PharmaSUG 2024 - Paper DV - 458

Quarto 1.4: Revolutionizing Open-source Dashboarding Capabilities

Joshua J. Cook, M.S., ACRP-PM, CCRC

Kirk Paul Lafler (@sasNerd)

ABSTRACT

The release of Quarto 1.4 marks a significant advancement in the realm of data visualization and reporting, introducing powerful dashboarding capabilities that cater to the evolving needs of data analysts. This abstract presents an introduction to utilizing Quarto for dashboard creation, emphasizing the importance of this development for the pharmaceutical and biotech industries. Dashboarding in Quarto 1.4 is pivotal, as it allows for the integration of complex data into cohesive and interactive visual summaries. This paper will explore the innovative features of Quarto's dashboarding tool, which simplifies the process of synthesizing large datasets into accessible and actionable insights. Quarto dashboards support a variety of output formats, which enable analysts to communicate their findings to a broad audience, ranging from high-level stakeholders to the general public. The versatility of Quarto ensures that users can create custom dashboards that are both informative and engaging, regardless of the viewer's technical expertise. The paper will demonstrate how Quarto's cross-language support, combined with its dashboarding functionality, provides a comprehensive environment for data reporting. This integration is crucial for many industries where making informed decisions often depends on understanding complex datasets. By leveraging Quarto's dashboarding tools, analysts from a variety of programming backgrounds can more effectively present their data, leading to improved decision-making processes and clearer communication of findings. The potential of Quarto to enhance open-source data visualization and reporting practices within the industry will be a focal point of this paper.

INTRODUCTION

DASHBOARDS

Dashboards are defined by Oxford as a “graphical summary of various pieces of important information, typically used to give an overview of a business.” In practice, dashboards are being used by more than just businesses, including hospitals, schools, and research organizations. But what exactly makes a dashboard distinct from a graph or other standard visualization tool? The key difference is in the data update capabilities of dashboards, which are typically much more frequent and often real-time compared to a static visualization. However, static dashboards are also common. Due to this, dashboards have become widely used and highly influential analytic tools that offer often real-time insights to inform decision makers.

This paper is not intended to serve as a comprehensive introduction to dashboards, for that, we direct you to the Recommended Reading section below. Instead, we will focus on the production of high-quality dashboards using Quarto 1.4, a free and open-source tool.

APPROACHES TO DASHBOARDING

Data scientists have developed many unique approaches to dashboarding that utilize different programming languages and techniques, including proprietary systems such as **Tableau**© or **Microsoft Power BI**©, as well as **open-source software (OSS)** options utilizing **R** or **Python**. Tableau© is a proprietary dashboarding tool that has become an industry standard in healthcare, research, and business. On the other hand, R is an open-source programming language for statistical reporting that is like the S language. R is not an industry standard but is gaining popularity and has recently been under development for pharmaceutical and regulatory use (see Shiny App Successfully Reviewed by FDA CDER Staff (Pilot 2 Announcement 2) - R Consortium (r-consortium.org)). Unlike Tableau©, R is not built solely for dashboard development, and has many other capabilities. However, several key packages such as flexdashboard, dash, and Quarto have been released to serve dashboarding purposes.

On January 24th, 2024, Posit released **Quarto 1.4,** which introduced **Quarto Dashboards** that aim to:

> Streamline the creation of interactive dashboards, giving you an effortless way to lay out interactive components, visualizations, tabular data, and annotations.

Prior to this, Quarto was still capable of creating dashboards (see WUSS-2023-Paper-165.pdf), however it was noted that:

1. It was difficult to specify the exact locations of data structures (now called cards).
2. Printing to HTML caused cards to become distorted.
3. Some visualizations like pie charts were much more complicated to create and fit into the dashboard real estate.

It was our hope that these issues would be addressed by a future update, allowing Quarto to become the gold-standard in open-source dashboarding, and we believe this update has achieved that goal. Let's get started with dashboarding in Quarto 1.4, starting from the ground up, with emphasis on each of the three previously identified areas of concern.

EXAMPLE DATA

Dashboards often use robust datasets that benefit from drilldown and storytelling. Furthermore, these datasets are frequently updated in time. Thus, the dashboard examples in this paper utilize a publicly available dataset that has been extracted from the **Aggregate Analysis of ClinicalTrials.gov (AACT)** database ("Clinical Trials Transformation Initiative" n.d.). This dataset, `clinical_trials_fl`, was generated by querying the AACT database using open-source tools like R, RStudio, DBI, PostgreSQL, the tidyverse suite, and gt. Specifically, this dataset represents all the clinical trials registered and completed within the state of Florida of the United States between 23MAR2014-23MAR2024 (10 years). The AACT database updates daily, and thus this dataset, and its dashboard, would need to be continuously updated. This dataset currently includes 73,500 observations of 39 variables, relating to both the protocol designs and the completion of the clinical trials, which are detailed in Table 1. All conditions and interventions are included for analysis, and the dataset was cleaned and wrangled by the authors prior to dashboarding.

Index	**Variable**	**Meaning**
1	nct_id	(primary key) NCT ID is a unique identification code given to each clinical study registered on ClinicalTrials.gov. The format is the letters "NCT" followed by an 8-digit number (for example, NCT00000419).
2	start_date	Registration date of the study.
3	completion_date	Completion date of ALL endpoints of the study.
4	study_type	Type of study (Interventional, Observational, Expanded Access).
5	acronym	Acronym of the study (if applicable).
6	baseline_population	The intended source of the enrollment population (ex: clinic, associated hospital).
7	brief_title	Brief title of the study that was analyzed.
8	official_title	Official study (protocol) title.
9	overall_status	Overall status of the study (Completed, Terminated, Enrolling, etc.).
10	phase	Phase of clinical development (Early, I-IV, combinations).
11	enrollment	Enrollment numbers. According to the planned/actual enrollment information

		provided in the protocol section of the study record.
12	enrollment_type	Describes if the enrollment number is planned or actual.
13	source	The sponsor of the study.
14	number_of_arms	The number of treatment arms in the study.
15	why_stopped	If stopped prematurely, an explanation of why.
16	has_expanded_access	Logical indicating if EA was utilized.
17	is_fda_regulated_drug	Logical indicating if the study involved an FDA regulated drug.
18	is_fda_regulated_device	Logical indicating if the study involved an FDA regulated device.
19	site	Site name (usually multi-site).
20	city	Location of the site.
21	state	Location of the site (all Florida in this dataset).
22	zip	Location of the site.
23	country	Location of the site (all United States in this dataset).
24	number_of_facilities	Number of facilities in the U.S.
25	actual_duration	Duration (in months) from start_date until the completion of the primary endpoint.
26	were_results_reported	Logical indicating if results were submitted to ClinicalTrials.gov.
27	has_us_facility	Logical indicating if the study had a site in the U.S. (all true in this dataset).
28	has_single_facility	Logical indicating if the study had only one site.
29	minimum_age_num	Minimum age enrolled in the study.
30	maximum_age_num	Maximum age enrolled in the study.
31	minimum_age_unit	Unit of minimum age.
32	maximum_age_unit	Unit of maximum age.
33	number_of_primary_outcomes_to_measure	Number of primary outcomes listed in the protocol.
34	number_of_secondary_outcomes_to_measure	Number of secondary outcomes listed in the protocol.
35	number_of_other_outcomes_to_measure	Number of teritary+ outcomes listed in the protocol.
36	condition_name	Name of the studied condition (indication).
37	condition_name_lower_case	Lowercase name of the studied condition (indication).
38	duration_months	Full duration of the study (in months) from start_date to completion_date.

39	duration_years	Full duration of the study (in years) from start_date to completion_date.

Table 1. All Variables Included in the `clinical_trials_fl` dataset extracted from the AACT database ("Clinical Trials Transformation Initiative" n.d.).

SETUP WITH R AND RSTUDIO

This demonstration utilized **RStudio®** as the primary **Integrated Development Environment (IDE).** As with any project in RStudio®, a primary folder should be setup that will be used as the working directory. This is where the R project (*.Rproj*) file will be housed, as well as any required subfolders. These examples use the following:

```
C:\PharmaSUG2024\
```

Create subordinate folders for resources that will be required by markdown, such as data and images:

```
C:\PharmaSUG2024\data\
C:\PharmaSUG2024\images\
```

Within RStudio®, only three libraries are installed and imported-`tidyverse`, which is used for data wrangling and visualization, `gt,` which is used for table generation, and `quarto`, which is an open-source publishing system used to generate the dashboard. Finally, the dataset is imported:

```
    packages <- c("tidyverse", "gt", "quarto")

    if (any(!sapply(packages, requireNamespace, quietly = TRUE))) {
      install.packages(packages[!sapply(packages, requireNamespace, quietly
= TRUE)])
    }

    library(tidyverse)
    library(gt)
    library(quarto)

 clinical_trials_fl <- readRDS("query_results_clean_final.rds")
```

The Quarto document can then be created with RStudio® and saved to the primary folder with any filename (although we **highly** recommend you name this file "index.qmd" as explained below. Do not use a standard R script.

QUARTO 1.4 SYNTAX

THE BASICS

Before we get started, you should become familiar with these terms:

- **Dashboard** – a snapshot of information, typically a high-level overview that can be narrowed-down, or "drilled down" to review specific selections of data.
  - **Static Dashboard** – dashboards that provided a fixed view of data, without frequent updates and frequently without the ability to drill-down.
  - **Dynamic Dashboard** – dashboards that are updated frequently, and often allow users to drill-down and interact with the data.
- **Report** – a comprehensive document, typically in extreme detail that has subsections for each topic of interest.

Quarto can generate all these formats (and many more), but we will focus on dashboards specifically. Dashboards written with Quarto 1.4+ can utilize Python (*via Jupyter notebooks - .ipynb*), R (*via plain text markdown - .qmd*), Julia, and Observable ("Quarto - Quarto Dashboards" n.d.). Importantly, Quarto dashboards can include various mixes of these languages as required by the user/company. Therefore, you could create a dashboard that includes both `ggplot2` and `matplotlib` visualizations. Quarto dashboards consist of several components that we will use throughout all examples:

1. **Navigation Bar** – serves to provide navigation between pages of the dashboard (if more than one exists).
2. **Pages** – dashboards have traditionally been defined as a single page. However, with more complex datasets, it may be useful to divide the data into pages of a dashboard, each telling its own story.
3. **Cards** – cards are the *fundamental unit* of quarto dashboards and are the containers of all data elements (tables, figures, markdown). The content of cards maps to the cells of a notebook or markdown document. Cells that do not produce output are not considered cards. This advancement leads to easier location specification compared to the referenced paper.
4. **Rows/Columns** – rows and columns are used to define where each data element (tables, figures markdown) is positioned on the dashboard.
5. **Tabsets** – these are groups of rows or columns on a dashboard that can be used to further divide content.
6. **Sidebars and Toolbars** – sidebars are collapsible vertical panels for inputs while toolbars are horizontal panels for inputs. This is where filters, tick boxes, and other drill-down tools can be placed.

Within each page, cards are laid on the dashboard using sets of rows and columns, which we define via markdown headings. Content is be placed within each markdown heading (i.e., card) to ensure proper layout within the dashboard real estate. The syntax associated with markdown is also outside the scope of this paper, but there are many free and publicly available resources to help with writing markdown syntax (Quarto – Markdown Basics).

Let's get started with a hand-on approach to dashboarding with Quarto 1.4.

OUR FIRST DASHBOARD

For the purposes of this tutorial, our dashboards will only consist of R and associated R packages. It is straight-forward to ask ChatGPT to translate these sections into Python and matplotlib, if desired. Our goal for this dashboard is threefold:

1. Visualize how many clinical trials are successfully completed each year in the state of Florida.
2. Visualize the phase distribution of these clinical trials, which may help determine how many patients are receiving care via clinical research (larger numbers of participants are required for higher phase studies, especially phase 3 and phase 4).
3. Visualize the top 10 conditions that are being successfully studied in Florida.

We will organize the three visuals into one page with three cards, divided into one column with two rows, as shown in Figure 1. We want chart 1 (the trials completed per year) to be emphasized more, while the other two charts supplement our understanding of chart 1.

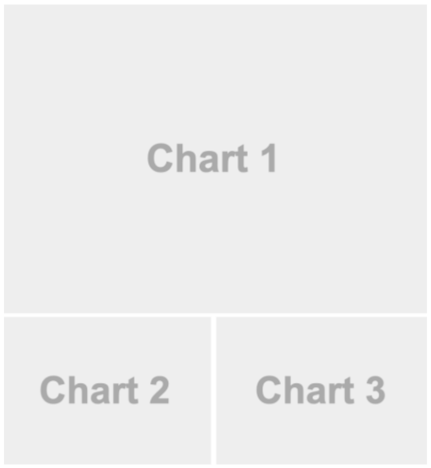


Figure 1. The dashboard layout for the specified three visuals (“Quarto - Quarto Dashboards” n.d.).

Let’s try this out using markdown. First, our YAML header (generated each time a Quarto document is made) will be edited to include our title, author names, and format as dashboard. It is of utmost importance that the format be switched to dashboard for our layout components to be recognized correctly. Optionally, a logo and navigation buttons can be added to the navigation bar.

Following the YAML header, pages are defined using level one headings (with one #). By default, level two headings (with two #) define rows. The YAML header can be edited to switch this to orient by columns:

```
---
title: "Dashboarding with Quarto"
author: "Joshua J. Cook, Kirk Paul Lafler"
format:
  dashboard:
    orientation: columns
    logo: images/logo.png
    nav-buttons: [linkedin, github]
embed-resources: true
---
```

We will continue with the standard row layout.

```
---
title: "Dashboarding with Quarto"
author: "Joshua J. Cook, Kirk Paul Lafler "
format:
  dashboard:
    logo: images/logo.png
    nav-buttons: [linkedin, github]
embed-resources: true
---
```

Importantly, the row names are only used for clean coding, and are not actually displayed in the dashboard. Sizes of the cards can be altered in the row headings, as show below:

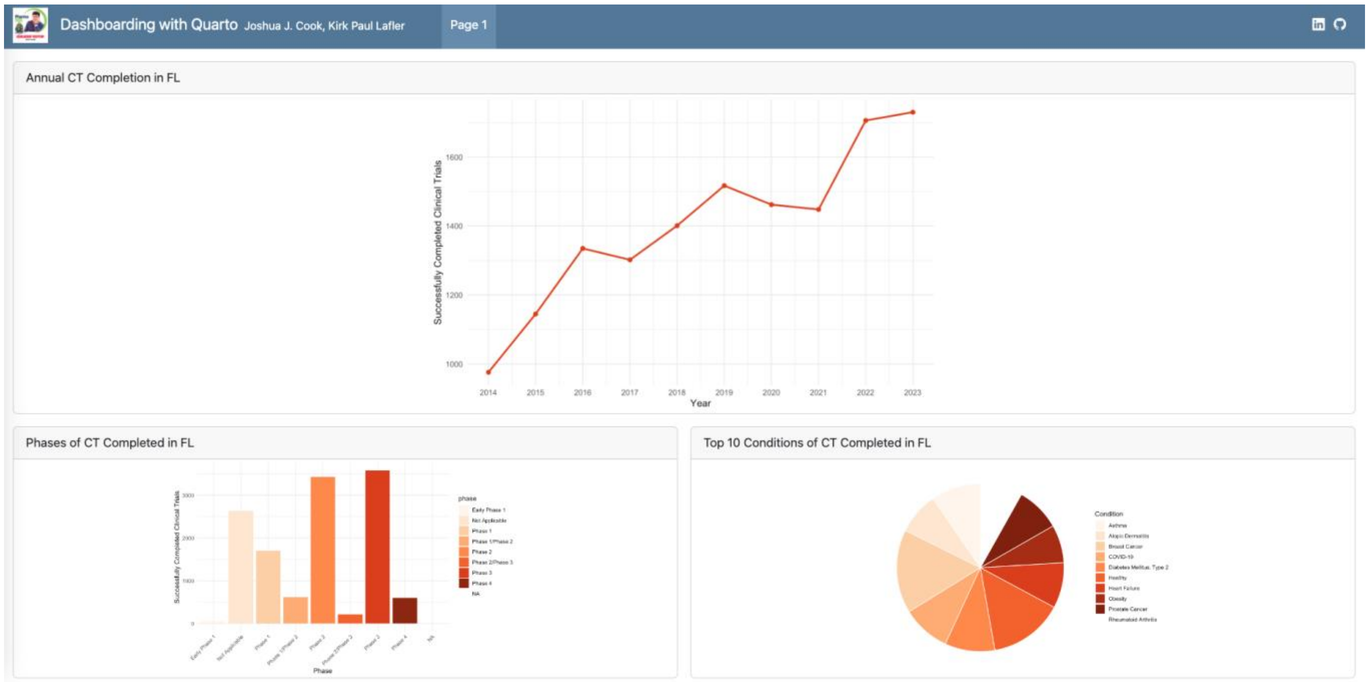
~~~
# Page 1

## Row 1 - Big Line Plot {height="70%"}

```{r}
#| title: "Annual CT Completion in FL"
#| padding: 0px

clinical_trials_fl_yearly_summary <- clinical_trials_fl %>%
  mutate(year = year(completion_date)) %>%
~~~

```
      group_by(year) %>%
      summarise(count = n_distinct(nct_id)) %>%
      filter(year >= 2014 & year <= 2023)

    # Extract the 7th color from the "Oranges" palette
    oranges_palette <- brewer.pal(8, "Oranges")
    dark_orange <- oranges_palette[7]

    # Line chart
    ggplot(clinical_trials_fl_yearly_summary, aes(x = year, y = count)) +
      geom_line(color = dark_orange, size = 1) +  # Specify the color and
size of the line
      geom_point(color = dark_orange, size = 2) +  # Specify the color and
size of the points
      scale_x_continuous(breaks = 2014:2023) +  # Ensure all years are
shown
      labs(x = "Year",
           y = "Successfully Completed Clinical Trials") +
      theme_minimal()  # Use a minimal theme for aesthetics
    ```

    ## Row 2 - Two small bar and pie plots {height="30%"}

    ```{r}
    #| title: "Phases of CT Completed in FL"
    #| padding: 0px

    clinical_trials_fl_phase_summary <- clinical_trials_fl %>%
      distinct(nct_id, .keep_all = TRUE) %>%  # Keep only unique NCT_IDs
      count(phase)  # Count the number of NCT_IDs for each phase

    clinical_trials_fl_phase_summary <- clinical_trials_fl_phase_summary
%>%
      mutate(across(where(is.character), ~na_if(., "Not Applicable")))

    # Bar chart
    ggplot(clinical_trials_fl_phase_summary, aes(x = phase, y = n, fill =
phase)) +
      geom_bar(stat = "identity") +  # Use identity stat to plot counts
directly
      scale_fill_brewer(palette = "Oranges") +
      labs(x = "Phase",
           y = "Successfully Completed Clinical Trials") +
      theme_minimal() +  # Minimal theme for aesthetics
      theme(axis.text.x = element_text(angle = 45, hjust = 1))  # Rotate x-
axis labels for readability
    ```

    ```{r}
    #| title: "Top 10 Conditions of CT Completed in FL"
    #| padding: 0px

    clinical_trials_fl_condition_summary <- clinical_trials_fl %>%
      group_by(condition_name) %>%
      summarise(n = n_distinct(nct_id)) %>%
      ungroup() %>%
      arrange(desc(n)) %>%
      top_n(10, n)  # Adjust this to include more or fewer conditions
```

```
    # 2. Create a pie chart
    ggplot(clinical_trials_fl_condition_summary, aes(x = "", y = n, fill =
condition_name)) +
      geom_bar(width = 1, stat = "identity", color = "white") +
      coord_polar(theta = "y") +  # This converts the bar chart to a pie
chart
      scale_fill_brewer(palette = "Oranges") +  # Apply the Oranges palette
      labs(x = NULL,
           y = NULL,
           fill = "Condition") +
      theme_minimal() +
      theme(axis.line = element_blank(),
            axis.text = element_blank(),
            axis.ticks = element_blank(),
            axis.title = element_blank(),
            panel.grid = element_blank())
    ```
```

The resulting dashboard is shown in Figure 2. Charts 2 and 3 are quite small, so we can adjust them by changing the {height="30%"} to {height="50%"} within the row markdown heading. Alternatively, each individual chart can also be expanded to full screen without loss in quality (a huge benefit compared to the old versions of dashboards) by clicking the expansion buttons in the lower right-hand corners of each chart. Going back to base markdown syntax, each individual chart can be adjusted by using knitr options fig-width and fig-height at the beginning of each code cell.



Figure 2. Our initial dashboard, with a timeline of annual clinical trial completion in Florida as chart 1, and two supplemental charts that add context regarding the phases and indications of these clinical trials.

That's it for the basic dashboard with Quarto (completely free, open-source, and only about 100 lines of code total)!

BEYOND THE BASICS

A few features can be added to Quarto dashboards to improve navigation, visibility, and storytelling:

- **Fill v. Flow** – instead of defining explicit dimensions for charts (like we did with height above), we can allow charts to fill the available space {.fill} which is the *default* behavior, or flowing to its natural height {.flow}. Plots in `Shiny` dashboards dynamically resize.
```

- **Scrolling** – for very large dashboards, scrolling could be enabled, but it may also be useful to consider pagination or tabsets.
- **Padding** – padding (i.e., white space) can be added to charts to make them more distinct from surrounding content. We disabled padding in the example above.
- **Interactivity** – this can be achieved using a combination of `htmlwidgets` (which is utilized by packages such as `Plotly`, `Leaflet`, and `DT`), as well as `Shiny`. This applies to both figures and tables where drill-down, searchability, or dynamic resizing is preferred.
- **Value Boxes** – value boxes are hallmarks of traditional dashboards, frequently being utilized to describe single values such as key performance indicators or counts of an important variable. Value boxes are available in a variety of formats in Quarto 1.4 and can be customized with colors and icons. Importantly, the single value is pulled from a flat R value that will need to be extracted from the dataframe (shown in the code below).
- **Content is Context** – markdown content can be added anywhere in a dashboard using the {.card} div, and can even be added within a chart itself. This content can be dynamic by referencing in-line code. Titles of charts can also be dynamic in the same way.
- **Multiple Card Output** – if a card contains multiple charts, the output can be organized using knitr options layout-ncol and layout-nrow
- **Theming** – Quarto dashboards have access to all the themes available to other quarto documents and can be defined in the YAML header. All available themes can have their components (font, font size, color, table of contents, etc.) customized using Sass files *(.scss).*
- **Parameters** – computational parameters can be set using knitr (in this case) or Jupyter. For knitr, this is defined in the YAML. Setting parameters allows for reports to be generated for specific subsets of a variable. For example, generating three versions of the dashboard based on each subset of the `study_type` variable.

Let's build a second dashboard, implementing as many of these features as possible. Let's:

1. Allow all figures to fill the entire card space (code not shown because that is the default behavior).
2. Enable scrolling.
3. Add a small amount of padding to each chart.
4. Convert our `ggplot2` figures to `Plotly` (using the `ggplotly()` function for some figures, but this did not always work as shown in the code).
5. Add a top row with three value boxes to describe the study statuses.
6. Add some context using markdown.
7. Apply the united theme.
8. Add custom navigation buttons:

```
---
title: "Dashboarding with Quarto"
author: "Joshua J. Cook, Kirk Paul Lafler"
format:
  dashboard:
    theme: united
    scrolling: true
    logo: images/logo.png
    nav-buttons:
      - website
      - icon: person-raised-hand
        href: https://joshua-j-cook-portfolios.netlify.app/
      - li
      - icon: linkedin
        href: https://www.linkedin.com/in/KirkPaulLafler/
```

~~~
        - github
    embed-resources: true
    ---

    ```{r}
    #| include: false

    packages <- c("tidyverse", "gt", "quarto")

    if (any(!sapply(packages, requireNamespace, quietly = TRUE))) {
      install.packages(packages[!sapply(packages, requireNamespace, quietly
= TRUE)])
    }

    library(tidyverse)
    library(RColorBrewer)
    library(plotly)
    library(gt)
    library(quarto)

    clinical_trials_fl <- readRDS("query_results_clean_final.rds")
    str(clinical_trials_fl)

    clinical_trials_fl %>%
      select(nct_id, overall_status)

    clinical_trials_fl_completed <- clinical_trials_fl %>%
      filter(overall_status == "Completed") %>%
      summarise(total_unique_completed = n_distinct(nct_id)) %>%
      pull(1)

    clinical_trials_fl_recruiting <- clinical_trials_fl %>%
      filter(overall_status == "Recruiting") %>%
      summarise(total_unique_completed = n_distinct(nct_id)) %>%
      pull(1)

    clinical_trials_fl_terminated <- clinical_trials_fl %>%
      filter(overall_status == "Terminated") %>%
      summarise(total_unique_completed = n_distinct(nct_id)) %>%
      pull(1)
    ~~~

    # Page 1

    ## Row 1 - Value Boxes

    ```{r}
    #| content: valuebox
    #| title: "Total Trials Completed"
    list(
      icon = "person-check-fill",
      color = "success",
      value = clinical_trials_fl_completed
    )
    ```

    ```{r}
    #| content: valuebox
    #| title: "Total Trials Recruiting"
```

```
    list(
      icon = "person-fill-add",
      color = "warning",
      value = clinical_trials_fl_recruiting
    )
    ```

    ```{r}
    #| content: valuebox
    #| title: "Total Trials Terminated"
    list(
      icon = "person-fill-x",
      color = "danger",
      value = clinical_trials_fl_terminated
    )
    ```

    ## Row 2 - Big Line Plot

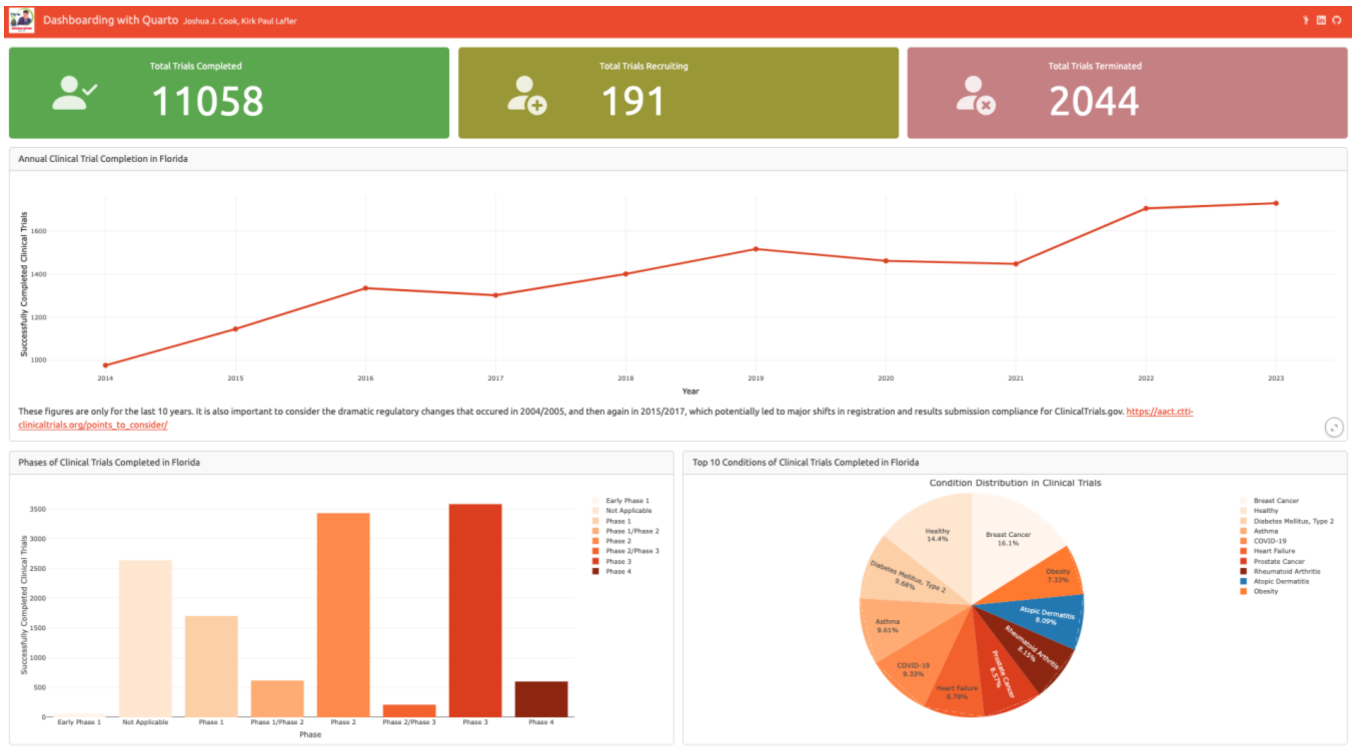
    ::: {.card title=" Annual Clinical Trial Completion in Florida"}

    ```{r}
    #| title: "Annual Clinical Trial Completion in Florida"
    #| padding: 10px

    clinical_trials_fl_yearly_summary <- clinical_trials_fl %>%
      mutate(year = year(completion_date)) %>%
      group_by(year) %>%
      summarise(count = n_distinct(nct_id)) %>%
      filter(year >= 2014 & year <= 2023)

    # Extract the 7th color from the "Oranges" palette
    oranges_palette <- brewer.pal(8, "Oranges")
    dark_orange <- oranges_palette[7]

    # Line chart
    gg <- ggplot(clinical_trials_fl_yearly_summary, aes(x = year, y =
count)) +
      geom_line(color = dark_orange, size = 1) +  # Specify the color and
size of the line
      geom_point(color = dark_orange, size = 2) +  # Specify the color and
size of the points
      scale_x_continuous(breaks = 2014:2023) +  # Ensure all years are
shown
      labs(x = "Year",
           y = "Successfully Completed Clinical Trials") +
      theme_minimal()  # Use a minimal theme for aesthetics

    # Convert the ggplot object to a plotly object
    ggplotly(gg)
    ```

    These figures are only for the last 10 years. It is also important to
    consider the dramatic regulatory changes that occured in 2004/2005, and
    then again in 2015/2017, which potentially led to major shifts in
    registration and results submission compliance for ClinicalTrials.gov.
    https://aact.ctti-clinicaltrials.org/points to consider/
    :::
```
```

~~~
    ## Row 3 - Two small bar and pie plots

    ```{r}
    #| title: " Phases of Clinical Trials Completed in Florida"
    #| padding: 10px

    clinical_trials_fl_phase_summary <- clinical_trials_fl %>%
      distinct(nct_id, .keep_all = TRUE) %>%  # Keep only unique NCT_IDs
      count(phase)  # Count the number of NCT_IDs for each phase

    clinical_trials_fl_phase_summary <- clinical_trials_fl_phase_summary
%>%
      mutate(across(where(is.character), ~na_if(., "Not Applicable")))

    # Bar chart; ggplotly() didn't work because of scales, reverted to full
plotly

    plot_ly(data = clinical_trials_fl_phase_summary,
            x = ~phase,
            y = ~n,
            type = 'bar',
            color = ~phase,
            colors = "Oranges") %>%
      layout(xaxis = list(title = "Phase"),
             yaxis = list(title = "Successfully Completed Clinical
Trials"))
    ~~~

    ```{r}
    #| title: "Top 10 Conditions of Clinical Trials Completed in Florida"
    #| padding: 10px

    clinical_trials_fl_condition_summary <- clinical_trials_fl %>%
      group_by(condition_name) %>%
      summarise(n = n_distinct(nct_id)) %>%
      ungroup() %>%
      arrange(desc(n)) %>%
      top_n(10, n)  # Adjust this to include more or fewer conditions

    # 2. Create a pie chart; same as above
    plot_ly(clinical_trials_fl_condition_summary,
                     labels = ~condition_name,
                     values = ~n,
                     type = 'pie',
                     textinfo = 'label+percent',
                     marker = list(colors = RColorBrewer::brewer.pal(n = 8,
name = "Oranges")),
                     hole = 0.0) %>%
      layout(title = 'Condition Distribution in Clinical Trials',
             showlegend = TRUE,
             xaxis = list(showgrid = FALSE, zeroline = FALSE,
showticklabels = FALSE),
             yaxis = list(showgrid = FALSE, zeroline = FALSE,
showticklabels = FALSE))
```
```

Our `plotly`-interactable (powered by `htmlwidgets`) dashboard is shown in Figure 3. This is contained within the HTML output and browser, and thus does not require a backend server to function. This can be uploaded to sites such as GitHub pages, or directly emailed to your colleagues.
```


Dashboarding with Quarto Joshua J. Cook, Kirk Paul Lafler
Total Trials Completed
11058
Total Trials Recruiting
191
Total Trials Terminated
2044
Annual Clinical Trial Completion in Florida
Successfully Completed Clinical Trials
Year
Phases of Clinical Trials Completed in Florida
Phase
Early Phase 1
Not Applicable
Phase 1
Phase 1/Phase 2
Phase 2
Phase 2/Phase 3
Phase 3
Phase 4
Top 10 Conditions of Clinical Trials Completed in Florida
Condition Distribution in Clinical Trials
Breast Cancer 16.1%
Healthy 14.4%
Diabetes Mellitus, Type 2 9.68%
Asthma 9.61%
COVID-19 9.33%
Heart Failure 8.78%
Prostate Cancer 8.5%
Rheumatoid Arthritis 8.15%
Atopic Dermatitis 8.09%
Obesity 7.33%

These figures are only for the last 10 years. It is also important to consider the dramatic regulatory changes that occured in 2004/2005, and then again in 2015/2017, which potentially led to major shifts in registration and results submission compliance for ClinicalTrials.gov. https://aact.ctti-clinicaltrials.org/points_to_consider/

**Figure 3.** An updated, plotly-interactable (powered by `htmlwidgets`) version of our dashboard. This is hosted locally using HTML and thus does not need a server.

Although this dashboard can be interacted with (i.e., clicking and hovering reveals additional information), it cannot yet be filtered or drilled down, which is a hallmark of modern dashboards. Let's fix that using new features from Quarto 1.4!

### ADVANCED DASHBOARDS – <u>SERVER REQUIRED</u>

There are three ways to add layout inputs (i.e., drill downs or filters) in Quarto, each with their own distinguishing area and background color:

1. **Sidebars** – collapsible <u>vertical</u> panels.
2. **Toolbars** – collapsible <u>horizontal</u> panels.
3. **Card Inputs** – panel for <u>card-specific</u> inputs; these appear in the title area of a card for more condensed organization.

You define all three layout inputs using level two headers, much like rows. These features can be local to a page, or global across pages (requires a level one heading). Furthermore, they can be placed on any side of the page and can even be embedded into rows themselves for better clarity on what the input is affecting. This interactivity is provided by **htmlwidgets** (*R*), **Jupyter Widgets** (*Python*), or **Observable JS** (a dialect of *JavaScript*).

More flexible interactive components and drill down can be added using the following resources:

- **Shiny for R** – for dashboards that use knitr for computations (**requires deployment to a server**).
- **Shiny for Python** – for dashboards that use Jupyter for computations (**requires deployment to a server**).

`Shiny` is beyond the scope of this paper, but resources are attached to learn basic `Shiny` syntax and deploy a Quarto dashboard using `Shiny` (see Recommended Reading).

## DASHBOARD DEPLOYMENT AND UPDATES

### STATIC DASHBOARDS – <u>NO SERVER REQUIRED</u>

Static dashboards have no server-dependency, and thus can be published to any web server, including:

- **GitHub Pages** *(our method of choice).*
- **Quarto Pub.**
- **Posit Connect.**
- **Post Cloud.**
- **Confluence.**

Due to the rising popularity of GitHub/Git, GitHub Pages is our method of choice in this paper. GitHub Pages requires that the dashboard be rendered to a static file and shared as a GitHub repository. This can be accomplished by using the `quarto publish` command, but we really prefer to use the GitHub Desktop Application (GitHub Desktop | Simple collaboration from your desktop). Let's walk through the steps:

1. In your R project working directory, make sure your quarto markdown *(.qmd)* file is named **index.qmd.** This is required to publish to GitHub Pages. You also have a file named "_quarto.yml." It is essential to open this file to name your project and define an output directory for your rendered dashboard. We will be using the "docs" directory as shown below. Once setup, close the file and render your qmd document within RStudio. A static HTML of your dashboard should now generate inside the "docs" folder. Your "docs" folder should appear similar to Figure 4. This is the folder that we will select to render on GitHub Pages:

```
project:
  title: " Dashboarding with Quarto 1.4"
  output-dir: docs
editor: visual
```

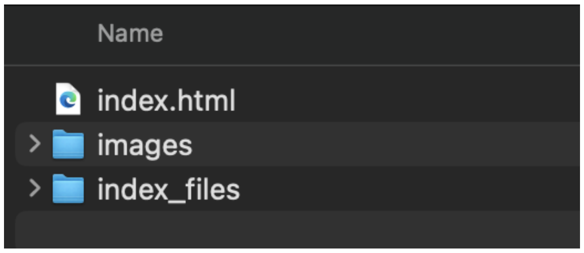


Figure 4. The directory organization for the "docs" folder. This will be used to render the dashboard using GitHub Pages.

2. Now install GitHub desktop, sign-in to your GitHub account, and add the R project repository (via "Add Existing Repository") to your GitHub, as shown in Figure 5. Navigate to your repository folder, open it, and hit select.

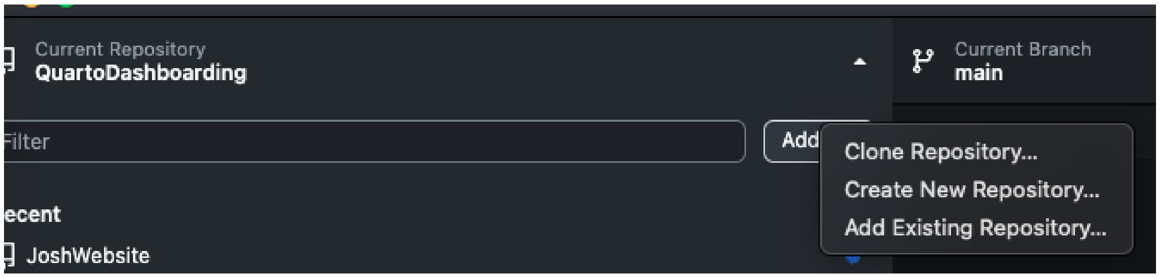
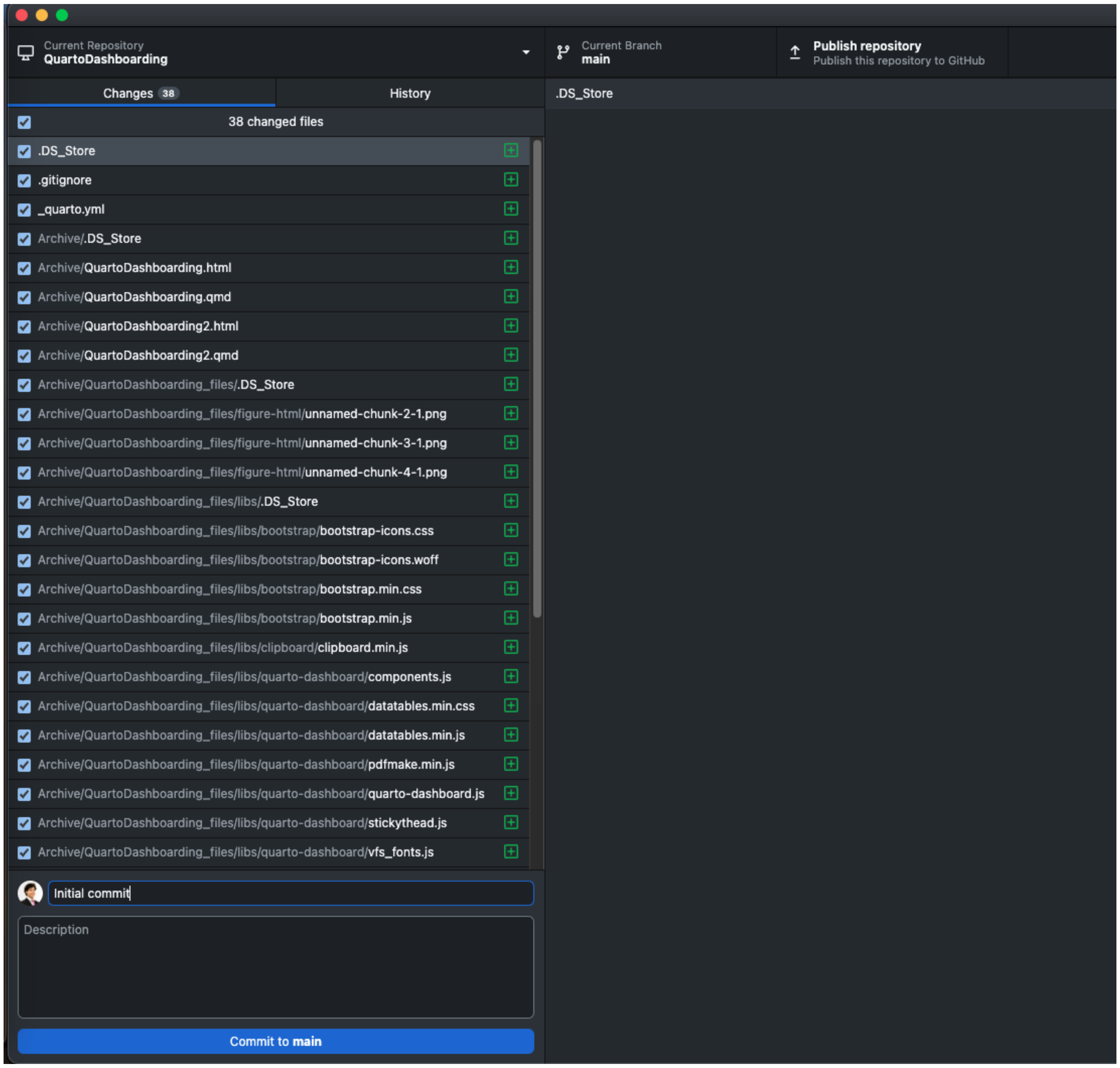


Figure 5. The user interface of GitHub Desktop, which is used to add a repository to GitHub, as well as commit/push/pull changes to projects.

3. You will now need to commit all your changes (which in this case is every single file) with a summary of what the changes are (initial commit in this case), and publish the repository, as shown in Figure 6. Use any name/description you desire, but make sure the code is not private.

GitHub desktop will detect any changes you make to the local files, and any made to the repository. You will use the application to commit/push any new changes you make, and “pull” any changes that are uploaded to the repository by other people if you are working on a group project or are part of an organization.



.DS_Store
.gitignore
_quarto.yml
Archive/.DS_Store
Archive/QuartoDashboarding.html
Archive/QuartoDashboarding.qmd
Archive/QuartoDashboarding2.html
Archive/QuartoDashboarding2.qmd
Archive/QuartoDashboarding_files/.DS_Store
Archive/QuartoDashboarding_files/figure-html/unnamed-chunk-2-1.png
Archive/QuartoDashboarding_files/figure-html/unnamed-chunk-3-1.png
Archive/QuartoDashboarding_files/figure-html/unnamed-chunk-4-1.png
Archive/QuartoDashboarding_files/libs/.DS_Store
Archive/QuartoDashboarding_files/libs/bootstrap/bootstrap-icons.css
Archive/QuartoDashboarding_files/libs/bootstrap/bootstrap-icons.woff
Archive/QuartoDashboarding_files/libs/bootstrap/bootstrap.min.css
Archive/QuartoDashboarding_files/libs/bootstrap/bootstrap.min.js
Archive/QuartoDashboarding_files/libs/clipboard/clipboard.min.js
Archive/QuartoDashboarding_files/libs/quarto-dashboard/components.js
Archive/QuartoDashboarding_files/libs/quarto-dashboard/datatables.min.css
Archive/QuartoDashboarding_files/libs/quarto-dashboard/datatables.min.js
Archive/QuartoDashboarding_files/libs/quarto-dashboard/pdfmake.min.js
Archive/QuartoDashboarding_files/libs/quarto-dashboard/quarto-dashboard.js
Archive/QuartoDashboarding_files/libs/quarto-dashboard/stickythead.js
Archive/QuartoDashboarding_files/libs/quarto-dashboard/vfs_fonts.js

Initial commit

Figure 6. The GitHub Desktop interface for committing and publishing repositories.

4. On the GitHub website (Your Repositories (github.com)), find your repository, open it, and click the “Settings” tab. On the left sidebar, navigate to Pages. Make sure the Source is set to “Deploy from a branch” the branch is set to “main” and the folder is set to “/docs” then hit save.
5. After a few moments, the GitHub Pages website will render, and a link will be displayed under the Pages tab. You can also display this link in your project description for public use.
6. Any future changes will need to be committed and pushed using the GitHub Desktop application. However, now that GitHub Pages is configured, it will automatically re-deploy the dashboard anytime it receives updates via commits/pushes, you just need to **make sure you render to docs before you commit.** If the data structure is unchanged (ex: the same AACT information is being queries on a set schedule), the updating of the data and subsequent refresh of the dashboard can be automated using GitHub Actions (built into GitHub) or other products such as Posit Connect WITHOUT the need to manually render each time.

TIPS AND TRICKS

Here's a few things we noticed that all users should keep in mind when working with Quarto 1.4 dashboarding:

- Not renaming the quarto markdown file (.qmd) to "index.qmd" can lead to severe issues when rendering and deploying the dashboard. Double check this file name if any issues are encountered. The actual name of the dashboard (that appears on the dashboard itself) can be whatever you want and is defined inside the quarto markdown file.
- The GitHub Pages section of the Quarto documentation does not utilize the GitHub Desktop Application (Quarto – GitHub Pages). In fact, most resources refer to terminal commands, but we do not recommend them due to added complexity. Try out the desktop application!
- When in doubt, check the Dashboard Examples (see Recommended Reading). Our full code and example are also linked below.
- It can help to have some familiarity with Quarto's other formats before attempting dashboarding. The syntax and structure follow very similar patterns as other formats, such as Quarto websites and reports.

CONCLUSION

The paper demonstrated how Quarto's cross-language support, combined with its new and improved dashboarding functionality, provides a comprehensive environment for data reporting. This integration is crucial for many industries where making informed decisions often depends on understanding complex datasets that update over time. By leveraging Quarto's new dashboarding tools, analysts from a variety of programming backgrounds can more effectively present their data, leading to improved decision-making processes and clearer communication of findings. Quarto 1.4 has an extraordinary opportunity to enhance open-source data visualization and reporting practices within a multitude of industries leaning toward open-source software.

REFERENCES

R Consortium (September 27, 2023). "Shiny App Successfully Reviewed by FDA CDER Staff (Pilot 2 Announcement 2)."

"Clinical Trials Transformation Initiative" (n.d.). *CTTI*, Available athttps://ctti-clinicaltrials.org/.

"Quarto - Quarto Dashboards" (n.d.). *Quarto*, Available athttps://quarto.org/docs/dashboards/.

Lafler, Kirk Paul; Joshua M. Horstman and Roger D. Muller (2019), "Building a Better Dashboard Using SAS® Base Software," Proceedings of the 2019 SouthEast SAS Users Group (SESUG) Conference.

ACKNOWLEDGMENTS

The authors thanks PharmaSUG for their support through the PharmaSUG New Professionals Scholarship Program. Joshua J. Cook thanks his mentors, Kirk Paul Lafler (@sasNerd), and Dr. Achraf Cohen, for their ongoing mentorship and support.

RECOMMENDED READING

- Common Pitfalls in Dashboard Design (perceptualedge.com)
- Quarto – Quarto Dashboards
- Quarto – Markdown Basics
- Shiny - Welcome to Shiny (posit.co)
- Quarto – Dashboards with Shiny for R
- Quarto – Dashboard Examples
- jjc54/QuartoDashboarding: PharmaSUG 2024 Dashboarding with Quarto 1.4 Example. (github.com)

- Understanding GitHub Actions - GitHub Docs
- Quarto
- *R for Data Science (2e)*

CONTACT INFORMATION

Joshua J. Cook, M.S., ACRP-PM, CCRC is a recent graduate of Wake Forest University (WFU) where he earned his Master of Science for Clinical Research Management. He is a current graduate student at the University of West Florida (UWF) studying Data Science while working at the university as an Adjunct Professor. Joshua worked in the field of clinical research for nearly three years, starting in neurology clinical trials and then specializing in orthopedic regenerative medicine as a Research Quality Analyst. He has published his undergraduate honors thesis, entitled "Endurance exercise-mediated metabolic reshuffle attenuates high-caloric diet-induced non-alcoholic fatty liver disease" in the *Annals of Hepatology* and has recently submitted several orthopedic research papers to various journals. He has also presented his research at over ten unique conferences at the local, state, and national levels with topics spanning from the impact of blood sugar on Alzheimer's Disease to publication metric tracking with R and Microsoft Power BI®. Joshua has developed a passion for bench-to-bedside research and aims to synthesize his knowledge of the biomedical sciences, clinical research, and data science to become a physician-scientist capable of integrating clinical care with clinical research in a way that maximizes evidence-based care options for his patients.

Kirk Paul Lafler is a consultant, developer, programmer, educator, and data scientist; and currently teaches SAS Programming and Data Management as a lecturer and adjunct professor in the Statistics Department at San Diego State University; provides project-based consulting and programming services to client organizations in a variety of industries including healthcare, life sciences, and business; and teaches "virtual" and "live" SAS, SQL, Python, Database Management Systems (DBMS), Excel, R, and cloud-based technology courses to users around the world. Kirk has years of development and programming experience, and specializes in SAS, Python, SQL, DBMS (e.g., Oracle, SQL-Server, Teradata, MySQL, MongoDB, PostgreSQL, AWS), Excel, R, and other software and tools. Currently, Kirk serves as the WUSS EC Open-Source Advocate and Coordinator and is actively involved with several proprietary and open-source software, DBMS, machine learning, cloud-computing user groups and conference committees. Kirk is the author of the popular PROC SQL: Beyond the Basics Using SAS, Third Edition (SAS Press. 2019) along with other technical books. He is also an Invited speaker, educator, keynote, and leader; and is the recipient of 28 "Best" contributed paper, hands-on workshop (HOW), and poster awards.

Your comments and questions are valued and encouraged. Contact the authors at:

Joshua J. Cook, M.S., ACRP-PM, CCRC

Educator / Graduate Student / Clinical Researcher / Data Scientist

Website: https://joshua-j-cook-portfolios.netlify.app/

Email: jcook0312@outlook.com

Kirk Paul Lafler, sasNerd

Consultant, Developer, Programmer, Data Scientist, Educator, and Author

Specializing in SAS® / Python / SQL / Database Management Systems / Excel / R / AWS / Cloud-based Technologies

E-mail: KirkLafler@cs.com

LinkedIn: https://www.linkedin.com/in/KirkPaulLafler/

Twitter: @sasNerd