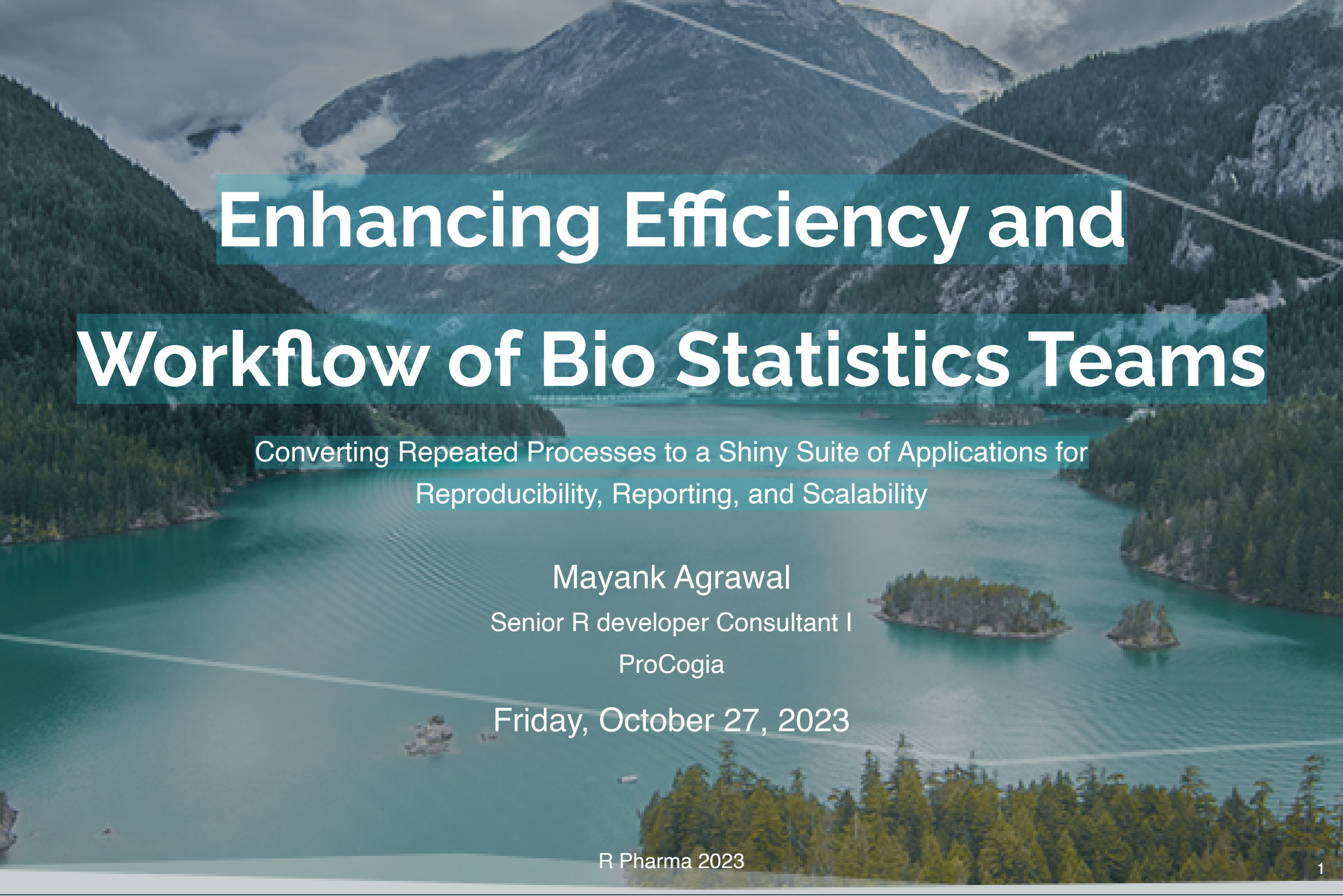

Enhancing Efficiency and Workflow of Bio Statistics Teams

Converting Repeated Processes to a Shiny Suite of Applications for Reproducibility, Reporting, and Scalability

Mayank Agrawal

Senior R developer Consultant I

ProCogia

Friday, October 27, 2023

Hello there!

- Background: Computer Science Engineer and MBA in Business Analytics.
- Curious to learn.
- Using R and R Shiny for 6+ years.
- Designed, built and managed more than 40+ R Shiny applications and dashboards.
- Built and led teams of 10+ R developers.
- Trained 30+ colleagues on R from diverse backgrounds.
- Extensive hands on experience on all flavors of R products - Shiny, R Markdowns, Quarto, and package development.
- Working with **ProCogia**; an Official full service partner with Posit.

Current Challenges

Clinical Trails are often complex and time consuming with repeated similar workflow steps for each new variation of the trail.

This often leads to

- Delayed Analysis and reporting
- Longer on-boarding time
- Delayed Time to Market

resulting in

- Hindered workflows
- Lower productivity
- Repeated boring processes

Key Issues

- Manual processes
- Lengthy simulation times
- Scalability constraints
- Limited collaboration
- Reporting challenges
- Flexibility
- Reproducibility

Solution

Empowering Biostatisticians with R Shiny suite of applications

- Collaborated with Biostats to understand their pain points and challenges.
- Developed a basic R Shiny app (MVP) for dosage simulation.
- Fully scaled through an iterative process of development and feedback
- Created multiple R Shiny apps targetted to specific workflows.
- Developed an ecosystem of Biostatistics R Shiny applications
- Automated and improved reporting.
- Enhanced workflow efficiency and productivity among biostats team

What was the journey to success?

Migration Process Steps

- Define scope of each process.
- Document repetition rate, importance, and time investment for each work request.
- Identify the most time-consuming, yet simplest workflow.
- Develop a **MVP** (Minimum Viable Product)
  - Showcase a demo with the smallest workflow.
  - This aids leaders in visualizing the impact of approval.
- Integrate workflows incrementally from small to large.

Engage with the Biostats team

- Weekly feedback sessions with biostats team and other stakeholders
- Adopt **Agile methodology** for faster iterative improvement and development.
- Procatively address feedback from from end users
  - Prioritize features based on complexity, time, effort and need.
  - Research and address suggested features to keep developers and users engaged and happy.
- Utilize GitHub Issues to document feature requests or bug fixes.
  - Ensure commits are linked with respective issues for traceability during testing.
- A picture speaks a thousand words. If it can be explained with a **screenshot**, don't shy away.
- Establish clear requirements to expedite implementation, with developer input on the overall app workflow.

Principles of Application Design

- Think about creating and showcasing dashboard outlines in the early discussions.
  - Prototype the UI layout using draw.io or similar tools.
  - Use conditional panels, modules, and well-designed layouts to create a user-friendly interface.
  - Iterate rapidly to test how different implementations enhance UI/UX.
- Establish a standard application/dashboard template layout such as Shiny Dashboard, bslib and/or bs4Dash.
  - Create custom R function wrappers based on organization theme and color layout.
  - Re-use the template for new builds and ideas to maintain consistency and coherence.

Principles of Application Design

- Define the **flow of reactivity** for the overall application/dashboard.
  - **Waterfall Reactivity Model**.
  - Avoid **Reactivity Spaghetti Mess**
  - Leverage reactive UI elements based on user input. This enhances interactivity and responsiveness.
- Provide default selections for input widgets, enabling users to explore and understand the application easily.
- Prioritize user-friendliness: if it's not intuitive, it won't be used.
- Incorporate "tooltips", "notifications", section write-ups and clear instructions to guide users in using the application.

Organize your project structure

- Maintain a clear and logical directory structure.
- Group related files together for easy navigation.
- Create, adopt and enforce a standard development template for easier developer onboarding.
- Have a project template repository for initial cloning.
- Please refer to the right image for an illustrative R project directory structure.

Streamline Workflow

- Use RStudio Projects
  - Isolates your development environment.
  - Relative path referencing for your project files.
  - Segregate development flow and context.
- Version Control with Git
  - Track changes, collaborate with team members, and manage project history effectively using Git.
  - Use Issues, Pull Requests and Connected Commits for efficient and effortless parallel development.

Shiny Development Tips

- Establish and adhere to a **standard** file and code structure.
- Implement **coding standards** to facilitate seamless collaboration between Bio Statistics teams and R developers.
- Emphasize the use of **functions** and **shiny modules** whenever possible.
- Optimize Server logic via **Profiling** and performance tuning
  - Minimize unnecessary computations and avoid redundant calculations.
  - Use tools like `profvis` to identify **performance bottlenecks** and Optimize critical sections of your code for speed.
  - Use **benchmarking** techniques to compare computation speeds of various packages. Refer R function `rbenchmark::benchmark()`.
- Create smaller, independent **Proof of Concepts (POCs)** for new feature requests.
  - For example, capture simulation attributes in a table and select them for subsequent runs.

Shiny Development Tips

- Debugging like a pro
  - Debug via `browser()`, `debug(function_name)`, and debug points.
  - Set breakpoints, step through code, and inspect variables to troubleshoot efficiently.
- Apply standard software development principles:
  - DRY (Don’t Repeat Yourself)
  - YAGNI (You aren’t going to need it)
  - KISS (Keep it Simple Silly)
  - Document your code for humans.
  - Prioritize clean code at all times.
  - Separation of concerns.

Facilitate Consistent Reporting

- Utilize **parameterized markdown** reports for dynamic MS Word report generation within the application.
- Adhere to the organization's document template for uniformity in reporting.
- Ensure each page or process has a predefined standard write-up with dynamic bits/sections based on simulation calculations.

Ensuring Reproducibility

- Leverage package management tools
  - Utilize `renv` or `packrat` to manage package dependencies.
  - Provides a controlled environment for your project.
  - Ensure reproducibility and minimize version conflicts.
- Rely on packages published and actively maintained on **CRAN** for a validated R environment.
- Establish unit tests for all functions used to ensure accurate and consistent results.
- Invest time to create an in-depth GitHub ReadMe with the sections **Project Overview**, **Getting Started**, **User Application Flow and Usage**, **Key Programming Concept Implemented** (if any) and **Developer Guide** providing comprehensive project reproducibility instructions.

Enhancing Scalability

- Implement **modularization** and **functional programming** for a **plug-and-play** development format across applications.
- Enable multiple studies to be added concurrently using standard **git branching strategies**, involving multiple concurrent developers.
- **Async Programming**: Evaluate longer simulations in a **separate R process** preventing app performance issues.
  - Few R packages to aid this are: callr, mirai, crew, coro, future and promises
- Take a step further and deploy simulation functions as internal APIs with **Plumber**.
- Write your **custom JavaScript and R bindings** for implementing unique feature requests.
- Approach feature requests as a blend of **web development**, **software engineering**, and **R development**.

Utilize Automated Testing

- Writing Test Cases (Inputs, Expected Outputs)
- Full Stack Testing
  - testthat for back-end testing,
  - shinytest2 for front end testing, and
  - shinyloadtest for load testing.
- Types of Tests: **Unit**, **Functional**, **Integration**, and **End-to-End**
- Continuous Integration for Testing with Git branching strategies.
- Benefits
  - Early bug detection
  - Efficiency and speed
  - Consistent and repeatable testing
  - Increased test coverage
  - Regression testing capabilities
  - Greater confidence in release stability

Common Application Features

- Download any **visualization** from the application.
- Access downloadable interim **simulation calculations** , aiding in simulation fine-tuning.
- Download **dynamically** rendered MS Word **reports** with FDA submission format for faster review and iterative changes.
- Add and execute multiple simulations simultaneously for efficiency.
  - **Compare** similar graphs side by side for comprehensive analysis across all simulations.
  - Generate dynamic **grouped plots** based on the sequence of simulations requested.
- Receive **email notifications** to stay updated on the progress of lengthy simulations, along with attached reports.

Common Application Features

- Include and display the underlying **mathematical equation** for a study or simulation on the user interface using **LaTeX**.
- Receive appropriate **notifications** upon completion of each process.
- **Helper Tabs** for - Application Information, Usage Manual, Release Tabs, Feedback and Contact Business Lead - all through the application.
  - Distribute an updated User Manual in the quarterly release email and include it as a ‘User Manual’ tab within the application.
  - Feature a ‘User Feedback’ tab for direct communication of feedback with the project manager via email.
  - Include a ‘Release Notes’ tab to display app changes over time, promoting transparency.

Enhancing User Adoption

- Create **GIFs** showcasing the application layout and user flow.
  - Include them in the Git readme and announcement emails.
- Create detailed application interaction user manuals with screenshots and highlights for each step.
- Conduct regular (quarterly) **training sessions** to provide guidance, answer questions, and assist users with new features.
  - Record and share them for easier re-visit.
- Continuously engage user base for better ROI and on boarding.
- Prioritize most requested user features for each sprint.

Thank you

- Quarto presentation code available on GitHub at http://bit.ly/3ZHZTNf
- Slides available on GitHub Pages at https://bit.ly/3RsdExn
- Connect and/or send me a DM for a follow up question or catch up
  - Linkedin: mayank-agrawal-7jan
  - X (previously Twitter): mayank7jan
  - Mastodon: mayank7j

References - R Packages

- shinyDashboard, bslib, bs4dash for standard dashboard template.
- rmarkdown and Quarto for parameterized reporting.
- renv for package management in a R project.
- glue for interpreted string literals for dynamic reporting.
- callr for separate r sessions.
- plumber for API creation.
- httr2 for API calls.
- pins for shareable secured publishing of data, models, and R objects
- testthat, shinytest2 and shinyloadtest for testing.
- dplyr for data manipulation.
- ggplot2, plotly and echarts4r for visualization.
- profvis for code profiling and time estimation

R Pharma 2023