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(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, frequencies for the categorical 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")
```

Verbs of Data Wrangling

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

```
penguins %>%
  select(species, island, bill_length_mm)
```

- `filter()` – filters observations (rows) out of / into a dataframe
  - 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 %>%
  mutate(body_mass_kg = body_mass_g / 1000,
         class = if_else(body_mass_kg > 27, "large",
                         "not large")
         )
```

- `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 after a `group_by()` step!

```
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
  - By default arranges the variable in ascending order
  - To arrange in descending order, must be paired with `desc()`

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

# 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)
```

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 missing observations out prior or use the built-in `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

- `get_correlation()` – finds the correlation between two numerical variables
  - Variables are specified with "formula" syntax (e.g., `x ~ y`)
  - Not resistant to `NA` values, so either filter missing observations out prior or use the built-in `na.omit = TRUE` argument

```
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)) %>%
  select(body_mass_g, missing_weight)
```

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

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

- `if_else()` – function which creates output based on a condition to be satisfied
  - The first argument is the logical comparison / test we wish to perform
  - The second argument is what you want output if the result of the logical test are `TRUE`
  - The third argument is what you want output if the result of the logical test are `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 two 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
  - This is necessary if fitting a data model or producing a visualization with a numerical variable that you want treated as categorical (e.g., `year`)
  - `as.character()` will also work!

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

Data Visualization

- `ggplot()` – a function to create the blank canvas for a visualization
  - Maps specific variables to different aspects of the plot (e.g., x-axis, y-axis, color, fill)

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

- `aes()` – aesthetics that can be used when creating a `ggplot()`
  - The aesthetics are **always** associated with a variable (e.g. `color = sex`).
  - The following are the aesthetic options for *most* plots:
    - `x`
    - `y`
    - `color` – produces colored outline (good for points)
    - `fill` – fills with color (good for barplots and boxplots)

- `+` – separates "layers" of a `ggplot()`
  - The `ggplot()` function is separated from the `geom_XXX()` function with a plus sign because they are different layers of a plot
  - The `+` sign can be thought of as the end of a layer, so you should always hit enter/return after it.

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

One Variable Plots

Histograms

- `geom_histogram( )` – adds a histogram to the plot
  - Observations are binned into ranges of values and then frequencies of observations are plotted on the y-axis
  - You can specify the width of the bins you want with the `binwidth` argument (e.g., `binwidth = 2`) – **This is my preference**
  - Or you can specify the number of bins you want with the `bins` argument (e.g., `bins = 20`)

```
penguins %>%
  ggplot(aes(x = bill_length_mm)) +
  geom_histogram(binwidth = 2)
```

Dotplots

- `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)
```

Boxplots

- `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 $1.5 \times$ IQR rule.

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

Density Plots

- `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")
```

Two Variable Plots

Scatterplots

- `geom_point()` – adds points to a plot
  - can change the transparency of the point with `alpha` numbers closer to 0 are more transparent (e.g., `alpha = 0.2`)

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

- `geom_smooth()` – plots a line over a set of points
  - When used with a `color` variable, produces different lines for different groups (slopes and intercepts are different)
  - The methods we will use are "lm" for a linear model (straight line)
  - By default, the smoother gives you gray SE bars, to remove use the `se = FALSE` argument for the `geom_smooth()` function

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

- `geom_parallel_slopes()` – fits different 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()
```

Additional Plotting Options

- `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")
```

- `facet_wrap()` – creates subplots of your original plot, based on the levels of the variable you input
  - To facet by one variable, use `~ variable`.
  - 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)
```

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

```
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.

```
predict(model1)

new_penguin <- data.frame(bill_depth_mm = 200, body_mass_g = 500)

predict(model1, newdata = new_penguin)
```