Human Computer Interaction

Victor Møller Poulsen, Studie Nr.: 201707639

June 7, 2021

1 Abstract

This paper presents a prototype Streamlit app which teaches a Gelman (Gelman et al., 2020) inspired Bayesian workflow in R with the package brms (Bürkner, 2017) and in Python with the package pyMC3 (Salvatier et al., 2016). The app is built with theories from user experience (UX) in mind and relies on a well-documented and reproducible code-base. The app is aimed at users who are already familiar with Bayesian analysis in either R (brms) or Python (pyMC3) and should be used as an introduction to the implementation that the user is unfamiliar with. The app has several interactive elements, including optional sections that the user can expand to dive deeper into parts of the material, quizzes with feedback and live rendering of plots and figures based on user selection.

Link to app: Stremlit app

Link to Github: Github page

Contents

1 Abstract 1

2 Introduction 2

3 Target Group 4

4 Intended Usage 5

5 Design Philosophy 6

6 Interaction 9

7 Code Base 11

8 Improvements & Testing 12

9 Conclusion 14

2 Introduction

The main idea with this app is to teach users who know how to conduct a Bayesian analysis in one coding language how to conduct a similar analysis in the other coding language. This is something that I had to figure out when we transitioned

from using R and brms for Bayesian analyses on the Cognitive Science bachelor's degree to using Python and pyMC3 on the Cognitive Science master's degree. There are lots of great resources and examples both online and in books for both brms and pyMC3 independently, but I have so far not found any resources that directly map between the two. That is the niche that this Streamlit app is intended to occupy.

Besides differing in content from books and online resources, this app is also intended to be interactive. This is important because it facilitates user engagement and agency (Janlert & Stolterman, 2017). It also useful because it means that we can let the user select what they want to see on their screen and which parts they would like to engage more with. It thus saves a lot of clutter from the app, because we do not have to display everything at once. This means that the app can have extensive functionality and retain high usability. This is where Streamlit becomes relevant. Streamlit is a Python package which allow users to launch interactive web applications to showcase analyses. It thus fulfills the same function that Shiny fulfills in the coding language R. Streamlit is developing rapidly however, and already appears to support more diverse options than Shiny. For instance, the expandable sections that I use throughout the app were released in 2021. I chose to use streamlit because it is an exciting and developing tool.

3 Target Group

Obtaining knowledge and understanding of the target group for a product is among the most important steps in creating a useful product (Mills et al., 1992, p. 13). The target group for this app is people who are familiar with Bayesian statistics and know how to conduct a Bayesian analysis in either R (brms) or Python (pyMC3). The idea is to conduct (almost) identical analyses in the two languages, such that users can transfer what they already know in one language to the other language. Explanations are targeted more towards users who want to transition from R (brms) to Python (pyMC3) than the other way. This is because I have kept Cognitive Science students in mind as the primary audience. Most Cognitive Science students have been taught Bayesian statistics in the R programming language, as implemented in the brms package. I am in the privileged position of knowing my audience well. I have both (1) been taught Bayesian statistics in R on the Cognitive Science Bachelor's degree, and (2) been instructor on the Bachelor's course in which the students learn Bayesian statistics in R. While the app is mainly designed with Cognitive Science students in mind, the app should also be useful for users outside of Cognitive Science. Python and R are the two largest Data Science programming languages, and as such it is common for Data Scientists to code in both languages.

4 Intended Usage

The first use case (*UC1*) that I think is reasonable is to use the app as a one-time thorough introduction, where users should ideally run their own code in Python and/or R while following the analysis on the app. This should be relatively easy since I provide reproducible code for all parts of the analysis, which can be easily copied to clipboard from the expandable code-chunks. Users do not need to manually select the code they want to copy but can simply click the text selection icon in the upper-right corner of the code-chunks. For *UC1* the user will have to interact with the page, expand code-selections and explanations and tweak the plots to get a feel for what is going on. As such, this is the use case where the user will utilize the full functionality of the app. The user might also learn something new, either conceptually or related to a healthy Bayesian workflow. This however is secondary to the main goal of building a bridge between the code implementations.

The second use case (*UC2*) that I think makes sense is to use the app repeatedly as a reference while conducting an analysis in either pyMC3 or brms. The user can quickly glance through the app to be reminded of how a good Bayesian workflow can look for either implementation. I have also included a section with references that I have found helpful for both pyMC3 and brms, and as such the user can also use the app as a gateway to more advanced analyses. For this use case the user might not want to expand the optional sections, but simply glance through the plots and headers to quickly find what they are looking for. For this use case, only a small part of the functionality in the app is used.

5 Design Philosophy

The design philosophy for the app is naturally related to the content that it attempts to convey and introduce to the user. It is also designed to facilitate both *UC1* and *UC2*. The design philosophy burrows from the Pragmatic/Hedonic model of User Experience (UX) which is described in Hassenzahl (2010) as well as the design principles outlined in Mills et al. (1992).

Firstly, the app is designed with a philosophy of *avoiding clutter* in mind. The importance of this is emphasized throughout Mills et al. (1992). I have hidden everything that I think is not necessary for navigating the app in expandable sections. I wanted to avoid cluttering the app with text specifically since this can feel intimidating and could scare users away. If users want to use the app in line with *UC1* they will have to interact with the optional elements and decide to see the code-chunks and the optional explanatory sections. If these were shown by default, I think that *UC2* would not be facilitated well by the app.

Secondly, the app is designed with *consistency* in mind. The importance of consistency is discussed in Mills et al. (1992, p. 7) where they highlight that consistency allows users to transfer knowledge from one application to another. In our case, since we only have one app, what will be important is that users can transfer what they have learned in one section to other sections. In other words, I am aiming for the app to be consistent within itself (Mills et al., 1992, p. 8). The users will quickly learn that there are three types of expandable boxes in the app: “Code-Monkey”,

"Language-Learner" and "Concept-Guru". Each of these expandable sections allow the user to access different types of additional info. When clicked, the "Code-Monkey" sections will pop out and show reproducible code, "Language-Learner" sections go into more detail about how R (brms) differ from Python (pyMC3) while "Concept-Guru" sections discuss Bayesian analysis more generally. Python and R placement on the page is consistent in the sense that everything which relates to R (e.g. plots and code) is located in the right column, while everythin related to Python is located in the left column. The order of presentation is also consistent throughout. Each subsection starts with a header and is then followed by a brief text. Below this are plots or outputs from Python and R, and only after this will the user find the optional sections. These are also presented in consistent order, such that "Code-Monkey" comes first, "Language-Learner" second and "Concept-Guru" comes last. For an example of this consistent format, see *Fig. 1* and *Fig. 2* below. Lastly, each expandable section has a unique icon associated with it. This has been implemented to make it easier for the user to navigate the page (see *Fig. 1 & Fig. 2*). Mills et al. (1992, pp. 223–255) focus heavily on the importance of icons. One highlighted advantage of icons is that users typically will recognize images faster than they will read text (Mills et al., 1992, p. 244).

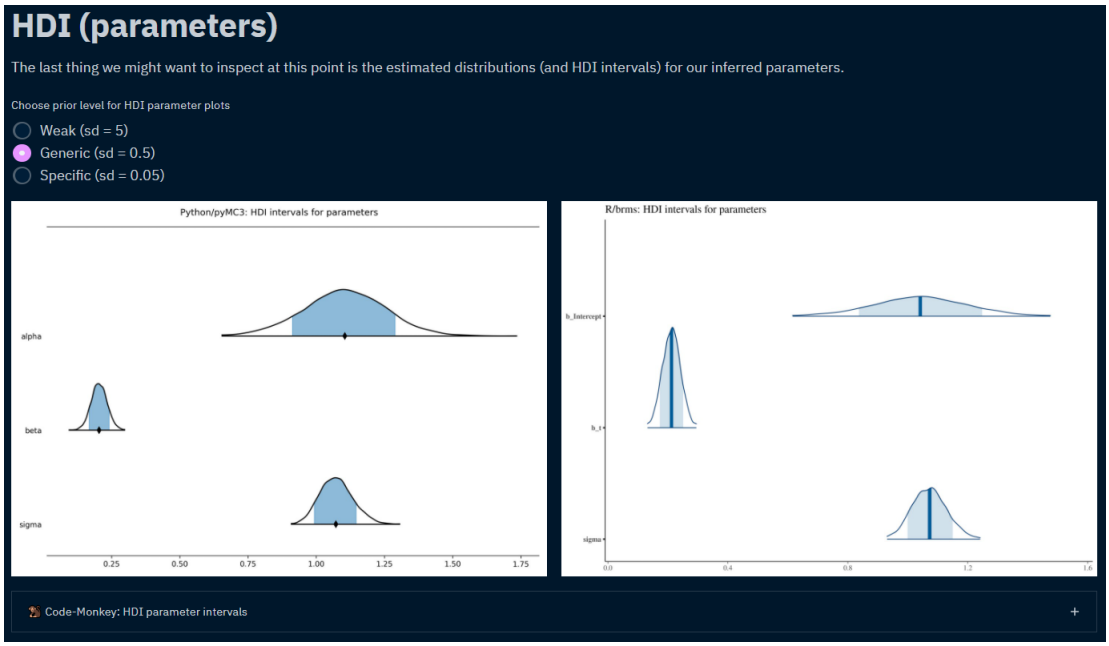

Figure 1: Screen capture from Streamlit. Shows the consistent, pretty and simple layout on the app. Each subsection starts with a header and a brief text. Then the user has a selection (here the *Generic* prior is chosen). A plot or figure is then shown based on the selection (left: Python, right: R). At the bottom, we see an expandable "Code-Monkey" section with a monkey icon.

Third, the app should be *pretty* and *simple*. These are general and obvious design principles which are discussed in Tractinsky et al. (2000) as well as Hassenzahl (2010, pp. 25–26). I have created a custom color-theme for the app, which I think makes the app more interesting than with the standard white theme. I have also tried to make the formatting both inside Streamlit, and for the R and Python plots pretty and consistent (see *Fig. 1* above & *Fig. 2* below). Considerations around making the app engaging and pretty, is another reason why I show plots by default and hide most text and code by default. I hope that users will find that the app and

the content on it is pretty, something that is understood to be an important driver in creating an enjoyable experience (Hassenzahl, 2010, p. 65).

6 Interaction

There are three basic types of interaction implemented in the app at this point. They provide a relatively high level of interactability, with adaptable outputs that are contingent on user selection (Janlert & Stolterman, 2017).

Firstly, the user can choose which model and prior to see analysis for. Plots and figures from R and Python are shown by default and cannot be toggled off by the user. However, the user can influence what is shown in plots. In most cases the user can choose among different prior options that have been used to fit the models in the app (see *Fig. 1* above). In some cases, the user can choose to plot the outputs of analysis between different models, e.g. with different random effects structure and likelihood functions. When the user makes a selection the plots and figures will change, and the code in the "Code-Monkey" sections will also change in the cases where the analyses differ. As such, the display is contingent on the actions of the user, and there is an element of adaptability imbedded in the app (Janlert & Stolterman, 2017).

Secondly, the user can choose to expand the optional sections: "Code-Monkey", "Language-Learner" and "Concept-Guru" (see *Fig. 2* below). As I have already said, the "Code-Monkey" section will render code which matches the selection.

Most often, the content in "Language-Learner" and "Concept-Guru" will not differ based on the selection of the user, because these sections contain more general considerations.

The degree to which a product engages the user in interaction has been called *interactiveness* (Janlert & Stolterman, 2017). This is implemented as quizzes that the app presents to users and the feedback that the app provides based on user answers. These quizzes are not shown on the page by default but hide inside the expandable boxes (see *Fig. 2* below). Again, this was done to ensure that the app will not be too cluttered and can be glanced through quickly (for *UC2*).

Streamlit is often used for Data Science applications which run extremely fast machine learning analyses or frequentist statistics. The apps are typically interactive in the sense that when the user makes a choice some corresponding code is run, which generates an output. This means that the possibility space is almost infinite, and as such these apps are high on interactiveness (Janlert & Stolterman, 2017). This is not the case in the prototype app that I am presenting. In my case, I am not running any analysis code live because full Bayesian sampling is prohibitively slow. Thus, I have made an extensive code base from which I generate plots and outputs from several analyses (e.g. for different priors and different models) that the user then sees based on selection. As such, the functionality, or interactability is limited to what I have already pre-run. This is a classic trade-off between functionality on one hand (flexible options) and usability on the other (no delay in runtime). However, Janlert and Stolterman (2017) point out that higher inter-

actability does not necessarily mean that a product gives the user a feeling of higher agency. Even though the app only offers limited interactability I hope that it does give users a sense of agency. If users want to tweak the analysis in ways that I have not facilitated, they *should* run their own code anyways. After all, the whole purpose is to teach users how to implement Bayesian analysis, and the best way for users to learn that is to code themselves.

7 Code Base

Although I have focused extensively on the user experience (UX) on the app, the most demanding work has been to develop good analyses and managing an extensive code base which must be reproducible. I will not go too deep into that here, as you can check the Github. There are bash-scripts for both R and Python which execute all the necessary scripts to run the whole pipeline and reproduce all analysis on the Streamlit app. This means that if users have the necessary packages in their environment, they can run all R and Python analysis with two lines of code from their terminal. This is of course unnecessary for most users, who will just stay in the app, but it is an option for users who want to dive deeper into the analysis and the convenience functions that I have created. It is important not exclusively to focus on the ”average” user (Mills et al., 1992, p. 14). Besides this, a well-structured code base has been absolutely necessary for me to be able to manage the project and ensure that all formatting is consistent.

8 Improvements & Testing

Something I really would have liked to implement in the app is a consistent color-code for the expandable sections that I have. This would make sense because I use the same three types of boxes (“Code-Monkey”, “Language-Learner” and “Concept-Guru”) throughout the app. A color scheme, where each type of section has its own color-code should make the page more intuitive to navigate. Unfortunately, this functionality is not natively supported in Streamlit. Streamlit can be configured with CSS code, so in principle it should be possible to implement. Unfortunately, I am unfamiliar with CSS so there is currently no color-coding for the expandable sections. As I have mentioned previously, each section has an associated icon, which aims to achieve the same goal of directing the users attention. I do think that colored sections would be more effective at that than small icons, and would make interaction with the app easier for the user.

With regards to feedback and interactivity there is one thing that would be ideal to implement for the use cases that the app attempts to facilitate. At present, the user is only quizzed with multiple choice questions (see *Fig. 2* below). This is because it is easy to handle, since I can specify the feedback for a limited number of options. This is what is currently supported, and can be characterized as a ”responding” interaction type (Rogers et al., 2011, p. 81). Because the app attempts to teach the user code-practices it would make sense to have a system which corresponds to what is used on platforms such as DataCamp. On DataCamp users type in code, which is then executed, and the user received feedback based on what the code

evaluates to. This would correspond to a ”conversing” interaction type (Rogers et al., 2011, p. 81), and would be more sophisticated than what is currently in place. The tricky part is that the code has to be compiled, and not just treated as a string of text. Treating the input as a string is infeasible, because code which accomplishes the same thing can be expressed in infinitely many ways, There also has to be a nice interface where the user can type in the code that they want to execute. I think that both would be tricky to implement, especially the first part. Another reason to not support this functionality is that I would ideally like users to code along from their own instance of R and/or Python.

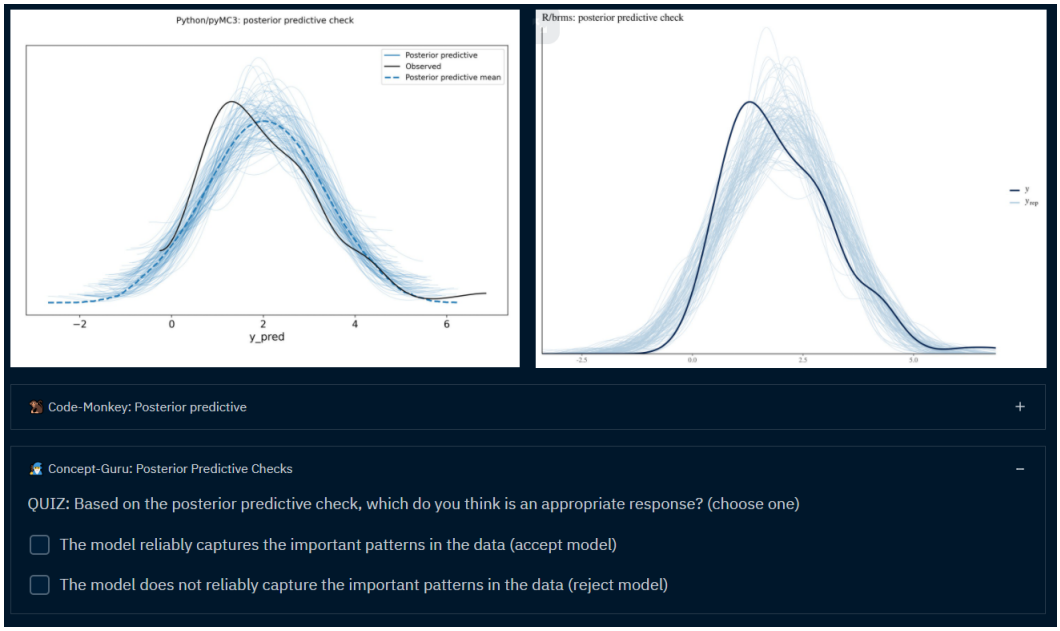

Figure 2: Screen capture from Streamlit. Shows plots for posterior predictive checks in the top (right: Python, left: R). Below is an expandable ”Code-Monkey” section, and below is and expanded ”Concept-Guru” section, which here has a quiz. As can be seen, the quiz is verbal, and only take multiple-choice answers. It will provide feedback once a selection has been made.

I intend to share the app in cognitive science forums once I have double checked that everything is reproducible, and that everything is well documented. Hopefully, the app will be useful to some of the students, and perhaps some outsiders will even find their way to the app. Additionally, sharing the app should give me the chance to receive feedback on whether the app is usable. Both whether the content is good and whether the format is intuitive. I intend to gather verbal feedback, although I know that this is not the ideal testing conditions.

There are at least two ways of gathering feedback that could be more useful than verbal feedback. Firstly, one could attempt a quantitative evaluation, e.g. based on A-B testing, eye-tracking data and statistical analysis. A good example of this approach can be found in Sutcliffe and Hart (2016). A more useful approach in this initial stage might be to simply observe users who interact with the app. Norman (1999) has argued that the best way of finding out how users engage with products is to observe them engage with products. I think this is true in our case, where one of my main interests is simply *how* users engage with the app (e.g. *UC1*, *UC2* or something else).

9 Conclusion

This paper has introduced a Streamlit app which teaches a Bayesian workflow in R (brms) and Python (pyMC3). The app is designed for users who are already familiar with Bayesian analysis in either R (brms) or Python (pyMC3). The app focuses

on building a bridge between the implementations in the different languages and packages. The app supports limited interaction through selection boxes, and expandable sections for the user who wants to dive deeper. The app also attempts to engage the user with quizzes and provides feedback based on answers. The app is designed with a philosophy of consistency, simplicity and beauty in mind. Users who want to go further than the app supports can consult a well-organized code-base at Github

References

Bürkner, P.-C. (2017). brms: An R package for Bayesian multilevel models using Stan. *Journal of Statistical Software*, *80*(1), 1–28. https://doi.org/10.18637/jss.v080.i01

Gelman, A., Vehtari, A., Simpson, D., Margossian, C. C., Carpenter, B., Yao, Y., Kennedy, L., Gabry, J., Bürkner, P.-C., & Modrák, M. (2020). Bayesian workflow. *arXiv preprint arXiv:2011.01808*.

Hassenzahl, M. (2010). Experience design: Technology for all the right reasons. *Synthesis lectures on human-centered informatics*, *3*(1), 1–95.

Janlert, L.-E., & Stolterman, E. (2017). The meaning of interactivity—some proposals for definitions and measures. *Human–Computer Interaction*, *32*(3), 103–138.

Mills, D., Bonn, E. A., & San Juan, S. S. T. M. (1992). Macintosh human interface guidelines.

Norman, D. (1999). Affordance, conventions, and design. *Interactions*, *6*, 38–42. https://doi.org/10.1145/301153.301168

Rogers, Y., Sharp, H., & Preece, J. (2011). *Interaction design: Beyond human-computer interaction*. John Wiley & Sons.

Salvatier, J., Wiecki, T. V., & Fonnesbeck, C. (2016). Probabilistic programming in python using PyMC3. *PeerJ Computer Science*, *2*, e55. https://doi.org/10.7717/peerj-cs.55

Sutcliffe, A., & Hart, J. (2016). Analysing the role of interactivity in user experience. *International Journal of Human-Computer Interaction*, *33*. https://doi.org/10.1080/10447318.2016.1239797

Tractinsky, N., Katz, A. S., & Ikar, D. (2000). What is beautiful is usable. *Interacting with computers*, *13*(2), 127–145.