Stats 701 Final Project

Josh Errickson

The final project is to build a fully-functioning R package with complete documentation and a full test suite. Below you will find two parts, the first consists of three options (implement at least one of these) and the second is an exploration of power and assumption violations.

If you are interested in building an R package for beyond this course and have an idea of what you want to do, come talk to me. We can decide on what a reasonable goal for this project would be and you can submit your progress on that package instead of the below.

Submission instructions

Due Thursday April 27th

- You may either put both parts into a single package, or create two separate packages.
- All exported functions should be well documented.
- All functions (exported or internal) should be well tested.
- Check for reasonable inputs and provide informative errors on bad input.
- All tests should pass, and all Checks should pass.
  - If you are unable to address all Checks, document the errors and your steps taken to address them and provide that as an additional attachment (pdf or html from Rmarkdown).
- There is no need to submit an Rmarkdown file otherwise.
- As with homework, provide proper citations if you get help on the code.
- Use the "Build Source Package" option in RStudio to create a .tar.gz zip file, which you should submit via Canvas.

Part 1: Build-an-API

For each of these, be sure to properly clean the output and return it in a nice format (probably a `data.frame` but I'll leave that up to you).

Implement at *least* one of these

1a: Baseball Data

http://www.baseball-reference.com/teams/ This site lists statistics for the current players on each team. Each team has a 2017 statistics page such as http://www.baseball-reference.com/teams/BAL/2017.shtml for the Baltimore Orioles, as well as a 2017 roster page like http://www.baseball-reference.com/teams/BAL/2017-roster.shtml.

Write a function that returns all current players. Write a second function (or multiple functions) that returns a given players Batting, Pitching, or Fielding statistics (returning an appropriate error if a player has no data in a given table, which is not the same as entering an invalid player!). You can either write a single function that takes an argument like `table = "pitching"`, or write three separate functions like `stats.pitching` etc.

If I were to write this, this is what my function calls would look like - *this is not the required interface, merely a suggestion.*

```
getPlayers(team = "Baltimore Orioles")
getStats(player = "Dylan Bundy",
         team = "Baltimore Orioles",
         table = "fielding")
```

You'll likely need a helper function to convert from the team name to its three letter abbreviation (e.g "Baltimore Orioles" to "BAL"). Use the "stringdist" package to try and match, to allow users to type partial names or make mistakes. (Not full clustering, but try the function `stringdist`).

You do *not* need to worry about inexact player names to your functions, as the function which returns the players names should allow the user to get that exactly right.

1b: Recipe Puppy

http://www.recipepuppy.com/about/api/ This API allows access to recipes based upon ingredients or a search query. Create a function which takes in a list of ingredients a user has (and an optional search term) and returns the first 100 recipes which include at least j of the k ingredients (users can input j), informing the user what additional ingredients they will need to purchase, and linking them to the recipe.

If I were to write this, this is what my function calls would look like - *this is not the required interface, merely a suggestion.*

```
getRecipes(ingredients = c("onion", "cheese", "garlic"),
           search = "burger",
           minNumIngredients = 2)
```

Notes:

- The API returns only 10 items per page, so you'll have to iterate over pages until you have enough or get no results.
- By "at least j of the k ingredients", I mean that if a user entered chicken, onions, garlic and salt and `j = 2`, then return 100 recipes which include at least 2 of those ingredients. You'll need to loop over the $\binom{4}{2}$ different combination of ingredients and make some decisions about how to combine those results and how to order them. (Perhaps order by total number of ingredients used? Or inform the user that order of ingredients implies importance, and order by that?)
- I have contacted the author of the API to let him know about the project (the author has a note asking to be informed if over 1,000 queries per day will take place) and he said it was no problem, but be a good data citizen and try to avoid massively spamming the API.

1c: Daily Time Of

https://sunrise-sunset.org/api and http://open-notify.org/Open-Notify-API/ISS-Pass-Times/. Build a function (or series of functions) that enable a user to enter a location, and returns the sunrise and sunset times as well as the time of the next pass of the international space station. The two APIs only take in latitude and longitude, so you'll need some way to convert location into lat/long. Here's a few suggestions, but feel free to find your own way:

- Use Google Maps Geocoding API available at https://developers.google.com/maps/documentation/geocoding/start. Note that you'll need to register for a (free) API key.
- Use the zipcode package which contains a `zipcode` dataset, including zip, city, state and lat/long.
- Use one of the APIs available here: https://github.com/toddmotto/public-apis#geocoding

Your function should allow users to enter a zip code or a city and state, or the latitude and longitude directly. If a user enters an older date, return the sunrise and sunset with a reasonable warning for why no ISS results are returned.

For the ISS passes, the function `as_datetime` in the "lubridate" package is useful for converting the numeric datetime into human-recognizable. You should report the time in the local timezone (e.g. a search on Los Angeles should return the time in the Pacific time), check the arguments of `as_datetime`. Again, a few suggestions for how to get the proper time zone:

- Google Maps Time Zone API! https://developers.google.com/maps/documentation/timezone/start
- Find an online database that contains both zip codes and time zone (e.g. https://boutell.com/zipcodes/, there are others if you search) and include it in your package so you can refer to it. (Give proper attribution!!!)

Part 2: Simulation

Create a function which takes in parameters as arguments (and number of iterations) and performs the simulation as described below. There should be sensible default arguments so that if the function is called without any arguments, it produces a result.

When performing linear regression, the assumption that the errors are normally distributed is not necessary for the estimated coefficients to be valid, but it can invalidate the standard errors (and hence, in turn, the confidence interval and p-value). Create a simulation which examines this in the face of the following parameters:

- n: The sample size.
- β: The strength of the data-generating coefficient for a single predictor of interest.
- ϵ: Details about the distribution of the errors.
  - Support at least "uniform", "t" and "laplace". (Note that Laplace distribution is "double" exponential.)
  - Pass additional arguments to the distributions via the `...` arguments to your function. (Recall my notes on using the `...`: http://dept.stat.lsa.umich.edu/~jerrick/courses/stat701/notes/functions.html#the-triple-dots)
  - Recall that errors should either be symmetric around 0, or the errors should be multiplied by `sample(c(-1, 1), n, replace = TRUE)` to mirror them (for non-symmetric distributions like F, if you chose to implement it).
  - The `switch` function may be useful to avoid too many `if ... else` statements.

For each set of input parameters, run the simulation twice: once with the provided distribution of ϵ, and once with ϵ being normally distributed with with the same (approximate) standard deviation from the non-normal distributions (either using the algebraic formula for it, or estimating from the data). Return the following:

- The distribution of $\hat{\beta}$.
- An estimate of the power of the test to detect the effect β.
- The distribution of R^2.

I'd suggest returning these result as a named list, but if you can come up with another way, go for it.

If I were to write this, this is what my function calls would look like - *this is not the required interface, merely a suggestion.*

```
regsim <- function(reps, n, beta, p, eps.dist, ...) { }
regsim(reps = 10000,
       n = 100,
       beta = 1.2,
       eps.dist = "uniform",
       min = -1,
       max = 1)
```

In addition, write a function which takes in a vector of values for n and β (the ϵ arguments need not be vectorized, but feel free to if you're feeling ambitious). It should loop over those vector, producing plots of

the chosen parameters versus $\hat{\beta}$, power and R^2. You may have an additional argument that specifies which of those three plots to produce, or be clever and produce all three, similar to how plotting a `lm` result works (or be even more clever and return a list of `ggplot` objects).

If I were to write this, this is what my function calls would look like - *this is not the required interface, merely a suggestion.*

```
plotRegsim(reps = 1000,
           n = c(50, 100, 500),
           beta = c(.4, .8, 1, 1.5),
           eps.dist = "uniform",
           min = -1,
           max = 1,
           which.plot = "r2")
```