DaSEH R Cheatsheet

Basic R

Major concepts

- **Package:** A bundle of code and/or data that can be loaded for repeated use or sharing, like an expansion pack.
- **Object:** Something that can be worked with in R.
- **Function:** A piece of code that performs a task in R. Functions can come from base R, additional packages, or your own code.
- **Argument:** An option specified in a function.
- **`tidyverse`:** A set of packages designed for data science that can make code more intuitive than base R.
- **R console:** A full calculator:
  - `+`, `-`, `/`, and `*` add, subtract, divide, and multiply.
  - `^` or `**` raises to a power.
  - Parentheses, `(` and `)`, control order of operations.
  - `%%` finds the remainder.
- **Comments:** `#` is the comment symbol; nothing to its right is evaluated.

Functions

Library/Package	Piece of code	Example of usage	What it does
Base R	`<-`	`x <- 1`	Assigns a name, `x`, to something in the R environment.
Base R	`c()`	`x <- c(1, 3)`	Combines values into a vector.
Base R	`str()`	`str(x)`	Summarizes the structure of object `x`.
Base R	`class()`	`class(x)`	Returns the type of values in object `x`.
Base R	`length()`	`length(x)`	Returns the length of object `x`.
Base R	`seq()`	`seq(from = 0, to = 100, by = 5)`	Generates regular sequences.
Base R	`rep()`	`rep(1, times = 10)`	Replicates values. Can use the `times` or `length.out` argument.

Library/Package	Piece of code	Example of usage	What it does
Base R	`sample()`	`sample(1:12)`	Samples from values, with or without replacement. `replace = TRUE` samples with replacement.
Base R	`install.packages()`	`install.packages("tidyverse")`	Installs packages.
Base R	`library()`	`library(tidyverse)`	Loads and attaches packages to the R environment. Run each time you start R.

RStudio

Major concepts

- **RStudio:** An Integrated Development Environment (IDE) for R that makes R easier to use.
- **Source/Editor:** Where code is written and saved in an `.R` script.
- **R Console:** Where code is executed. Code entered here is not saved on disk.
- **Workspace/Environment:** Shows objects currently loaded in R.
- **R Markdown:** `.Rmd` files that generate reports with code and output.
- **R Project:** Helps organize work, manage working directories, and track the active project.
- **Getting help:** For any function, use `?FUNCTION_NAME` or `help("FUNCTION_NAME")` to view its help file.
- **RStudio keyboard shortcuts:** RStudio keyboard shortcuts

Functions

Library/Package	Piece of code	Example of usage	What it does
Base R	`View()`	`View(mtcars)`	Opens data in a spreadsheet-like viewer.
Base R	`head()`	`head(mtcars)`	Returns the first 6 rows by default. Use the `n` argument to specify the number of rows.
Base R	`tail()`	`tail(mtcars)`	Returns the last 6 rows by default. Use the `n` argument to specify the number of rows.

Reproducibility

Major concepts

- **Reproducibility/Reproducible:** A different analyst re-runs the analysis with the same code and data and gets the same result.
- **Repeatable:** The same analysis is repeated under the same conditions and gives the same results.
- **Replicable:** New data are used to test whether the same inferences hold.

Functions

Library/Package	Piece of code	Example of usage	What it does
Base R	`sessionInfo()`	`sessionInfo()`	Returns the R version, OS, and attached packages in the current R session.
Base R	`sample()`	`sample(x, size = 10, replace = FALSE)`	Returns a random vector of a specified size, with or without replacement.
Base R	`set.seed()`	`set.seed(1234)`	Makes random-number output reproducible for subsequent code.

More resources

- The R Markdown book
- Jenny Bryan's organizational strategies
- Write efficient R code for science
- Reproducibility in Cancer Informatics course

Data Input

Major concepts

- **Delimited file:** A file in which columns are separated by punctuation, such as commas in CSV files or tabs in TSV/TXT files.
- **Tibble:** A rectangular data frame with data organized into rows and columns.
- **File path:** A file's location on your computer or the internet. File paths can be relative or absolute.
- **R Projects:** Set the working directory to the folder containing the `.Rproj` file.
- **Reading in data manually:** Download the data and put it in the project folder. In RStudio, go to File > Import Dataset > From Text (`readr`), browse for the file, then click "Update" and "Import."

Functions

Library/Package	Piece of code	Example of usage	What it does
`tidyverse` (`readr`)	`read_csv()`	`df <- read_csv("<url>")`	Reads in a CSV file from a specified file path.
`tidyverse` (`readr`)	`read_tsv()`	`df <- read_tsv("<url>")`	Reads in a TSV file from a specified file path.
`tidyverse` (`readr`)	`read_delim()`	`df <- read_delim("<url>", delim = ":")`	Reads in a delimited file from a specified file path.
Base R	`head()` / `tail()`	`head(df)` / `tail(df)`	Returns the first or last part of a dataset or other object.
Base R	`getwd()`	`getwd()`	Returns the current working directory.
Base R	`setwd()`	`setwd("Desktop")`	Changes the current working directory.
Base R	`View()`	`View(df)`	Opens a spreadsheet-style data viewer for a matrix-like R object.
Base R	`str()`	`str(df)`	Displays the internal structure of an R object, including dimensions and class.

Subsetting Data in R

Functions

Library/Package	Code	Example	What it does
Base R	`nrow()`; `ncol()`	`nrow(x)`; `ncol(x)`	Get the number of rows and columns in object `x`, respectively.
Base R	`dim()`	`dim(x)`	Get the number of rows and columns in object `x`.
`tidyverse` (`dplyr`)	`glimpse()`	`glimpse(mtcars)`	Get an overview of data frame `x`.
`tidyverse` (`dplyr`)	`slice_sample()`	`slice_sample(mtcars)`	View a random subset of rows from `x`.
Base R	`data.frame()`	`df <- data.frame(1:3)`	Create a data frame. Named arguments must have the same length.
`tidyverse` (`tibble`)	`tibble()`	`tibble(mtcars)`	Create a tibble from a data frame or matrix.
`tidyverse` (`tibble`)	`rownames_to_column()`	`df <- rownames_to_column(df, "new_col")`	Convert row names to a new first column. The argument names the new column.
Base R	`colnames()`	`colnames(df)`	Get or set column names for a matrix or data frame.
`tidyverse` (`dplyr`)	`rename()`	`df <- rename(df, MPG = mpg)`	Rename selected columns while keeping all variables.
`tidyverse` (`dplyr`)	`rename_with()`	`df <- rename_with(df, toupper)`	Rename selected columns with a function.
Base R	`toupper()`	`toupper("text")`	Convert lowercase characters to uppercase.
Base R	`tolower()`	`tolower("TEXT")`	Convert uppercase characters to lowercase.
`janitor`	`clean_names()`	`clean_names(df)`	Return a data frame with clean names.
`tidyverse` (`dplyr`)	`pull()`	`pull(df, "existing_variable_name")`	Extract a column as a vector. Base R notation: `df$column`.
`tidyverse` (`dplyr`)	`select()`	`select(df, "existing_variable_name")`	Select columns that match the specified argument. (*See note*)
`tidyverse` (`dplyr`)	`filter()`	`filter(df, mpg > 20)`	Return rows that match the specified logical condition.

Library/Package	Code	Example	What it does
Base R	`==, <=, >=, !=`	`filter(df, mpg > 20)`	Compare values in an object. Useful with `filter()`.
Base R	`%in%`	`filter(df, mpg %in% c(20, 21, 22))`	Check whether values on the left are in the object on the right. Returns `TRUE` or `FALSE`.
Base R	`|>`	`df <- df |> select("variable_name")`	Pipe a data frame through additional operations.
`tidyverse` (`magrittr`)	`%>%`	`df <- df %>% select("variable_name")`	Pipe a data frame through tidyverse operations.
`tidyverse` (`dplyr`)	`mutate()`	`df <- mutate(df, newcol = wt / 2.2)`	Add a new column based on existing columns.
`tidyverse` (`dplyr`)	`relocate()`	`df_carb <- relocate(.data = df, wt, .before = mpg)`	Reorder columns in a data frame or tibble.
`tidyverse` (`dplyr`)	`arrange()`	`df <- arrange(df, mpg)`	Reorder rows in ascending order. Use `arrange(desc())` for descending order.

NOTE: See tidyselect helpers (`?dplyr_tidy_select`) for handy logical functions to use with `select()`.

Data Summarization

Functions

Library/Package	Piece of code	Example of usage	What it does
Base R	`min()`	`min(x)`	Returns the minimum value in object `x`.
Base R	`sum()`	`sum(x)`	Returns the sum of values in object `x` (integer, numeric, or logical).
Base R	`mean()`	`mean(x)`	Returns the arithmetic mean of values in object `x` (integer, numeric, or logical).
Base R	`log()`	`log(x)`	Returns the natural logarithm of object `x`. Use `log2(x)` for base 2, or specify another base.
Base R	`range()`	`range(x)`	Returns the minimum and maximum values in object `x`.
Base R	`sd()`	`sd(x)`	Returns the standard deviation of object `x`.
Base R	`sqrt()`	`sqrt(x)`	Returns the square root of object `x`.
Base R	`quantile()`	`quantile(x, probs = .5)`	Returns sample quantiles for specified probabilities.
Base R	`summary()`	`summary(x)`	Returns a summary of values in object `x`.
`tidyverse` (`dplyr`)	`pull()`	`x_vect <- df |> pull(x)`	Extracts a single column as a vector. Useful before summary functions like `mean()` and `sum()`.
`tidyverse` (`dplyr`)	`summarize()`	`df <- df |> summarize(mean_x = mean(x))`	Summarizes values into one or more output values. `summarize()` and `summarise()` are synonyms.
`tidyverse` (`dplyr`)	`distinct()`	`df |> distinct(factor_name)`	Displays unique rows from a data frame or tibble.
`tidyverse` (`dplyr`)	`n_distinct()`	`x_vect |> n_distinct()`	Counts unique values or combinations in one or more vectors.
`tidyverse` (`dplyr`)	`count()`	`df |> count(factor_name)`	Counts the number of rows in each group of a factor variable.
`tidyverse` (`dplyr`)	`group_by()`	`df |> group_by(factor_name)`	Groups rows by specified value(s).
Base R	`unique()`	`unique(df)`	Returns object `x` with duplicate elements or rows removed.

Library/Package	Piece of code	Example of usage	What it does
Base R	`rowSums()`	`rowSums(df)`	Calculates sums for each row.
Base R	`colSums()`	`colSums(df)`	Calculates sums for each column.
Base R	`rowMeans()`	`rowMeans(df)`	Calculates means for each row.
Base R	`colMeans()`	`colMeans(df)`	Calculates means for each column.

NOTE: Many summarizing functions (e.g., `mean()`, `sum()`) have the argument `na.rm = TRUE` to ignore missing data.

Data Classes

Major concepts

- **Character:** Strings or individual characters, quoted.
- **Numeric:** Any real number(s).
- **Double:** A subset of numeric values that can contain decimals.
- **Integer:** Whole number(s).
- **Factor:** Categorical or qualitative variables.
- **Logical:** Values composed of `TRUE` or `FALSE`.
- **Date/POSIXct:** Calendar dates and times.
- **Matrix:** Two-dimensional data where all rows and columns have the same data type.
- **Data frame:** Two-dimensional data where columns can have different data types.
- **List:** Data that can vary in dimensions and contain many data types, including vectors, strings, matrices, models, and other lists.

Functions

Library/Package	Piece of code	Example of usage	What it does
Base R	`class()`	`class(x)`	Returns the class of an object.
Base R	`as.numeric()`	`as.numeric(x)`	Coerces object `x` to numeric class. Similar functions include `as.character()`, `as.data.frame()`, `as.matrix()`, and `as.Date()`. Use with `mutate()`.
`tidyverse` (`lubridate`)	`ymd()`	`ymd("2024-01-31")`	Coerces character object `x` to date class. Use functions such as `mdy()` or `dmy()` for other date formats.

NOTE: `lubridate` is a powerful, widely used R package from `tidyverse` family to work with Date / POSIXct class objects.

Data Cleaning

Major concepts

- **Most important rule of data handling:** Always look at your data.
- **`NA`:** General missing data.
- **`NaN`:** "Not a Number"; occurs when you calculate `0/0`.
- **`Inf` and `-Inf`:** Infinity; occurs when you divide a positive or negative number by `0`.
- **`naniar`:** Useful for exploring missingness in large datasets. See the `naniar` package.
- **`tidyverse (stringr)`:** Useful for detecting and working with parts of character values, especially with `filter()` and `str_detect()`.

Examples

Example of `case_when()` with `mutate()`:

```
Orange |>
  mutate(old = case_when(
    age > 1000 ~ TRUE,
    .default = FALSE
  ))
```

Functions

Library/Package	Piece of code	Example of usage	What it does
Base R	`is.na()`	`is.na(x)` `any(is.na(x))`	Checks if `x` is `NA`. With `any()`, checks if any values are `NA`.
Base R	`is.nan()`	`is.nan(x)`	Checks if `x` is `NaN`.
Base R	`is.infinite()`	`is.infinite(x)`	Checks if `x` is `Inf` or `-Inf`.
`naniar`	`pct_complete()`	`pct_complete(x)`	Reports the percentage of complete data in `x`.
`naniar`	`gg_miss_var()`	`gg_miss_var(x)`	Plots the percentage of missing data by variable in `x`.
`naniar`	`miss_var_summary()`	`miss_var_summary(x)`	Reports the percentage of missing data by variable in `x`.
`naniar`	`miss_var_which()`	`miss_var_which(x)`	Returns variable names that contain missing values in `x`.
`tidyverse` (`tidyr`)	`drop_na(df)`	`drop_na(df)` `drop_na(variable)`	Drops rows with `NA` values. When a variable is specified, only drops rows where that variable is `NA`.

Library/Package	Piece of code	Example of usage	What it does
tidyverse (dplyr)	na_if()	na_if(vector, value)	Converts specified values to NA. Use with mutate().
tidyverse (tidyr)	replace_na()	replace_na(vector, value)	Replaces NA values with a specified value. Use with mutate().
tidyverse (dplyr)	case_when()	case_when(test ~ outcome, .default = x)	Vectorizes multiple if_else() statements. If no cases match, NA is returned unless .default is specified. Use with mutate().
tidyverse (dplyr)	mutate()	df <- mutate(df, newcol = wt / 2.2)	Adds a new column based on existing columns.
tidyverse (tidyr)	separate()	df |> separate(x, c("A", "B"))	Separates one character column into multiple columns using a regular expression or numeric locations.
tidyverse (tidyr)	unite()	df |> unite("z", x:y, remove = FALSE)	Unites multiple columns into one column.
tidyverse (stringr)	str_detect()	df |> filter(str_detect(col_name, "string_pattern"))	Returns a logical vector indicating whether a string pattern was detected.
tidyverse (stringr)	str_replace()	str_replace(string = vector, "replace_me", "with_me")	Replaces the first instance of one specified string with another.
tidyverse (stringr)	str_replace_all()	str_replace_all(string = vector, "replace_me", "with_me")	Replaces all instances of one specified string with another.
tidyverse (stringr)	str_sub()	str_sub(string = vector, start = 1, end = 3)	Subsets a string to the specified character positions.

Data Manipulation

Major concepts

- **Wide data:** Multiple columns per individual, with values spread across columns
- **Long data:** Multiple rows per observation, with values stored in a single column

Functions

Library/Package	Piece of code	Example of usage	What it does
`tidyverse (tidyr)`	`separate()`	`df |> separate(x, c("A", "B"))`	Separates a character column into multiple columns using a regular expression or numeric positions
`tidyverse (tidyr)`	`unite()`	`df |> unite("z", x:y, remove = FALSE)`	Combines multiple columns into one column
`tidyverse (tidyr)`	`pivot_longer()`	`df |> pivot_longer(!col_to_keep, names_to = "new_col_with_labels", values_to = "new_col_with_values")`	Lengthens a data frame by increasing rows and decreasing columns
`tidyverse (tidyr)`	`pivot_wider()`	`df |> pivot_wider(names_from = "col_with_names", values_from = "col_with_values")`	Widens a data frame by decreasing rows and increasing columns
`tidyverse (dplyr)`	`?_join()`	`inner_join(x, y)`	Joins data from two data frames. `inner_join(x, y)` Keeps only rows that match in both `x` and `y`. `full_join(x, y)` Keeps all rows from `x` and `y`. `left_join(x, y)` Keeps all rows from `x`, even if they do not match `y`. `left_join(x, y)` Keeps all rows from `y`, even if they do not match `x`. `anti_join(x, y)` Keeps rows from `x` that do not match `y`, retaining only columns from `x`.

Data Visualization with `esquisse`

Major concepts

- **`esquisse`:** Package for testing and learning the syntax for making plots

Functions

Library/Package	Piece of code	Example of usage	What it does
`esquisse`	`esquisser()`	`esquisser(Orange)` `esquisser(Orange, viewer = "browser")`	Starts an interactive `esquisse` session to create a plot from a `data.frame` or `tibble`

Data Visualization

Major concepts

- **`tidyverse (ggplot2):`** Produces graphics.
- **Tidy data:** Each variable is a column, and each observation is a row.
  - Having data in tidy format and long format makes creating plots easier!
- **Plot layers:** Use `+` at the end of each line when adding layers to plots.
- **Pipes:** `|>` and `%>%` do not add plot layers, but they can pipe data into plots.
  - **Works:** `ggplot(data = iris, aes(x = Species, y = Petal.Length)) + geom_boxplot()`
  - **Works:** `iris |> ggplot(aes(x = Species, y = Petal.Length)) + geom_boxplot()`
  - **Does not work:** `iris |> ggplot(aes(x = Species, y = Petal.Length)) |> geom_plot()`

Functions

Library/Package	Piece of code	Example of usage	What it does
`tidyverse` (`ggplot2`)	`ggplot()`	`ggplot(data = iris)`	Begins a plot that is finished by adding layers.
`tidyverse` (`ggplot2`)	`aes()`	`ggplot(data = iris, aes(x = Species, y = Petal.Length))`	Maps variables to visual properties in a `ggplot()` object.
`tidyverse` (`ggplot2`)	`geom_boxplot()`	`ggplot(data = iris, aes(x = Species, y = Petal.Length)) + geom_boxplot()`	Creates a boxplot when added as a layer to a `ggplot()` object.
`tidyverse` (`ggplot2`)	`geom_density()`	`ggplot(data = iris, aes(x = Petal.Length)) + geom_density()`	Creates a smoothed density plot when added as a layer to a `ggplot()` object.
`tidyverse` (`ggplot2`)	`geom_point()`	`ggplot(data = iris, aes(x = Species, y = Petal.Length)) + geom_point()`	Creates a scatterplot when added as a layer to a `ggplot()` object.
`tidyverse` (`ggplot2`)	`geom_line()`	`ggplot(data = iris, aes(x = Species, y = Petal.Length)) + geom_line()`	Creates a line plot by connecting points in order of the x-axis variable.
`tidyverse` (`ggplot2`)	`geom_hline()`	`ggplot(data = iris, aes(x = Species, y = Petal.Length)) + geom_point() + geom_hline(yintercept = 4)`	Adds horizontal lines to a plot.

Library/Package	Piece of code	Example of usage	What it does
`tidyverse` (`ggplot2`)	`theme_classic()`	`ggplot(data = iris, aes(x = Petal.Length)) + geom_density() + theme_classic()`	Displays a `ggplot()` object without grid lines. See the `ggtheme` documentation for additional themes.
`tidyverse` (`ggplot2`)	`xlab()`; `ylab()`; `ggtitle()`	`ggplot(data = iris, aes(x = Petal.Length)) + geom_density() + xlab("Petal Length")`	Modifies the x-axis label, y-axis label, and plot title, respectively.
`tidyverse` (`ggplot2`)	`facet_grid()`	`ggplot(data = iris, aes(x = Petal.Length)) + geom_density() + facet_grid(~Species)`	Creates a grid of plots using specified variables to subset the data.
`tidyverse` (`ggplot2`)	`facet_wrap()`	`ggplot(data = iris, aes(x = Petal.Length)) + geom_density() + facet_wrap(~Species, scales = "free", nrow = 2)`	Creates individual plots using specified variables to subset the data, with flexible layouts and axis scales.
`tidyverse` (`ggplot2`)	`ggsave()`	`ggsave(filename = "plotname.pdf")`	Saves the last plot in the working directory.

NOTE: Click here for a summary of the `ggplot2` theme system.

Factors

Major concepts

- **Factor:** A special character vector whose elements have predefined groups, or levels. Factors are useful for qualitative or categorical variables, such as grade levels. Specify factors with the `factor` function so R recognizes them correctly.

Functions

Library/Package	Piece of code	Example of usage	What it does
Base R	`factor()`	`fact_qual <- factor(c("poor", "fine", "good"))`	Creates a factor from a vector. Levels are ordered alphanumerically by default.
Base R	`levels()`	`levels(fact_qual)`	Shows the levels and order of a factor vector.
`tidyverse` (`forcats`)	`as_factor()`	`fact_qual <- as_factor(c("poor", "fine", "good"))`	Creates a factor from a vector. Levels are ordered by first appearance in the data by default.
`tidyverse` (`forcats`)	`fct_reorder()`	`ggplot(iris, aes(x = fct_reorder(Species, Sepal.Width, mean), y = Sepal.Width)) + geom_boxplot()`	Reorders factor levels according to another variable. Here, `Species` is ordered by the mean of `Sepal.Width`.

Statistics

Functions

Library/Package	Piece of code	Example of usage	What it does
Base R	`cor()`	`cor(x, y)`	Calculates correlation between two vectors.
`corrplot`	`corrplot()`	`corrplot(cor_mat, type = "upper", order = "hclust")`	Creates a correlation matrix plot.
Base R	`t.test()`	`t.test(x, y, alternative = "two.sided")`	Performs one- and two-sided t-tests.
`broom`	`tidy()`	`tidy(t_test_result)`	Converts statistical objects to tidy data frames.
Base R	`wilcox.test()`	`wilcox.test(x, y)`	Performs nonparametric Wilcoxon signed-rank or rank-sum tests.
Base R	`shapiro.test()`	`shapiro.test(x)`	Tests normality with the Shapiro-Wilk test.
Base R	`ks.test()`	`ks.test(x)`	Tests a distribution with the Kolmogorov-Smirnov test.
Base R	`var.test()`	`var.test(x, y)`	Compares two variances with Fisher's F-test.
Base R	`chisq.test()`	`chisq.test(x, y)`	Performs chi-squared contingency table and goodness-of-fit tests.
Base R	`summary()`	`summary(linear_model_result)`	Summarizes an object, including models and statistical test results.
Base R	`glm()`	`glm(x ~ y, data = df, family = binomial())`	Fits generalized linear models; specify distribution and link with `family`.
`epitools`	`oddsratio()`	`oddsratio(x, y)`	Calculates odds ratios.

Data Output

Functions

Library/Package	Piece of code	Example of usage	What it does
`tidyverse` (`readr`)	`write_csv()`	`write_csv(df, "file.csv")`	Writes a data frame to a CSV file.
`tidyverse` (`readr`)	`write_delim()`	`write_delim(df, "file.txt", delim = ":")`	Writes a data frame to a delimited file.
`tidyverse` (`readr`)	`write_rds()`	`write_rds(df, "file.rds")`	Saves a single R object to an RDS file.
`tidyverse` (`readr`)	`read_rds()`	`df <- read_rds("file.rds")`	Reads a single R object from an RDS file.
Base R	`save()`	`save(df1, df2, file = "file.RData")`	Saves one or more R objects to an RData file.
Base R	`load()`	`load("file.RData")`	Loads R objects from an RData file into the environment.

Functions

Major concepts

- **Create functions to:**
  - Do custom actions
  - Reduce repetitive code
  - Organize code into manageable chunks
  - Avoid running code unintentionally
  - Use names that make sense to you
- **Custom function syntax:**

```
function_name <- function(inputs) {
    # Document inputs here
    <function body>
    return(value)
}
```

Examples

Example of an **anonymous function** used with `across()`:

```
iris |>
  summarize(across(
    starts_with("Sepal"),
    \(x) mean(x, na.rm = TRUE)
  ))
```

Functions

Library/package	Piece of code	Example of usage	What it does
Base R	`function()` or `\()`	`div_100 <- function(x) x / 100` or `div_100 <- \(x) x / 100`	Creates a function; here, `x` is divided by 100.
Base R	`lapply()`	`lapply(some_list, a_function)`	Applies a function over a list or vector.
Base R	`sapply()`	`sapply(some_list, a_function)`	Applies a function over a list or vector; returns a matrix or vector by default.

Library/package	Piece of code	Example of usage	What it does

`tidyverse` (`dplyr`)	`across()`	`iris |>`
		` mutate(across(`
		`  contains("Sepal"), round`
		` ))`
		or
		`iris |>`
		` group_by(Species) |>`
		` summarise(across(`
		`  starts_with("Sepal"),`
		`  \(x) mean(x, na.rm = TRUE)`
		` ))`
			Applies a function across columns in a data frame.

`tidyverse` (`dplyr`)	`mutate_if()`	`iris |> mutate_if(is.numeric, round)`	Applies a function to selected columns in a data frame.

* This cheatsheet format was adapted from Alex's Lemonade Stand materials (source).

* Find more resources at https://daseh.org.