only on rows), use `.` to specify nothing for the unused direction.
- `facet_wrap()`
  - Wrap a 1D ribbon of panels into 2D (observe a variable, conditional on another variable).
  - See `ggforce::facet_wrap_paginate()` when there are too many panels.
- `geom_abline()`
  - Add reference lines to a plot.
- `geom_bar()`
  - Create a bar chart, where the height of the bar is proportional to the number of cases in each group.
- `geom_boxplot()`
- `geom_col()`
  - Create a bar chart, where the height of the bar represents values in the data.
- `geom_density()`
  - Create a kernal density estimate (a smoothed version of a histogram).
  - Consider using `ggplot2::geom_rug()` with `ggplot2::geom_density()` in order to be transparent about smoothed data.
- `geom_dotplot()`
  - Create a histogram out of dots.
- `geom_errorbar()`
  - Shortcut for `geom_bar(stat = "identity")`.
- `geom_freqpoly()`
  - See `ggplot2::geom_histogram()`.
- `geom_histogram()`
  - Use `y = stat(density)` to rescale the y-axis from counts to a probability estimate.
  - See `ggplot2::geom_freqpoly()`.
- `ggplot2::geom_hline()` -`geom_jitter()`
  - Useful with `ggplot2::geom_boxplot()`.

- `geom_path()`
  - Connect observations in the order in which they appear.
- `geom_point()`
- `geom_rug()`
  - Useful with `ggplot2::geom_density()`.
- `geom_smooth()`
  - Smoothed conditional means; aids the eye in seeing patterns in the presence of overplotting.
- `geom_text()`
- `geom_tile()`
  - Heat maps.
- `geom_violin()`
- `geom_vline()`
- `ggplot()`
- `ggplot_build()`
  - Access calculated values.
- `ggtitle()`
- `group()`
  - Used within `aes()`.
  - Note that `group` is usually unnecessary when specifying `color`, `shape`, `fill`, or `linetype` within `aes`, or when using facets. See "Aesthetics: grouping" for more information.
- `labs()`
  - Modify axis, legend, and plot labels.
- `position_identity()`
  - Don't adjust position.
- `position_dodge()`
  - Dodge overlapping objects side-to-side.
- `position_nudge()`
  - Nudge points a fixed distance.
- `position_stack()`
  - Stack overlapping objects on top of each other, as counts.
- `position_fill()`
  - Stack overlapping objects on top of each other, as densities.

- `position_jitter()`
- `position_jitterdodge()`
- `scale_x_*()`
- `scale_y_*()`
- `scale_color_manual()`
- `scale_fill_*()`
- `scale_shape_*()`
- `scale_size()`
- `scale_linetype_*()`
- `stat_bin()`
- `stat_bin2d()`
- `stat_bindot()`
- `stat_binhex()`
- `stat_boxplot()`
- `stat_contour()`
- `stat_quantile()`
- `stat_smooth()`
- `stat_sum()`
- `theme()`
  - See example for `grid::unit()`.
- `theme_get()`
- `theme_set()`
- `theme_update()`
  - "When you call 'theme_update' and assign it to an object (e.g., called `old`), that object stores the *current* default theme, and the arguments *update* the default theme. If you want to restore the previous default theme, you can get it back by using `theme_update` again." -DataCamp
- `xlab()`
- `xlim()`
- `ylab()`

`{ggridges}`

- `geom_density_ridges()`
  - Create a ridgeline plot.

`{graphics}`

- `boxplot()`
- `hist()`
- `par()`

- `stripchart()`
  - One dimensional scatter plots.
  - Prefer over `ggplot2::ggplot()` when creating one-dimensional plots.

`{grDevices}`

- `colorRamp()`
  - Color interpolation.
- `colorRampPalette()`
  - Color interpolation.

`{grid}`

- `unit()`
  - Create a `unit` object, to specify locations and dimensions in a coordinate system.
  - See example below.

`{plotly}`

- `plot_ly()`

`{RColorBrewer}`

- `brewer.pal()`
  - Make the ColorBrewer color palettes available as R palettes.

5.2.1 Examples

`grid::unit()`

```
# Increase spacing between facets in a `ggplot` object:

plot +
  theme(
    panel.spacing.x = unit(2, "cm"),
    panel.margin = unit(c(1, 2, 1, 1), "cm"))
```

5.3 Graphs, Charts, and Plots - Statistics

{ggplot2}

- mean_cl_boot()
- mean_cl_normal()
- sdl()
- median_hilow()
- stat_function()
  - Compute y values from a function of x values.
- qq()
  - Perfrm calculations for a quantile-quantile plot.
- qq_line()
- summary()
  - Summarize y values at distinct x values.

{stats}

- biplot()

Chapter 6

Model

"Models are complementary tools to visualisation. Once you have made your questions sufficiently precise, you can use a model to answer them. Models are a fundamentally mathematical or computational tool, so they generally scale well. … But every model makes assumptions, and by its very nature a model cannot question its own assumptions. That means a model cannot fundamentally surprise you. - Garrett Grolemund & Hadley Wickham, *R for Data Science*"

`{base}`

- `I()`
  - Use when raising a variable to an exponent in order to evaluate the value as a mathematical expression rather than as an interaction variable.
- `sample()`
- `set.seed()`
- `summary()`
  - Pass the model as the data argument.

`{broom}`

- `augment()`
  - Augment data with information from an object.
- `glance()`
  - Construct a single row summary of a model, fit, or other object.
- `tidy()`
  - Turn an object into a tidy tibble.

{caret}

{class}

- knn()

{cluster}

- pam()
  - Useful for silhouette analysis; similar but not identical to stats::kmeans().

{dplyr}

- sample_n()

{gbm}

{mgcv}

- gam()
  - Generalized additive models (GAMs) with integrated smoothness estimation.
  - Use plot() on a GAM object to view what the function learned from the data.
- s()
  - Indicates which of the explanatory variables should be considered non-linearly.

{mixtools}

{naivebayes}

- naive_bayes()

{pROC}

- auc()
- roc()

{randomForest}

- `randomForest()`

`{rpart}`

- `rpart()`

`{rpart.plot}`

- `rpart.plot()`

`{rsample}`

`{stats}`

- `as.formula()`
  - Use to define an object to pass as a formula into a model (e.g., `fmla <- as.formula("unemployment ~ population"), lm(fmla, data)`)
- `coef()`
  - Extract model coefficients.
- `cor()`
- `cov()`
- `cov2cor()`
  - Scale a covariance matrix into a correlation matrix.
- `cutree()`
  - Pair with `stats::hclust()`.
- `df.residual()`
  - Get the residual degrees of freedom.
- `dist()`
- `hclust()`
- `kmeans()`
- `lm()`
  - Fit linear models.

- `model.matrix()`
  - Use to create dummy variables from categorical data.
- `p.adjust()`
  - Adjust p-values for multiple comparisons.
- `prcomp()`
  - Principal Component Analysis.
- `predict()`
  - Model predictions.
  - Use with `lm` class objects and new data to predict new values (e.g., `predict(model, newdata)`).
  - For GAM, `predict()` returns a matrix so use `as.numeric()` to convert the matrix to a vector if adding to a data set (such as for plotting the predictions versus the actual outcomes).
- `residuals()`
  - Extract model residuals.
- `step()`
  - Step-wise regression.
- `var(libr{tidyr}‘`
- `nest()`
  - Nest repeated values in a list-variable.
  - Helpful when separating a data frame in preparation to model the data for each grouping.

`{vtreat}`

- `kWayCrossValidation()`
- `splitPlan()`
  - Use to create a cross-validation plan.

`{WVPlots}`

- `GainCurvePlot()`

Chapter 7

Communicate

"The last step of data science is communication, an absolutely critical part of any data analysis project. It doesn't matter how well your models and visualisation have led you to understand the data unless you can also communicate your results to others."
- Garrett Grolemund & Hadley Wickham, *R for Data Science*

7.1 CSS

Center Title

```
<style type="text/css">

h1.title {
  text-align: center;
}
</style>
```

7.2 Export

`{base}`

- `file.path()`
- `print()`
  - Use the `include.rownames = FALSE` argument to remove row numbers (or names) from the output.

- `save()`
- `saveRDS()`
  - See "A better way of saving and loading objects in R" to understand the differences between `save()` and `saveRDS()`.

`{readr}`

- `write_csv()`
- `write_delim()`
  - About twice as fast as `utils::write.csv()` and never writes row names.
- `write_excel_csv()`
- `write_tsv()`

`{utils}`

- `write.csv()`
- `write.csv2()`
- `write.table()`
  - Prefer `readr::write_delim()` to `utils::write.table()`.

`{XLConnect}`

- `loadWorkbook()`

7.3 Format Output

`{base}`

- `format()`

`{DT}`

- `datatable()`
  - Use `options = list(dom = 't')` to remove the search bar.
  - Use `options = list(dom = "t", lengthchange = FALSE)` to remove the search bar and box that allows the user to choose how many rows to see.

  - Use `options = list(columnDefs = list(list(className = "dt-center", targets = (column_position))))` to center column values (where `0` indicates the first column).

- `formatCurrency()`

`{gt}`

- `gt()`
  - Alternative to `knitr::kable` and `DT::datatable`, styled after `ggplot2`.

`{kableExtra}`

- `kable_styling()`
- `add_footnote()`

`{knitr}`

- `kable()`

`{lubridate}`

- `stamp()`
  - Format dates and times based on human-friendly templates.

`{scales}`

- `comma()`
- `dollar()`

7.4 Graphics

`{knitr}`

- `include_graphics()`
  - Embed external images in 'knitr' documents.
  - Preferable to the `![alt text or image title](path/to/image)` Markdown syntax for embedding an external image, as `include_graphics` offers more control over the attributes of the image.

7.5 Plots

`{ggplot2}`

- `element_*()`
  - Specify the display of how non-data components of a plot are drawn.
- `labs()`
  - Modify axis, legend, and plot labels.
  - Child functions: `xlab()`, `ylab()`, `ggtitle()`
- `theme()`
  - Customize the non-data components of a plot (see example below).
  - Use `?theme_classic` for a list of predefined themes.

7.5.1 Examples

`ggplot2::theme()`

```
ggplot(plot_data_2006) +
  geom_histogram(aes(x = working_hours)) +
  labs(
    x = "Working hours per week",
    y = "Number of countries") +
  theme(
    text = element_text(family = "Bookman", color = "gray25"))
```

7.6 RStudio Connect

`{rsconnect}`

- `writeManifest()`

7.7 RMarkdown

- See RMarkdown: The Definitive Guide
- See Mozzila Developer Network for CSS help.

Chapter 8

Program

"Surrounding [the tools for importing, tidying, transforming, visualising, modeling, and communicating data] is programming. Programming is a cross-cutting tool that you use in every part of a project. You don't need to be an expert programmer to be a data scientist, but learning more about programming pays off because becoming a better programmer allows you to automate common tasks, and solve new problems with greater ease."
- Garrett Grolemund & Hadley Wickham, *R for Data Science*

8.1 Characters

References:

- `?Quotes`

8.2 Conditionals & Control Flows

`{base}`

- Control (access documentation using `?Control`)
  - `if (cond) expr`
    - The key difference between `if (cond) expr` and `ifelse` is that `if (cond) expr` will evaluate only the first element of an object with `length > 1`. See the documentation for each function and the ateucher's GitHub example titled "R: if vs ifelse" to learn more.

  - `for (var in seq) expr`
  - `while (cond) expr`
  - `repeat expr`
  - `break`
  - `next`

- `identical()`: Test objects for exact equality.
  - Use `identical()` rather than `==` and `!=` in `if` and `while` statements to test for equality.
- `ifelse()`: Conditional element selection.
  - `dplyr::if_else()` is more strict by checking the object type.
- `stop()`: Stop execution of the expression and execute an error action.
  - Useful in combination with an `if` statement when you want to generate helpful error messages.
- `stopifnot()`: Ensure the truth of an R expression.
  - Prefer `base::stop()` to `base::stopifnot()`.

`dplyr`

- `case_when()`: A general vectorized `if`.
- `if_else()`: Vectorized `if`.

8.3 Environment and Workspace

`{base}`

- `.libPaths()`
- `baseenv()`
  - The environment of the `{base}` package; it's enclosing environment ("parent environment") is the empty environment.
- `dir()`
  - List the files in a directory.
- `dir.create()`
  - Create a file path in the computer's file system.
- `emptyenv()`
  - The empty environment, which is the ancestor of all environments and the only environment without an enclosing environment.

- `environment()`
  - The current environment.
- `environmentName()`
- `exists()`
  - Remember that `R` will look for an object in parent environments until it reaches the empty environment, so use `inherits = FALSE` to limit the search to only the current environment.
- `getOption()`
  - Set and examine global options.
- `getwd()`
  - Get the working directory.
- `globalenv()`
  - The environment in which you normally work, it's enclosing environment is the last package attached with `library()` or `require()`.
- `history()`
  - Display the previous 25 commands.
- `install.packages()`
  - Install packages from repositories or local files.
- `library()`
  - Load and attach packages, returning an error if the packages does not exist.
- `list.files()`
  - List the files in a directory/folder.
- `loadedNamespaces()`
  - Return the loaded name spaces.
- `loadhistory()`
  - Recall command history.
- `ls()`
  - List objects in the specified environment.
- `list2env()`
  - From a list, build or add to an environment.
- `new.env()`
  - Create a new environment.

- `options()`
  - Set and examine global options.
- `parent.env()`
  - Return the enclosing environment of the environment listed as an argument.
  - `base::parent.env()` returns information that can be unhelpful, so use with `base::environmentName()`, as follows: `parent.env(environment_name) %>% environmentName()`.
- `q()`
  - Terminate an R session.
- `R.version()`
  - Version information.
- `R.version.string()`
  - Version information.
  - Same call as `R.version$version.string()`.
- `require()`
  - Load and attach packages, returning `FALSE` if the package does not exist.
- `rm()`
  - Remove objects from a specified environment.
- `savehistory()`
  - Save command history (default value is ".Rhistory").
- `save.image()`
  - Save the current workspace.
- `search()`
  - Return a list of attached packages and R objects.
- `searchpaths()`
  - Return the path to attached packages.
- `setwd()`
  - Set the working directory file path.
  - When using Windows, use "/" instead of "".
- `Sys.getenv()`
  - See also `Sys.setenv()`.

- `Sys.info()`
  - Extract system and user information.
  - Example: `Sys.info()[c("sysname", "release")]`.
- `Sys.setenv()`
  - See also `Sys.setenv()`.

`{gdata}`

- `object.size()`
  - Report the space allocated for an object.
  - See also `utils::object.size()`.

`{here}`

- `here()`

`{installr}`

- `updateR()`
  - Check for the latest R version; downloads and installs new R versions.

`{pryr}`

- `where()`: Find where a name is defined.

`{utils}`

- `ls.str()`: List objects and their structure.
- `object.size()`: Report the space allocated for an object.
  - See also `gdata::object.size()`.
- `sessionInfo()`: Collect information about the current R session.

References:

- "Environments" (Hadley Wickham, *Advanced R*)

8.4 Evaluation (Standard and Non-standard)

`base`

- `cat()`
  - Concatenate and print.
- 'print()
  - Print the argument to the Console.
  - A shortcut to `print()` is to place the code you want printed inside parentheses.
- `quote()`
  - Return the argument, unevaluated.
- `writeLines()`
  - Display quotes and backslashes as they would be read, rather than as R stores them (i.e., see the raw contents of the string, as the `print()` representation is not the same as the string itself).

`rlang`

- Quosures
  - `enquo()`, `new_quosure()`, `quo()`.

References:

- "Non-standard evaluation" (Hadley Wickham, *Advanced R*)
- "Non-standard evaluation" (Hadley Wickham, `lazyeval` package vignette)
- "Programming with dplyr" (dplyr.tidyverse.org)

8.5 Functionals

`{base}`

- Apply Functions
  - `apply`
    - Apply functions over array margins.
  - `lapply`

        * Apply a function over a list or vector.
    - `sapply`
        * Apply a function over a list or vector and return a vector or matrix.
    - `vapply`
        * A safer version of `sapply`, as it requires the output type to be predetermined.
    - `mapply`
        * Apply a function to multiple list or vector arguments.
    - `rapply`
        * Recursively apply a function to a list.
    - `tapply`
        * Apply a function over a ragged array.

`{purrr}`

- `map`
    - Apply a function to each element of a vector.
    - `tidyr::unnest` is useful in changing the list-column output of `map` into rows.
- `map2`
    - Map over multiple inputs simultaneously.
- `map_if`
    - Apply a function to elements of that match a condition.
- `possibly`
    - Uses a default value whenever an error occurs.
- `quietly`
    - Capture side effects in a list with components `result`, `output`, `messages`, and `warnings`.
- `safely`
    - Capture side effects in a list with components `result` and `error`.
- `transpose`
    - Transpose a list (turn a list-of-lists inside-out).

8.6 Functions

`{assertive}`

- `assert_*()`
  - Check whether the input is `*` (e..g, `assert_is_numeric()`) and throw and error if the input does not meet the condition.
- `coerce_to()`
  - Coerce the input to a different class, with a warning.
- `is_*()`
  - Checks whether the input matches the condition specified by `*` (e.g., `assertive::is_non_positive()`).
- `use_first()`
  - Use only the first element of a vector.

`{base}`

- `do.call()`
  - Execute a function call from a name or a function and a list of arguments to be passed to the function.
- `invisible()`
  - Return a (temporarily) invisible copy of an object.
- `match.arg()`
  - Argument verification. Useful when matching a character argument specified in the function signature. See example.
- `message()`
  - Generate a diagnostic message.
  - Preferable to generating a message using `cat`.
- `return()`
  - Return a value from a function.
  - Useful in `if` statements where one condition is simple and the other is complex (see section 19.6.1 “Explicit return statements” in Hadley Wickham’s *R for Data Science*.
- `setNames()`
  - Set the names in an object.
  - Useful in function writing; see documentation.

- `stop()`
  - Stop execution of the expression and execute an error action.
  - Useful in combination with an `if` statement when you want to generate helpful error messages.
- `stopifnot()`
  - Ensure the truth of an R expression.
  - See section 19.5.2 "Checking values" in Hadley Wickham's *R for Data Science* for a discussion of `stop()` versus `stopifnot()`.
  - Consider functions from `{assertive}` as an alternative to `stopifnot` and `stop`.
- `unlist()`
  - Flatten lists.
  - Useful when using `purrr`'s `map` functions, which return objects as type `list`.

`{zeallot}`

- `%<-%`
  - Multiple assignment operator (see example below).

8.6.1 Examples

`base::args()`

```
args(prop.test)
```

```
## function (x, n, p = NULL, alternative = c("two.sided", "less",
##     "greater"), conf.level = 0.95, correct = TRUE)
## NULL
```

```
# The body of `prop.test` contains the following line of code:
# `alternative <- match.arg(alternative), which reassigns it to the selected
# character vector.
```

`zeallot::%<-%`:

```
session <- function() {

  list(
    r_version        = R.version.string,
```

```
    operating_system = Sys.info()[c("sysname", "release")],
    loaded_pkgs      = loadedNamespaces()
  )
}

c(vrsn, os, pkgs) %<-% session()
```

8.7 Learn About an Object

- `?object_name`
- `??object_name`

`{base}`

- `args()`
- `attributes()`
  - View or assign an objects attributes (e.g., `class()`, `dim()`, `dimnames()`, `names()`, `row.names()`).
- `body()`
  - Get or set the body of a function.
- `colnames()`
- `dim()`
  - Retrieve or set the dimnames of an object.
- `dimnames()`
  - Retrieve or set the dimension names of an object.
- `formals()`
  - Get or set the formal arguments of a function.
- `help()`
  - Get the topic documentation.
- `help.search()`
  - Search the help system for documentation matching a given character string.
- `vignette()`
- `rownames()`

8.8 Loops

- `base::seq()`: Sequence generation (this functions makes `length()` unnecessary).
- `base::seq_along()`: In `for` loops, safer than using `base::ncol()` or `base::nrow()`.

8.9 Optimization

`microbenchmark::microbenchmark()`: Sub-millisecond accurate timing of expression evaluations. - A more accurate replacement of `system.time(replicate(1000, expr))`.

8.10 Pipes

- `magrittr::%<>%`: Compound assignment-pipe operator.
- `magrittr::%>%`: Forward-pipe operator.
- `magrittr::%$%`: Expositions-pipe operator.
- `magrittr::add()`: `+`, for pipes.
- `magrittr::and()`: `&`, for pipes.
- `magrittr::extract()`:`[`, for pipes (see also `purrr::pluck()`).
- `magrittr::extract2()`: `[[`, for pipes.
- `magrittr::freduce()`: Apply a list of functions sequentially.
- `magrittr::is_in()`: `%in%`, for pipes.
- `magrittr::multiply_by()`: `*`, for pipes.
- `magrittr::or()`: `|`, for pipes.
- `magrittr::raise_to_power()`: `^`, for pipes.
- `magrittr::subtract()`: `-`, for pipes.

8.11 Popups

- `svDialogs::dlg_message()`: Display a modal message box (works in Windows, MacOS, and Linux).
- `tcltk::tk_messageBox()`: Display a generic message box using Tk (Windows-specific).

8.12 Selecting & Subsetting

- `.$variable_name`: See example below.

- `.[["variable_name"]]`: See example below.
- `base::subset()`
- `dplyr::first()`
- `dplyr::las()t`
- `dplyr::nth()`
- `dplyr::rename()`
- `dplyr::select()`: Helper functions include `contains()`, `ends_with()`, `matches()`, `num_range()`, `one_of()`, `starts_with()`.
- `ggplot2::cut_number`
- `magrittr::extract()`
- `magrittr::extract2()`
- `purrr:pluck()`: See also `magrittr::extract2()`.

8.12.0.1 Examples

- `.$variable_name`:

```
ui_summary_table <-
  aws_vendors %>%
  filter(str_detect(vendor_name, "UTAH INTERACTIVE")) %>%
  .$vendor_id %>%
  map(query_summary_table) %>%
  bind_rows()
```

- `.[["variable_name"]]`:

```
odbc_aws %>%
  dbGetQuery(
    paste("
          SELECT id
          FROM batch
          WHERE entity_id = ", t_id, "
          AND status IN ('PROCESSED', 'PROCESSING')"
    )
  ) %>%
  .[["id"]] %>%
as.double()
```

8.12.0.2 References

- "Indexing lists in #rstats. Inspired by Residence Inn" (Hadley Wickham, Twitter, 14 September 2015)

8.13 Style Guide

- "The Tidyverse Style Guide" by Hadley Wickham.
- General Layout and Ordering (taken from the outdated "Google's R Style Guide")
  - Title
  - Author
  - File description (e.g., purpose of program, inputs, outputs)
  - `source()` and `library()` statements
  - Function definitions
  - Executed statements.

8.14 System Commands

`{base}`

- `shell()`
- `system()`
- `system2()`

`{sys}`

- `exec()`
- `exec_r()`

8.15 Version Control

- `base::update.packages()`
  - 00LOCK error when updating a package: Use `update.packages(ask = FALSE, checkBuilt = TRUE, INSTALL_opts = "--no-lock")`
- `installr::updateR()`
  - Remove old versions from Windows:
    - Control Panel > All Control Panel Items > Programs and Features > uninstall old versions
    - C: > Program Files > R > delete old versions.
- `packrat::snapshot()`: Capture and store the packages and versions in use.
- `packrat::restore()`: Load the most recent snapshot to the project's private library.
- Packrat still seems to be under development. The idea sounds good, but in practice the packrat package has caused lots of problems.

8.15.0.1 Git

- *A successful Git branching model*
- git-scm.com
- *Git and GitHub* by Hadley Wickham
- *Happy Git and GitHub for the useR* by Jenny Bryan
- *Pro Git* by Scott Chacon and Ben Straub
- *Understanding the GitHub flow*
- `git branch`: List, create, or delete branches.
  - `git branch -d <branch_name>`: Delete a local branch.
    - See "Delete branch in RStudio pop-up" for help removing branches in RStudio after removing them from Git.
- Git and R Projects
  - R Projects (.Rproj files) should not be nested. Doing so causes problems with Git when tracking changes to a child .Rproj file.

Chapter 9

Python

9.1 Import

Base Python

- `import ... as ...`

`Numpy`

- `array()`

9.2 Tidy

"Tidying your data means storing it in a consistent form that matches the semantics of the dataset with the way it is stored. In brief, when your data is tidy, each column is a variable and each row is an observation. Tidying data is important because the consistent structure lets you focus your struggle on questions about the data, not fighting to get the data into the right form for different functions."
- Garrett Grolemund & Hadley Wickham, *R for Data Science*

Base Python

- `shape`
- `type()`

`Numpy`

-

9.3 Transform

"Transformation includes narrowing in on observations of interest (like all people in one city, or all data from the last year), creating new variables that are functions of existing variables (like computing velocity from speed and time), and calculating a set of summary statistics (like counts or means)."
- Garrett Grolemund & Hadley Wickham, *R for Data Science*

9.4 Visualize

9.5 Model

9.6 Communicate

9.7 Program

"Surrounding [the tools for importing, tidying, transforming, visualising, modeling, and communicating data] is programming. Programming is a cross-cutting tool that you use in every part of a project. You don't need to be an expert programmer to be a data scientist, but learning more about programming pays off because becoming a better programmer allows you to automate common tasks, and solve new problems with greater ease." - Garrett Grolemund & Hadley Wickham, *R for Data Science*

Base Python

- `help()`
- `len()`
- `print()`
- `type()`

9.8 Reference

- *R to Python: Data Wrangling with dplyr and pandas*