Project Report Assignment 3 Group 2

Name: ArviZ

URL: https://github.com/arviz-devs/arviz

ArviZ is a Python script made for exploratory data analysis of Bayesian models.

Onboarding experience

Did it build and run as documented?
We first went with PyTensor but there were a lot of complications. The biggest being that it was a massive project with an enormous amount of tests that took over an hour to compile. We regrettably took a couple days before we switched repositories but we are happy that we did it at all. We briefly flirted with Evennia but it was really confusing and had its own test environment so we decided to go with something else.

As for ArviZ it was for the most part pretty simple. We had a few troubles with a couple dependencies but things worked out pretty smoothly. That inspired a lot of confidence in us so we ended up sticking with ArviZ.

Complexity

1. What are your results for five complex functions?
  ** Did all methods (tools vs. manual count) get the same result?*
  ** Are the results clear?*
See table on the next page.

2. Are the functions just complex, or also long?
They were pretty long as well, not just complex. It's not a perfect 1-to-1 match but there's definitely some degree of correlation.

3. What is the purpose of the functions?
They are functions within the program for different types of plots used for different kinds of data and statistical analysis.

4. Are exceptions taken into account in the given measurements?
We did not take exceptions into consideration in our calculations which can be the reason why our results differed from the CC Lizard calculated.

5. Is the documentation clear w.r.t. all the possible outcomes?

Function	LOC	Joline (CC)	Roger (CC)	Jacob (CC)	Victoria (CC)	Lizard (CC)
plots/ecdfplot.py	337	26 + P	26 + P	25 + P	25 + P	27
plots/essplot.py	295	17 + P	17 + P	17 + P	17 + P	15
plots/backends/matplotlib/pairplot.py	291	63 + P	63 + P	63 + P	63 + P	56
plots/bpvplot.py	278	26 + P	26 + P	26 + P	26 + P	23

Refactoring

Plan for refactoring complex code:
We can for larger codes during certain branching operations swap out the code blocks for some sort of function that instead does the same job as the code block. For example there is one massive block behind an if-statement in plots/backends/matplotlib/pairplot.py line 123-244 that we could maybe run in a separate function.

Estimated impact of refactoring (lower CC, but other drawbacks?).
In some cases it might make the code harder to trace since you need to look at different files. Also it may lead to more shared variables across different files which can also be annoying to track.

Coverage

Tools

We used coverage.py which was a well documented and easy to use tool. The tool showed the test coverage in an easy and readable way.

Your own coverage tool

We implemented all our own coverage tools in our own separated branches. But what we had in common was that we all by hand commented on the different branches and gave it an ID. Then we had a list with the length of the number of branches. Then we saved it in a text file. Here some wrote a script to calculate the coverage, while others did that by hand. We implemented our coverage tools in the same functions as we did the complexity calculations.

Evaluation

1. How detailed is your coverage measurement?
We measure the percent of covered branches. So if there are two branches and only one is reached then the coverage is 50 percent.

2. What are the limitations of your own tool?
That we have to manually go in and count how many branches there are. So every time a new branch is added to the code, that has to be changed as well.

3. Are the results of your tool consistent with existing coverage tools?
Not exactly, since other tools take the number of lines into consideration as well.

Coverage improvement

Report of old coverage: https://github.com/Rocygel/DD2480-3-arviz/tree/Original-branch

Report of new coverage: https://github.com/Rocygel/DD2480-3-arviz

Test cases added:
Roger: test_plot_seperation_idata_none_error() and test_plot_separation_idata_type_error() under arviz/tests/base_tests/test_plot_matplotlib.py for the arviz/plots/separationplot.py previous coverage 74% now 79%.

This is not the same as the method one I did measure on because the complexity made it very rough to digest the code in a short time and test the few hard-to-reach branches not already tested. They test a few error branches, specifically situations where the *idata* variable's predicate can cause errors.

Jacob: Added the test "test_plot_bpv_invalid" under arviz/tests/base_tests/test_plot_bokeh.py for the arviz/plots/bpvplot.py. Also improved the existing function "test_plot_bpv:" so more branches will be covered.

The function "test_plot_bpv_invalid" tests invalid inputs so it covers the previous uncovered branches of error raising. In the test_plot_bpv " {"kind": "t_stat", "t_stat": 0.5, "bpv": True, "flatten_pp": None, "flatten": ["dim1", "dim2"]}, " was added to the kwargs so it would reach a further branch. These changes increased the coverage from 83% to 89%.

Joline: added tests "test_plot_hdi_dimension_mismatch():", "test_plot_hdi_credible_interval_warning()" for function plot_hdi in file hdiplot.py as the ecdfplots was hard to implement more test to.

coverage on plot_hdi before new tests: 93% to 96%

Victoria: essplot.py had coverage of 100% so added test_plot_khat_false_annotate() and test_plot_dist_kind_type_error() in arviz/tests/base_test/testplots_matplotlib.py to trigger an previously uncovered warning log and a type error.

Cover on /plots/distplot.py before: 90% after: 95% and khat.py before: 96% after: 100%

Self-assessment: Way of working

Current state according to the Essence standard: …

Would say we are in the “In Place”-phase. We work a lot more together than previously, we pretty much all know what our working habits and systems are. We are not at the “Working well”-phase yet as it did turn out pretty chaotic at the start of this assignment. Naturally the nature of the assignment means that there is a big onboarding process so it makes sense that it is a bit chaotic in the beginning but overall we are progressing nicely in our Essence standard way of working.

Overall experience

What are your main take-aways from this project? What did you learn?

Is there something special you want to mention here?

It was fun, it was a bit stressful. We lost a bit of time because of being stuck-up with our initial choice of repository at first. Maybe it is good to consider your choice a bit extra when the assignment itself specifically asks if you will continue with the chosen repository, ay? None of us were familiar with measuring branch coverage this way or cyclomatic complexity so those were new lessons: tho branch coverage definitely seems like a universally applicable concept overall than cyclomatic complexity that’s a bit iffy.