## Classification

Start by piecing together the code cells from the slides that use `tidymodels` to predict the `species` (Gentoo or Adelie) of penguins using the value of `body_mass_g`. Put them into an .r script or Quarto document.

The following exercises have you investigate the different components of the code and make variations.

1. To build an understanding of the four metrics used to evaluate model performance, write four `dplyr` (`filter()`, `summarize()`, `mutate()`, etc.) pipelines to calculate each metric manually from `ad_gen_preds`. Alongside each of the pipelines, write the English interpretation of each (e.g. "the proportion of ___ penguins that the model predicted to be ____"). Check your results against the values in `all_metrics`.

2. Using the model that performed the *worst*, build a plot to understand where it made the errors. This should be a scatterplot with the predictor along the x-axis, the species along the y-axis, and points colored by whether it is a true positive, true negative, false positive, or false negative.

3. Add two additional predictors to the recipe: `island` and `bill_length_mm`. Also add a third step: `step_dummy(all_nominal_predictors())`. What are the different values that `island` can take?

4. Repeat the prep and bake phases as in the last problem set to get a glimpse of what the training data would look like when all of the steps are applied. What happened to the `island` column?

5. Rerun your original three models with this modified recipe. How does the predictive performance of each model on the test set compare?

---

6. Install the `kernlab` package and read the help file for `data(spam)`. What is the natural response variable in this dataset? What do each of the predictors represent?

7. Using the `spam` dataset, create a classification model to predict whether an email is spam or not spam. You may use whichever predictors you like and any model class (with any parameters). Use a 70/30 train/test split and evaluate the model using the same four metrics as in the penguin example. What is the highest you were able to get your test accuracy?

## APIs I

Revisit the homepage for the BART Legacy API discussed in Lecture.

https://www.bart.gov/schedules/developers/api

8. What is the public key that can be used to access data through the legacy API?

---

Next visit the documention site: https://api.bart.gov/docs/overview/index.aspx. Note that each category of data has its own endpoint / base URL, and all takes a similar format (e.g. for estimated departure data use https://api.bart.gov/api/etd.aspx.)

9. Provide the URL that will query the API for a list of all BART stations in JSON format.

10. Provide the corresponding shell/terminal command.

11. Provide the corresponding R code that will perform the query and parse the JSON response into an R list object.

12. Subset the list to to return a data frame that contains the station abbreviation, name, latitude, longitude, address, city, county, state, and zipcode for each station. Which county has the greatest number of stations?

---

13. Provide the URL that will query the API for estimated departure times (ETDs) from the Downtown Berkeley Station in JSON format.

14. Provide the corresponding R code that will perform the query and parse the JSON response into an R list object.

15. Extract the elemented called `root` and then the `station` element within it. View the object using the data viewer. What type of object is it an what are it's dimensions?

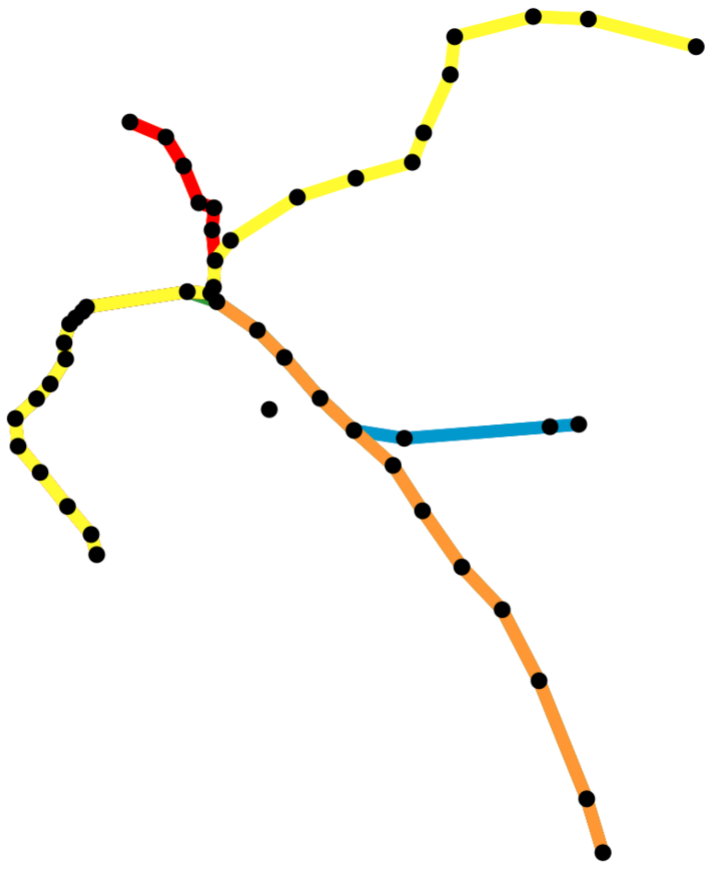
16. Use the `unnest()` function from the `tidyr` package to expand the `etd` list-column. What are the dimensions of the resulting data frame?

17. Use `unnest()` again to expand the `estimate` list-column. What are the dimensions of the resulting data frame?

18. How long (in minutes) is the shortest estimated wait time for a train departing from Downtown Berkeley Station? Which destination does this train go to? For reference, what time is it now?

19. Query the BSA (BART System Advisories) endpoint to retrieve the latest system advisory information. You're welcome to use any method: web browser, command line, or R. What are the different bits of information that they provide through this endpoint?

20. *Optional Challenge*: Use the data provided by the BART API to construct a map of the BART system showing the locations of all stations. You could make it interactive with leaflet or static like the one below.



## APIs II

Begin by copying the source for two endpoints into an R script called `stat133-api.R`: `/babynames` and `/grades`. Start your API server in your R session, then test it out by making requests using a web browser and navigating to `http://localhost:8080/babynames` and `http://localhost:8080/grades`.

21. Keeping track of the version of your API is very important. Other software systems may be built around your API, and your users need to access to a stable version. Consult to Plumber references (https://www.rplumber.io/articles/annotations.html) to learn: what is the tag that can be used to add a version to your API? Write it below then add it to your API.

22. Modify `babynames` to accept two more parameters: `sex`, that will return the babynames of the specified sex and `year`, that will return the babynames for the specified year. All three parameters should be optional. If no value is specified for a given parameter, the endpoint should return all babynames. For example:

- `http://localhost:8080/babynames?sex=F` should return all female babynames.
- `http://localhost:8080/babynames?year=2000` should return all babynames from the year 2000.
- `http://localhost:8080/babynames?sex=M&year=1990` should return all male babynames from the year 1990.

Provide your modified endpoint code below then test your endpoint using a web browser.

23. Create an additional endpoint called `/requestcontents` that is a POST request that does nothing but returns the contents of the requests as a JSON object. Write the code for this endpoint below.

24. Test your new endpoint using the `httr2` package by sending a POST request with a JSON body that contains the following data:[1]

```r
list([dish = "Turkey", quantity = 1, vegetarian = FALSE])
```

Provide the R code you used to send the request and the response you received below (formatted as an R list).

25. Modify the `/grades` endpoint to reflect two important components of our grading scheme that was omitted in the original implementation:
    - Each quiz number score is an average of an individual grade and a group grade.
    - Each quiz number score is evenly weighted when determining the quiz average.
    - The lowest quiz number score will be dropped before calculating the quiz average.

Write the code of the modified endpoint below. Also write out an R list containing a fictional students' full assignment grades, that will serve as the JSON payload when testing the endpoint.

[1]This is a good way to get a sense of what information is available in the request object when you write your endpoints.