Functions Learned So Far

Ronnie Steinitz

2025-11-06

Week 1: Is My Data Clean? Exploring, Diagnosing, and Visualizing Problems

0. Load Required Packages

- `library()` # Loads an installed package into your R session so its functions can be used.
  Example: `library(tidyverse)` → loads the tidyverse collection of packages.
- `janitor::clean_names()` # Cleans column names (lowercase, underscores instead of spaces/symbols).
  Example: `data <- janitor::clean_names(data_raw)` → turns "Flipper Length (mm)" into "flipper_length_mm".

1. Load and Preview Dataset

- `getwd()` # Shows the current working directory (the folder R is looking in by default).
  Example: `getwd()` → might return "/Users/rsteinitz/Documents/github/R Data Analysis Course".
- `setwd()` # Sets the working directory (where R should look for or save files).
  Example: `setwd("/Users/rsteinitz/Documents/github/R Data Analysis Course")`.
- `read_csv()` # Reads a .csv file into R as a data frame (from the readr package).
  Example: `data_raw <- read_csv("Week 1/Palmer Penguins Raw.csv")`.
- `glimpse()` # Provides a compact overview of a dataset (rows, columns, and types).
  Example: `glimpse(data)` → shows columns, data types, and sample values.
- `str()` # Displays the structure of an object.
  Example: `str(data)` → tells you number of rows, columns, and types.
- `head()` # Prints the first 6 rows of a dataset.
  Example: `head(data)` → shows the top rows of the penguins dataset.
- `names()` # Lists the column names in a dataset.
  Example: `names(data)` → returns column headers like "species", "island", "sex".
- `View()` # Opens the dataset in a spreadsheet-like viewer (interactive).
  Example: `View(data)` → opens a new tab in RStudio with your dataset.

2. Diagnosing Data Types and Structure

- `class()` # Shows the data type (numeric, character, factor, etc.) of an object.
  Example: `class(data$sex)` → returns "character".
- `table()` # Summarizes counts of unique values in a variable.
  Example: `table(data$species)` → counts how many penguins belong to each species.
- `unique()` # Lists unique values in a variable.
  Example: `unique(data$island)` → shows "Biscoe", "Dream", "Torgersen".
- `length()` # Tells how many elements are in a vector.
  Example: `length(unique(data$flipper_length_mm))` → number of distinct flipper lengths.
- `count()` # Counts rows by categories of a variable (from dplyr).
  Example: `count(data, island)` → counts penguins per island.

3. Missing Data: Detection and Summary

- `is.na()` # Tests whether values are missing (returns TRUE/FALSE).
  Example: `is.na(data$sex)` → shows TRUE for rows missing sex info.
- `colSums()` # Adds up values across each column. Often used with is.na().
  Example: `colSums(is.na(data))` → number of NAs in each column.
- `sum()` # Adds up all numeric values, or counts TRUE values in logical vectors.
  Example 1: `sum(is.na(data$flipper_length_mm))` → number of missing flipper lengths.
  Example 2: `sum(data$flipper_length_mm > 200, na.rm = TRUE)` → number of penguins with long flippers.
- `summary()` # Gives descriptive statistics (mean, median, min, max).
  Example: `summary(data$bill_depth_mm)` → outputs min, max, mean, etc.
- `range(..., na.rm = TRUE)` # Shows the minimum and maximum values.
  Example: `range(data$bill_length_mm, na.rm = TRUE)` → min and max bill length.

4. Basic Visualizations

- `hist()` # Creates a histogram of a numeric variable (base R).
  Example: `hist(data$flipper_length_mm)`.
- `ggplot()` # Starts a ggplot graph.
  Example: `ggplot(data, aes(x = flipper_length_mm)) + geom_histogram()`.

All `ggplot()` plots must have three basic components: `data`, `aes`, and a `geom`

- `aes()` # Maps variables to visual properties.
  Example: `aes(x = species, fill = sex)`.
- `geom_histogram()` # Adds a histogram layer in ggplot.
  Example: `geom_histogram(binwidth = 2, fill = "steelblue")`.
- `facet_wrap()` # Splits one plot into multiple panels by a grouping variable.
  Example: `facet_wrap(~ species)` → separate histograms per species.
- `geom_bar()` # Creates a bar chart for categorical variables.
  Example: `geom_bar()` → counts penguins per species.

By the end of Week 1, you should be comfortable with:

- Importing and previewing data (`read_csv()`, `glimpse()`, `head()`, `names()`).
- Checking and diagnosing data types (`class()`, `unique()`, `table()`).
- Detecting and summarizing missing data (`is.na()`, `colSums()`, `summary()`).
- Converting variables to correct types (`mutate()`, `as.factor()`).
- Making basic plots (`hist()`, `ggplot()`, `geom_bar()`, `facet_wrap()`, `geom_histogram()`).

Week 2: Wrangling Basics – Select, Filter, Mutate

1. Pipe Operator

- `%>%` (pipe operator) # Sends the output of one function as the input to the next.
  Example: `data_raw %>% clean_names() %>% glimpse()`.

2. Select Columns

- `select()` # Keeps or drops specific columns.
  Example 1: `data %>% select(species, island)` → keep these columns.
  Example 2: `data %>% select(-comments)` → drop the comments column.

3. Filter Rows

- `filter()` # Keeps rows meeting conditions.
  Example 1: `filter(data, species == "Adelie")`.
  Example 2: `filter(data, flipper_length_mm > 200)` → penguins with long flippers.
  Example 3: `ggplot(data %>% filter(flipper_length_mm > 200)) + geom_bar()`.
- Logical operators: `==` equal, `!=` not equal, `>`, `<`, `&` (and), `|` (or).
  Example: `filter(data, species == "Adelie" & island == "Dream")`.

4. Mutate / Create New Variables

- `mutate()` # Adds or transforms columns in a dataset.
  Example: `data <- mutate(data, body_mass_kg = body_mass_g / 1000)`.
- `as.factor()` # Converts a variable into a factor (categorical variable).
  Example: `data$sex <- as.factor(data$sex)`.
- `as.numeric()`, `as.integer()`, `as.logical()`, `as.character()`, `as.Date()` # Convert variables between data types.
  Example: `as.Date(data$date_egg, format = "%m/%d/%y")`.
- `case_when()` # Recode values or create categories.
  Example: `data %>% mutate(size_class = case_when(flipper_length_mm < 185 ~ "Small", flipper_length_mm >= 200 ~ "Large"))`.
- `ifelse()` # Conditional operation: if [condition], then do [action], otherwise do [different action].
  Example: `data %>% mutate(size_class = ifelse(flipper_length_mm > 200, "Large", "Small"))`.

- `word()` # Extracts words from a text string.
  Example: `data %>% mutate(species_simple = word(species, 1))` → "Adelie", "Gentoo", "Chinstrap".

5. Visualization

- `labs()` # Adds or edits labels for titles, axes, and legends.
  Example: `labs(title = "Penguin Counts", x = "Species", y = "Number of Penguins")`.
- `scale_fill_manual()` # Manually sets fill colors.
  Example: `scale_fill_manual(values = c("male" = "blue", "female" = "red"))`.

By the end of Week 2, you should be comfortable with:

- Using `%>%` pipes to link commands together and write clean, readable code.
- Selecting specific columns with `select()`.
- Filtering rows with conditions using `filter()`.
- Creating new variables with `mutate()` and `case_when()`.
- Visualizing subsets of data using `ggplot()` with filters.

Week 3: Grouping and Summarizing

1. Grouping and Summarizing

- `group_by()` # Defines how data should be split into groups before performing summaries.
  Example 1: `data %>% group_by(continent)` → creates a grouped tibble by continent.
  Example 2: `data %>% group_by(continent, year)` → groups simultaneously by continent and year.
- `summarize()` / `summarise()` # Reduces each group to summary statistics you specify.
  Example: `data %>% summarize(mean_life_exp = mean(life_exp, na.rm = TRUE))`
- `n()` # Counts the number of rows (observations) in each group.
  Example: `data %>% group_by(continent) %>% summarize(n_countries = n())`
- `arrange()` # Orders rows by one or more variables.
  Example 1: `arrange(data, life_exp)` → ascending.
  Example 2: `arrange(data, desc(life_exp))` → descending.
- `write_csv()` # Writes a data frame to a .csv file.
  Example: `write_csv(data1, "Mean Life Expectancy by Continent.csv")`

2. Combining Operations with Pipes

- `filter()` → `group_by()` → `summarize()` # Common workflow: filter your data, group it, then summarize.
  Example: `data %>% filter(year == 2007) %>% group_by(continent) %>% summarize(mean_life_exp = mean(life_exp, na.rm = TRUE), n_countries = n())`

3. Visualizing Summaries

- `geom_col()` # Creates a bar plot using pre-computed values (like group means).
  Example: `ggplot(data1, aes(x = continent, y = mean_life_exp)) + geom_col()`
- `geom_boxplot()` # Shows the distribution of a numeric variable across groups.
  Example: `ggplot(data, aes(x = continent, y = life_exp)) + geom_boxplot()`
- `geom_line()` # Connects data points by group to show change over time.
  Example: `ggplot(data3, aes(x = year, y = mean_life_exp, color = continent)) + geom_line()`

By the end of Week 3, you should be comfortable with:

- Grouping data using `group_by()` and computing summaries with `summarize()`.
- Counting observations with `n()` and ordering results using `arrange()`.
- Using pipes to link filtering, grouping, and summarizing steps cleanly.
- Exporting summarized tables using `write_csv()`.
- Visualizing summarized data with `geom_col()`, `geom_boxplot()`, and `geom_line()`.

Week 4: Joining and Reshaping

1. Reshaping Data

- `pivot_longer()` # Converts data from *wide* to *long* format (columns → rows).
  Example: `forest_long <- forest %>% pivot_longer(cols = -country, names_to = "year", values_to = "forest_area")`
- `pivot_wider()` # Converts data from long back to wide format (rows → columns). Example: `data_wide <- data_long %>% pivot_wider(names_from = year, values_from = forest_area)`
- `str_replace()` # Replaces patterns in text using regular expressions. Example: `mutate(year = as.numeric(str_replace(year, "^x", "")))` Removes the "x" from year names (e.g., "x1990" → "1990") and converts to numeric.

2. Joining Datasets

- `left_join()` # Combines two datasets, keeping all rows from the *left* (first) dataset.
  Example: `forest_clim <- left_join(forest_long, climate, by = c("country", "year"))`
  Merges forest area and climate data using shared columns (`country` and `year`).
- `inner_join()` # Keeps only rows that appear in both datasets.
  Example: `overlap <- inner_join(forest_long, climate, by = c("country", "year"))`
  Useful when you only want complete overlap (no missing data across sources).
- `anti_join()` # Returns rows in the first dataset with *no match* in the second.
  Example: `missing <- anti_join(forest_long, climate, by = "country")`
  Helps identify missing countries or mismatched names before joining.

3. Summarizing and Visualizing Joined Data

- `group_by()` + `summarize()` # Collapse data into group-level summaries.
  Example: `country_summary <- joined_cont %>% group_by(country, continent) %>% summarize(mean_temp = mean(annual_mean, na.rm = TRUE), mean_forest = mean(percent_forest, na.rm = TRUE))`
  Summarizes data by country and continent to calculate mean temperature and mean forest cover.
- `geom_point()` # Plots relationships between two numeric variables.
  Example: `ggplot(country_summary, aes(x = mean_temp, y = mean_forest, color = continent)) + geom_point()`
  Used here to visualize the relationship between mean annual temperature and mean percent forest cover by country.

By the end of Week 4, you should be comfortable with:

- Reshaping wide data into long format using `pivot_longer()` (and back using `pivot_wider()` if needed).
- Cleaning text-based variables using `str_replace()` and converting them to numeric.
- Joining multiple datasets using `left_join()`.
- Creating derived variables (like **percent forest cover**) with `mutate()`.
- Summarizing and visualizing joined datasets using `group_by()`, `summarize()`, and `geom_point()`.

Week 5: Skews and Transformations

1. Transforming Data

`log()` or `log10()` # Applies a logarithmic transformation to reduce strong right-skew or compress large values.
Example: `mutate(log_insulin = log10(insulin))` → transforms insulin values Example: `mutate(log_insulin = log10(insulin + 1))` → log10(0) produces `NA`, so add `+ 1` to the equation to keep every zero (0) observation as a zero in the dataset.

`sqrt()` # Applies a square root transformation to reduce moderate right-skew.
Example: `mutate(sqrt_bmi = sqrt(BMI))`

`scale()` # Standardizes numeric variables to z-scores (mean = 0, SD = 1).
Example: `mutate(z_glucose = as.numeric(scale(glucose)))` → compares variables measured in different units.

2. Exploring Distribution Shape

`geom_density()` # Creates a smooth curve showing the distribution of a numeric variable.
Example: `ggplot(data, aes(x = insulin)) + geom_density(fill = "steelblue", alpha = 0.6)`

`geom_smooth(method = "lm")` # Adds a linear regression trend line to a scatterplot.
Example: `geom_smooth(method = "lm", se = FALSE)` → shows whether the relationship between variables is linear. `se = FALSE` removes the grey standard error ribbon around the regression line.

By the end of Week 5, you should be comfortable with:

- Recognizing skewed vs. normal distributions using density plots.
- Applying transformations (log(), sqrt(), scale()) to make data more interpretable or meet model assumptions.
- Comparing raw and transformed variables visually using scatterplots and trend lines.

Week 6: Relationships – Regression and Correlation

1. Modeling and Correlation

- `lm()` # Fits a linear regression model (estimates intercept and slope).
  Example: `lm(flipper_length_mm ~ bill_length_mm, data = my_data)` → models flipper length as a function of bill length.
- `summary()` (for models) # Summarizes model output, including coefficients, R^2, and p-values.
  Example: `summary(model)` → prints regression table and model statistics.
- `cor()` # Computes correlation between two numeric variables.
  Example: `cor(bill_length_mm, flipper_length_mm, use = "complete.obs")` → returns the Pearson correlation coefficient.

2. Accessing Model Components

- `summary(model)$r.squared` # Retrieves the R^2 value from the model summary.

3. Visualization and Annotation

- `geom_smooth(method = "lm")` # Adds a fitted regression line to a plot.

By the end of Week 6, you should be comfortable with:

- Fitting and summarizing linear models using `lm()` and `summary()`.
- Calculating and interpreting correlation with `cor()`.
- Accessing model components directly with `$` and `summary(model)$r.squared`.
- Adding regression lines using `geom_smooth(method = "lm")`.

Week 7: Comparing Groups – Boxplots & Statistical Tests

1. Statistical Tests

- `t.test()` # Performs a two-sample or paired t-test to compare group means.
  Example: `t.test(body_mass_g ~ species, data = penguins)` → compares mean body mass between two species.
- `wilcox.test()` # Performs a nonparametric Wilcoxon rank-sum or signed-rank test.
  Example: `wilcox.test(body_mass_g ~ species, data = penguins)` → compares medians when normality is violated.

- `aov()` # Fits a one-way ANOVA model to test for mean differences across three or more groups.
  Example: `aov(body_mass_g ~ species, data = penguins)`

- `TukeyHSD()` # Conducts post-hoc pairwise comparisons after a significant ANOVA.
  Example: `TukeyHSD(model1)` → identifies which species pairs differ significantly.

- `var.test()` # Tests whether two samples have equal variances.
  Example: `var.test(body_mass_g ~ species, data = penguins)`

2. Visual Diagnostics

- `stat_qq()` # Adds a Q–Q plot layer showing data quantiles vs. theoretical normal quantiles.
  Example: `ggplot(penguins, aes(sample = body_mass_g)) + stat_qq()`

By the end of Week 7, you should be comfortable with:

- Performing t-tests, Wilcoxon tests, and ANOVA (`t.test()`, `wilcox.test()`, `aov()`).
- Running post-hoc analyses with `TukeyHSD()`.
- Checking assumptions with Q–Q plots (`stat_qq()`, `stat_qq_line()`) and variance tests (`var.test()`).
- Using boxplots to visualize group differences and interpret potential mean or median shifts.