PROGRAMMING EXERCISES

EXERCISE 1: MAKEFILES

Create some scripts and a makefile to run them

- Write two notebooks: `analysis.ipynb` and `plotting.ipynb`
- Mirror the outputs to `.R` scripts using `jupytext`
- `analysis.ipynb` should load up `palmerpenguins` and save it as a `.csv`
- `plotting.ipynb` should read in the `.csv`, make a plot, save it as a `.pdf`
- Create a `makefile` with three options: `analyze`, `plot`, and `all` to run the scripts, respectively.

EXERCISE 2: REFACTORING AS FUNCTIONS

Write a script to fit a KNN regression to predict one variable from another using `palmerpenguins` with some tuning parameter. Refactor the code to take the tuning parmaeter as an agument.

For example:

- You can use `knnreg` from the `caret` package
- Refactor this routine as a function with a single argument `K`, the number of neighbors, and return the KNN model from the function
- Apply this function over the sequence `K=1,5,10,15` and put the output in a `list`

EXERCISE 3: MAGIC NUMBERS

Write a function to generate data from the model

$$y = \beta_0 + \beta x + e$$

where $x \sim U(0, 1)$ and $e \sim N(0, \sigma^2)$. After generating the data, write a function to fit a regression (KNN regression?). Return the model.

- The arguments of this function should allow me to change: β_0,β, and any tuning parameter of your model.
- Run this simulation over the cases of $\beta_0 = 0, \beta = 1, \sigma^2 = 1$ and,
- $\beta_0 = 3, \beta = 5, \sigma^2 = 3$, along with some other combinations of your tuning parmeters (values of K maybe).
- Keep the respective outputs as lists.

EXERCISE 4: CACHING

Cache and read-in analysis.

- Using previous analysis, save the output to a `.RDS` file using `saveRDS`.
- Start a new notebook/session, read in the cached output using `readRDS`
- Do this over, but now use our `read_or_run` function in both places.

EXERCISE 5: RANDOM NUMBERS

Write a function to estimate the average value of $\log(X)$ where X is uniform over 0 to 1.

- Set the seed for this simulation and check that it reproduces the same result.
- Use the `future.apply` function/package to run this simulation 10 times in parallel. Check that it is reproducible.

EXERCISE 6: PUTTING IT ALL TOGETHER.

1. Open up the messy code messy.ipynb
2. Refactor this code so that it is more *proactively reproducible*. Remember the five idioms and try to incorporate them into your solution:
   - write it in code, not the console
   - don’t repeat yourself, use functions
   - avoid magic numbers, expose them
   - cache intermediate results
   - seed random numbers
3. Use an automatic linter like `styler` to style the code