Code Handout

Allison Theobold

This document contains all of the functions that we have covered thus far in the course. It will be updated every week, after we've added new skills. Each function is presented alongside an example of how it is used.

All of the examples below are in the context of the Palmer Penguins, found here (link).

Packages

- `library()` – loads packages into your `R` session

```
library(tidyverse)
library(moderndive)
library(skimr)
library(broom)

library(palmerpenguins)
```

Inspecting Data

- `glimpse()` – shows a summary of the dataset, the number of rows and columns, variable names, and the first 10 entries of each variable

```
glimpse(penguins)
```

- `skim()` – shows a summary of the variables in the dataset, summary statistics for the quantitative variables, and the number of missing observations.

```
skim(penguins)
```

Working with Data

- `<-` – "assignment arrow", assigns a value (vector, dataframe, single value) to the name of a variable

```
penguins_2007 <- penguins %>%
  filter(year == 2007)
```

- `c()` – the "concatenate" function combines inputs to form a vector, the values have to be the same data type.

```
cat_variables <- c("Species", "Island", "Sex")
```

- `data.frame()` – creates a dataframe based on input variables
  – Values for each variable are specified as vectors (using `c()`)
  – Variables are separated by commas

```
simple <- data.frame(year = c(2000, 2001),
                     species = c("Adelie", "Gentoo"),
                     island = c("Torgersen", "Dream"))
```

Verbs of Data Wrangling

- `select()` – selects variables (columns) from a dataframe

```
penguins %>%
select(species)
```

- `filter()` – filters observations (rows) out of / into a dataframe, where the inputs (arguments) are the conditions to be satisfied in the data that are kept

```
## It's nice to have a new line for each condition, so your code is easier to read!
penguins %>%
filter(species == "Adelie",
       body_mass_g > 3000,
       year == 2008)
```

- `mutate()` – creates new variables or modifies existing variables

```
penguins %>%
  filter(is.na(bill_length_mm) != TRUE,
         is.na(bill_depth_mm) != TRUE) %>%
  mutate(body_mass_kg = body_mass_g / 1000)
```

- `group_by()` – groups the dataframe based on levels of a categorical variable, usually used alongside `summarize()`

```
penguins %>%
  group_by(island)
```

- `summarize()` – creates data summaries of variables in a dataframe, for grouped summaries use alongside `group_by()`

```
penguins %>%
  filter(is.na(body_mass_g) != TRUE) %>%
  group_by(island) %>%
  summarize(mean_mass = mean(body_mass_g))
```

- `arrange()` – orders a dataframe based on the values of a numerical variable, paired with `desc()` to order in descending order

```
penguins %>%
  filter(is.na(body_mass_g) != TRUE) %>%
  group_by(island) %>%
  summarize(mean_mass = mean(body_mass_g)) %>%
  arrange(desc(mean_mass))
```

- `%>%` – the "pipe" operator, joins sequences of data wrangling steps together, works with any function that has `data =` as the first argument

```
penguins %>%
  select(species, island, body_mass_g, sex, year) %>%
  filter(island ==   "Torgersen",
         is.na(body_mass_g) != TRUE) %>%
  group_by(species, year) %>%
  summarize(mean_mass = mean(body_mass_g),
            median_mass = median(body_mass_g),
            observations = n()) %>%
  arrange(desc(mean_mass))
```

Other Data Wrangling Tools

- `count()` – counts the number of observations (rows) of the different levels of a categorical variable

```
penguins %>%
count(species)
```

- `mean()` – finds the mean of a numerical variable, not resistant to `NA` values, so either filter out prior or use `na.omit = TRUE` argument
  - Other summary functions include:
    - `var()` – find the variance of a numerical variable
    - `sd()` – finds the standard deviation of a numerical variable
    - `IQR()` – find the innerquartile range (Q3 - Q1) of a numerical variable
    - `median()` – finds the median of a numerical variable

- `cor()` – finds the correlation between two numerical variables
  - Can remove the NA values from the variables by specifying that `cor()` should only `use` the "pairwise.complete.obs".

```
penguins %>%
  summarize(correlation = cor(bill_length_mm, bill_depth_mm,
                              use = "pairwise.complete.obs")
  )
```

- `get_correlation()` – finds the correlation between two numerical variables
  - Specified using a formula (`y ~ x`)
  - Can remove the NA values from the variables by specifying `na.rm = TRUE`

```
penguins %>%
  get_correlation(bill_length_mm ~ bill_depth_mm, na.rm = TRUE)
```

- `is.na()` – returns a vector of `TRUE` and `FALSE` values corresponding to whether a particular row of a variable was `NA` (missing)

```
penguins %>%
  mutate(missing_weight = is.na(body_mass_g))
```

- `drop.levels()` – drops the levels of a categorical variable that have no observations in them, useful to use after filtering out levels of a categorical variable, so the only levels that appear are the ones you wanted to keep

```
penguins %>%
  filter(species != "Adelie") %>%
  droplevels()
```

- `distinct()` – selects the unique values of a variable

```
penguins %>%
  distinct(species)
```

- `slice_sample()` – selects rows from the dataframe, based on the value of `n` specified

```
penguins %>%
  slice_sample(n = 10)
```

- `if_else()` – function which creates output based on a condition to be satisfied, the first argument is the logical test we wish to perform, the second argument is what we want output if the result of the logical test is `TRUE`, the third argument is what we want output if the result of the logical test is `FALSE`

```
penguins %>%
  filter(is.na(flipper_length_mm) != TRUE) %>%
  mutate(large_flipper =
           if_else(flipper_length_mm >= mean(flipper_length_mm),
                   "above average",
                   "below average")
         )
```

- `%in%` – the "inclusion operator", used to specify 2 or more levels of a categorical variable that you wish to keep in your data
  - The levels to be kept must be included in a vector (`c()`).

```
penguins %>%
  filter(species %in% c("Gentoo", "Chinstrap"))
```

- `as.factor()` – converts a variable from numerical or character into a factor, necessary if fitting a data model or producing a visualization with a numerical variable that you want treated as categorical
  - `as.character()` will also work!

```
penguins %>%
  mutate(year_chr = as.character(year),
         year_fct = as.factor(year))
```

Data Visualization

- `ggplot()` – a function to create the shell of a visualization, where specific variables are mapped to different aspects of the plot

```
penguins %>%
  ggplot(aes(x = bill_length_mm, y = bill_depth_mm, color = species))
```

- `aes()` – aesthetics that can be used when creating a `ggplot()`, where the aesthetics can either be hard coded (e.g. `color = "blue"`) or associated with a variable (e.g. `color = sex`).
  - The following are the aesthetic options for *most* plots:
    - `x`
    - `y`
    - `alpha` – changes transparency
    - `color` – produces colored outline
    - `fill` – fills with color
    - `group` – used with categorical variables, similar to color

- `+` – an important aspect creating a `ggplot()` is to note that the `geom_XXX()` function is separated from the `ggplot()` function with a plus sign, `+`.
  - `ggplot()` plots are constructed in series of layers, where the plus sign separates these layers.
  - Generally, the `+` sign can be thought of as the end of a line, so you should always hit enter/return after it. While it is not mandatory to move to the next line for each layer, doing so makes the code a lot easier to organize and read.

```
penguins %>%
  ggplot(aes(x = bill_length_mm, y = bill_depth_mm, color = species)) +
  geom_point()
```

- `geom_histogram( )` – adds a histogram to the plot, where the observations are binned into ranges of values and then frequencies of observations are plotted on the y-axis
  - You can specify the number of bins you want with the `bins` argument

```
penguins %>%
  ggplot(aes(x = bill_length_mm)) +
  geom_histogram(bins = 20)
```

- `geom_dotplot()` – plots each observation as a dot that's placed at it's appropriate value on the x axis, then stacked as other cases take similar values
  - You can use the `dotsize` argument to decrease the size of the dots (1 is the default value).

```
penguins %>%
  ggplot(aes(x = bill_length_mm)) +
  geom_dotplot(dotsize = 1)
```

- `geom_boxplot( )` – adds a boxplot to the plot, where observations are aggregated (summarized), the min, Q1, median, Q3, and maximum are plotted as the box and whiskers, and "outliers" are plotted as points.
  - You can plot a vertical boxplot by specifying the `x` variable, or a horizontal boxplot by specifying the `y` variable.
  - Note: the min and max may not be included in the whiskers, if they are deemed to be "outliers" based on the 1.5 × IQR rule.

```
penguins %>%
  ggplot(aes(x = bill_length_mm)) +
  geom_boxplot()
```

- `geom_density()` – adds a density curve to the plot, where the probability density is plotted on the y-axis (so the density curve has a total area of one).
  - By default this creates a density curve without shading. By specifying a color in the `fill` argument, the density curve is shaded.

```
penguins %>%
  ggplot(aes(x = bill_length_mm)) +
  geom_density(fill = "tomato")
```

- `geom_smooth()` – plots a line over a set of points, draws the readers eye to a specific trend
  - The methods we will use are "lm" for a linear model (straight line), and "loess" for a wiggly line
  - By default, the smoother gives you gray SE bars, to remove these add `se = FALSE`

```
penguins %>%
  ggplot(aes(x = bill_length_mm, y = bill_depth_mm, color = species)) +
  geom_point() +
  geom_smooth(method = "lm")
```

- `geom_parallel_slopes()` – fits lines for each group of a categorical variable, but forces the lines to have parallel slopes

```
penguins %>%
  ggplot(aes(x = bill_length_mm, y = bill_depth_mm, color = species)) +
  geom_point() +
  geom_parallel_slopes()
```

- `labs()` – specifies the plot labels, possible labels are: x, y, color, fill, title, and subtitle

```
penguins %>%
  ggplot(aes(x = bill_length_mm, y = bill_depth_mm, color = species)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(x = "Bill Length (mm)",
       y = "Bill Depth (mm)",
       color = "Penguin Species")
```

- `xlim()` – specifies the limits of the x-axis, must be specified as a vector (`c()`)

```
penguins %>%
  ggplot(aes(x = bill_length_mm, y = bill_depth_mm, color = species)) +
  geom_point() +
  xlim(c(0, 70))
```

- `ylim()` – specifies the limits of the y-axis, must be specified as a vector

```
penguins %>%
  ggplot(aes(x = bill_length_mm, y = bill_depth_mm, color = species)) +
  geom_point() +
  ylim(c(0, 30))
```

- `facet_wrap()` – creates subplots of your original plot, based on the levels of the variable you input
  - To facet by one variable, use `~variable`.
  - To facet by two variables, use `variable1 ~ variable2`.
  - If you prefer for your facets to be organized in rows or columns, use the `nrow` and/or `ncol` arguments.

```
penguins %>%
  ggplot(aes(x = bill_length_mm, y = bill_depth_mm, color = species)) +
  geom_point() +
  geom_smooth(method = "lm") +
  facet_wrap(~island, nrow = 1)
```

Data Modeling

- `lm()` – fits a linear model to a dataset
  - You specify the variables as a formula (`y ~ x`), where `y` is your response variable and `x` is your explanatory variable
  - The second argument is the name of the dataset (`data = penguins`)

```
## Two quantitative explanatory variables
model1 <- lm(bill_length_mm ~ bill_depth_mm + body_mass_g, data = penguins)

## One quantitative and one categorical explanatory variable
model2 <- lm(bill_length_mm ~ bill_depth_mm + sex, data = penguins)
```

- `get_regression_table()` – produces a tidy table output of a regression model
  - Output includes coefficients, standard errors, p-values, and confidence intervals

```
get_regression_table(model1)
```

- `summary()` – produces a "raw" summary of a regression model
  - The "untidy" version of a regression summary.
  - Includes same information as `get_regression_table()`, but also includes R^2 and adjusted R^2.

```
summary(model2)
```

- `tidy()` – takes untidy output and creates a nice table!
  - Similar to `get_regression_table()`, but doesn't output confidence intervals.
  - Lives in the **broom** package

```r
tidy(model2)
```

- `get_regression_points()` – provides information on each observation used in a `lm()` in a tidy table format
  - Produces a table with the variables included in the regression, and the residual associated with each observation

```r
get_regression_points(model1)
```

- `predict()` – produces an untidy vector of the predicted y-values for each observation in the dataset
  - Can make predictions for new observations with the `newdata` argument.

```r
predict(model1)

new_penguin <- data.frame(bill_depth_mm = 200, body_mass_g = 500)
predict(model1, newdata = new_penguin)
```

- `augment()` – produces a tidy table of data values from a regression model
  - Lives in the **broom** package
  - Produces a table with the variables included in the regression, and 6 additional columns:
    - including `.fitted` (predicted y-value for that observation), `.resid` (residual for that observation)
  - Can make predictions for new observations with the `newdata` argument.

```r
augment(model2)

new_penguin <- data.frame(bill_depth_mm = 15, sex = "female")
augment(model2, newdata = new_penguin)
```