PharmaSUG 2026 - Paper AP-233

Detecting Abnormal Page Breaks Using Grayscale Pixel-Density Analysis in R Shiny

Yi Guo, Pfizer Inc.

ABSTRACT

Unexpected page breaks in tables, listings, and figures (TLFs) are difficult to prevent because page capacity is hard to estimate in advance. For example, overlong or newly added footnotes and constrained column widths may lead to truncation, resulting in more rows or larger figures than can fit on a single page. Since each TLF has a unique layout, programmers must still visually review outputs case by case even after dataset validation. While page break issues are relatively easy to detect in shorter outputs, manual review becomes increasingly time-consuming and less reliable as outputs span many pages, making such issues more likely to be overlooked.

Inspired by color science, this paper presents a standalone R Shiny tool that applies grayscale pixel-density analysis to support detection of abnormal page breaks, assessing page layout through content distribution rather than visual appearance. The tool converts uploaded RTF or PDF files into page-level images, transforms them into grayscale representations, and evaluates pixel-density distributions to identify unusually large blank regions indicative of unintended page breaks. The approach does not rely on predefined layouts and is applicable across different TLF types.

This work demonstrates how image-based analytical techniques can be repurposed as a practical and reusable QC strategy for managing output-level quality risks. Within existing QC workflows, it provides near real-time, page-level visual feedback, improving manual review efficiency and reducing the chance of missing abnormal pages. At the same time, this type of interactive feedback is naturally well supported by R and Shiny-based workflows.

INTRODUCTION

During TLF layout review, checking multi-page outputs page by page without skipping can be time-consuming and may even lead to overlooking abnormal pages. Frequent program reruns caused by updated data or content changes further increase the burden of manual visual inspection.

A more efficient strategy is to screen all pages and filter out those that are very likely normal, leaving only potentially abnormal pages for reviewers to examine. Pixel-based image analysis provides an effective way to implement this idea. It has been widely used in domains such as medical imaging and industrial defect detection to identify structural anomalies. However, such

techniques have not been widely explored in TLF layout quality control. In this work, this methodology is adapted and implemented using R and Shiny. The rationale, key algorithmic logic, and implementation highlights are presented in this paper.

CONCEPTUAL FRAMEWORK

In this work, visual layout inspection is treated as a statistical detection problem. Each page is first converted into a grayscale where pixel intensity values range from 0 (black) to 255 (white) and then binarized. In the resulting image, textual and graphical elements appear as black pixels, while background regions appear as white pixels. This representation captures layout structure through pixel density patterns, which can be further quantified for anomaly detection.

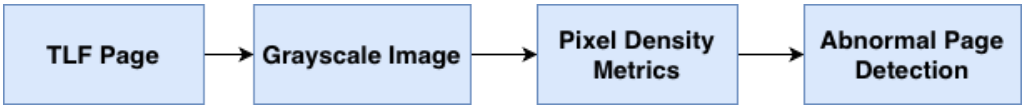

Figure 1. Conceptual framework for pixel-based detection in TLF layout quality control.

DESIGN CONSIDERATIONS FOR DETECTION

To improve robustness and computational efficiency, several practical considerations are included in the implementation.

Table 1. Detection Design Considerations

Category	Design Consideration	Purpose
Detection Scope	Prioritize detection of large blank regions at the bottom or right of a page, with full-page blank and ink coverage used as supporting references	Capture typical abnormal page break patterns
	Start scanning from page 2	Remove layout noise from title page and improve detection efficiency
Image Processing	Downsample images	Reduce computational cost and improve processing efficiency
	Crop page margins	Remove irrelevant regions and focus the analysis on the main content area
Reference Metrics	Use mean and standard deviation rather than robust statistics such as	Increase sensitivity to abnormal patterns in pixel density distributions

	median and median absolute deviation	

DETECTION SCOPE

(1) Blank Region

A key visual characteristic of abnormal page breaks is the presence of noticeable localized blank regions, rather than overall text or ink coverage. This is because ink density varies across different text content. Instead, focusing on localized blank regions provides a more reliable basis for detection.

Abnormal blank regions most commonly appear near the bottom of the page but may also occur along the right side depending on page orientation (portrait or landscape) or layout structure (Figure 2a).

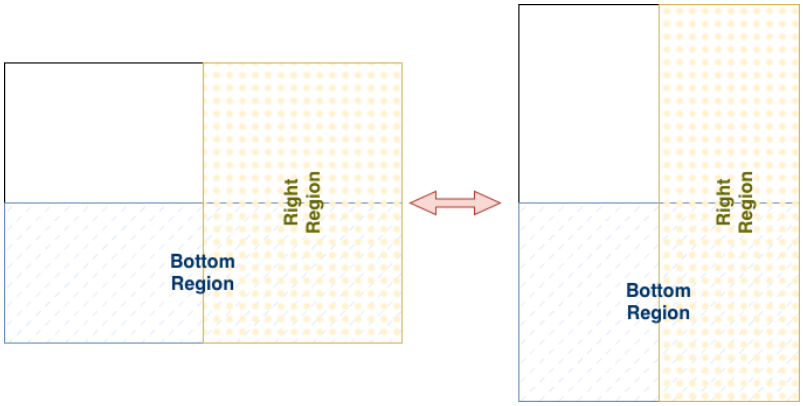

Figure 2a. Regions of interest for detecting abnormal TLF page breaks, with smaller regions flagging more pages as abnormal and larger regions fewer.

In this work, a 50% region is used as the default conservative threshold. Users can adjust this percentage based on specific needs.

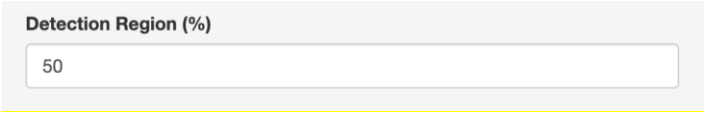

Figure 2b. Default detection percentage setting in the Shiny application interface.

(2) Page Scan Range

Excluding the first page from scanning can reduce noise and improve efficiency. This is because the first page often contains company logos, titles, or other layout elements and therefore provides limited information for detection. In addition, issues on the first page are rare, and

even if they occur, they can be identified and corrected immediately by reviewers during routine review.

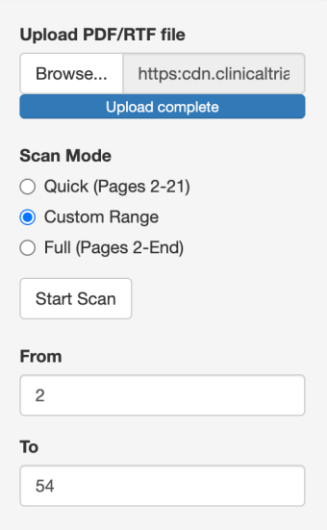

Figure 3. Scan mode options in the Shiny application interface: Quick (Pages 2-21), Custom Range, and Full (Pages 2-End, default). Scanning begins from the second page.

IMAGE PROCESSING CHOICES

(1) Downsampling

After rasterization, page images are downsampled (for example, to 25% of the original resolution) to reduce the number of pixels used in subsequent computations. This improves processing speed and reduces memory usage. Downsampling also introduces mild smoothing, where fine textual details are averaged while the overall layout structure is preserved. Because detection relies on large-scale density patterns rather than character-level details, this step improves computational efficiency without materially affecting detection results.

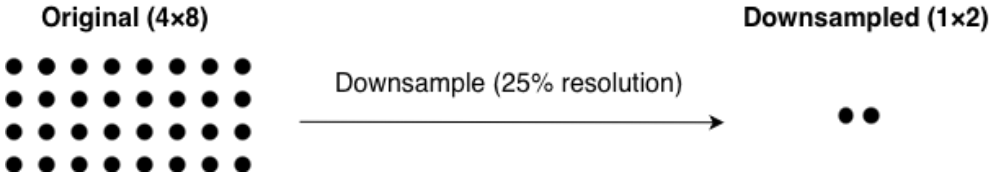

Figure 4a. Pixel count before and after downsampling.

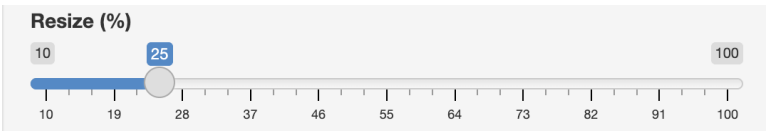

Figure 4b. Default image resize percentage setting in the app interface.

(2) Margin Cropping

TLF margins are typically blank and therefore contribute little information to layout detection. Removing margins can reduce unnecessary computation. Cropping after grayscale conversion simplifies processing because grayscale images contain a single channel whereas RGB images contain three. This is particularly beneficial when images remain at relatively high resolution.

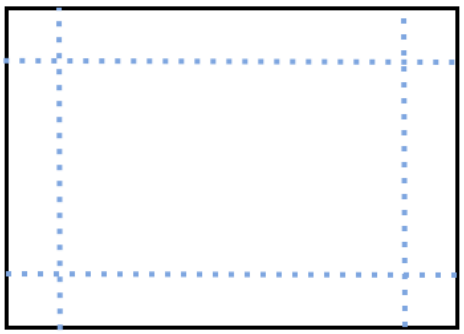

Figure 5a. Page margins and content area.

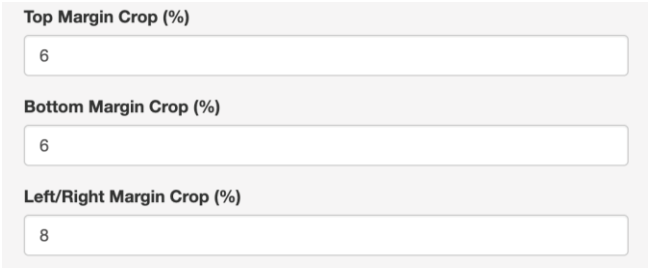

Figure 5b. Default margin cropping percentage in the application interface, adjustable by users based on the specific layout.

REFERENCE METRIC CONSIDERATIONS

Robust statistics such as the median, median absolute deviation (MAD), and upper quantiles were initially considered because they are commonly used in image analysis to reduce the impact of outliers, such as headers, logos, or isolated graphical elements. These methods work well when foreground elements make up a large portion of the image.

However, in typical TLF pages, dark pixels representing text or borders make up only a small portion of the page compared to the white background, except in figures with large filled regions. As a result, most pixels are close to white. This causes the median and upper quantiles to be close to 255 and MAD values to be very small, with little difference between normal and abnormal pages.

Although not robust to extreme values, the mean and standard deviation can still capture overall changes in grayscale intensity and variability. In this work, they are used as reference metrics to summarize the global pixel distribution.

PIXEL-DENSITY ANALYSIS METHOD

The proposed approach performs grayscale pixel-density analysis to summarize both white-space patterns and dark pixel projection profiles across pages.

In this work, bottom and right white ratios were prioritized because abnormal page breaks typically appear as localized blank regions. The overall white ratio was used as a supporting indicator to capture unusually sparse or blank pages, while horizontal and vertical darkness profiles were visualized to illustrate spatial content distribution across the page. Mean grayscale intensity and standard deviation (SD) were retained only as descriptive reference metrics.

Table 2. Pixel-Density Metrics Used for Page Layout Analysis

Metric[a]	Category	Description	Purpose	Detection Role
Mean	Pixel intensity statistic (global)	Average grayscale intensity of page pixels	Overall brightness	Descriptive reference
SD	Pixel intensity statistic (global)	Variation of grayscale intensity values	Brightness variability	Descriptive reference
Overall white ratio	Pixel density metric (global)	Proportion of pixels exceeding the white threshold[b]	Blank coverage across the page	Supporting indicator
Bottom white ratio	Spatial white-density metric (local)	White pixel proportion in the bottom region	Detect localized abnormal blank region	Primary detection indicator
Right white ratio	Spatial white-density metric (local)	White pixel proportion in the right region	Detect localized abnormal blank region	Primary detection indicator
Horizontal darkness profile	Spatial pixel-density metric (local)	Mean darkness across rows derived from grayscale pixel values	Describe vertical content placement	Supporting indicator

Vertical darkness profile	Spatial pixel-density metric (local)	Mean darkness across columns derived from grayscale pixel values	Describe horizontal content placement	Supporting indicator

a. Mean, SD, and overall white ratio represent global statistics, whereas bottom and right white ratios represent spatial statistics.

b. Pixels with intensity greater than 245 are treated as white pixels. This threshold was determined empirically based on inspection of multiple rasterized TLF pages, where background pixels typically fall within the range of 245 to 255. The value can be adjusted in the application. All metrics are calculated after margin cropping to ensure that the analysis focuses on the central content region.

The following sections explain why these metrics are selected and how they are used in the detection hierarchy.

MEAN AND STANDARD DEVIATION

Although robust statistics were not informative in this context, the mean grayscale intensity and its standard deviation still provide useful descriptive references.

Let g_i denote the grayscale intensity of pixel i, and N the total number of pixels on a page. The mean grayscale intensity and its standard deviation are defined as

$$\mu = \frac{1}{\mathrm{N}} \sum_{i=1}^{N} g_i$$

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (g_i - \mu)^2}$$

These statistics summarize the overall grayscale distribution but do not capture where blank areas occur on the page. Pages with blank spaces either concentrated in certain regions or dispersed across the page may produce similar values. In addition, some pages naturally contain little text but are still structurally correct.

WHITE RATIO METRICS

To better capture layout structure, the overall white ratio was introduced. This metric helps identify unusually sparse or blank pages and reflects the general balance between page content and background. Let $W_{i,j}$ denote a binary indicator for pixel (i, j):

$$W_{i,j} = \begin{cases} 1, & if\ pixel\ (i,j)\ is\ classfied\ as\ white \\ 0, & otherwise \end{cases}$$

Let H and W denote the page height and width in pixels. The overall white ratio is defined as

$$WR_{overall} = \frac{1}{HW}\sum_{i=1}^{H}\sum_{j=1}^{W} W_{i,j}$$

However, as a global measure, it still cannot determine where blank regions occur.

Therefore, two additional metrics are introduced to measure white density near the bottom and right boundaries of the page. Let R denote a selected boundary region of the page (e.g., bottom or right). The white ratio for region R is defined as

$$WR_R = \frac{1}{|R|}\sum_{(i,j)\in R} W_{i,j}\,, \quad \mathrm{R} \in \{\mathrm{bottom}, \mathrm{right}\}$$

where $|R|$ represents the number of pixels within the region. These metrics serve as primary indicators of abnormal page breaks.

PROJECTION PROFILES OF DARK PIXELS

Projection profiles of dark pixels provide a visual summary of content placement along the vertical and horizontal directions of a page and help show page layout patterns. Let $x_{i,j}$ denote the gray scale intensity of pixel (i,j) ranging from 0 (black) to 255 (white). The corresponding darkness value is defined as

$$d_{i,j} = 255 - x_{i,j}$$

Let H and W denote the page height and width in pixels. The mean darkness projection profile is defined as

$$P_k = \begin{cases} \frac{1}{W}\sum_{j=1}^{W} d_{i,j}, & k = vertical \\ \frac{1}{H}\sum_{i=1}^{H} d_{i,j}, & k = horizontal \end{cases}$$

These profiles help reveal abrupt blank regions associated with abnormal page breaks.

Figure 6 shows an example of an abnormal page break where rows from the previous page spill over onto the next page. The analysis is based on a PDF file (portrait orientation) converted from an RTF file (landscape orientation). A large blank region appears on the right side of the page. The horizontal ink distribution initially varies on the left but then drops sharply to near zero toward the right boundary, forming a flat line.

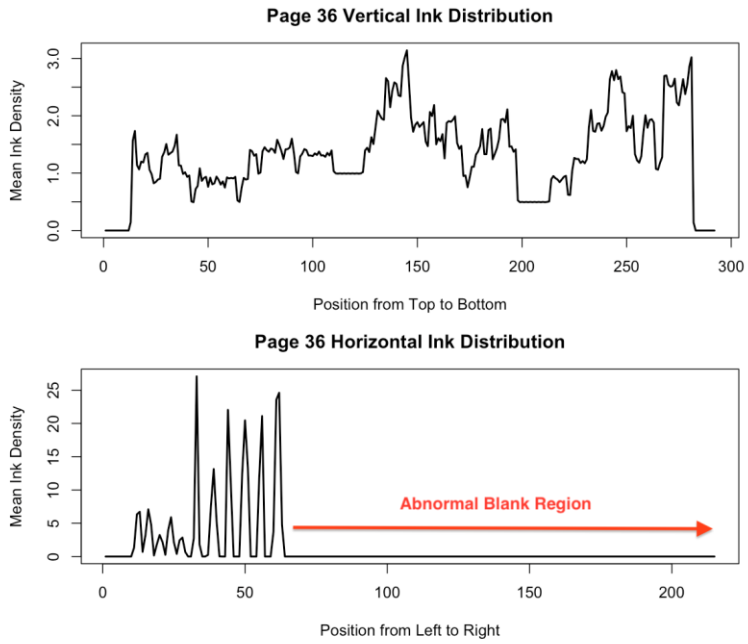

Figure 6. Vertical and horizontal projection profiles for an abnormal page break.

R SHINY IMPLEMENTATION

Based on the framework, design considerations, and statistical methods described above, the grayscale pixel-density analysis is implemented in R Shiny. The workflow is shown in Figure 7.

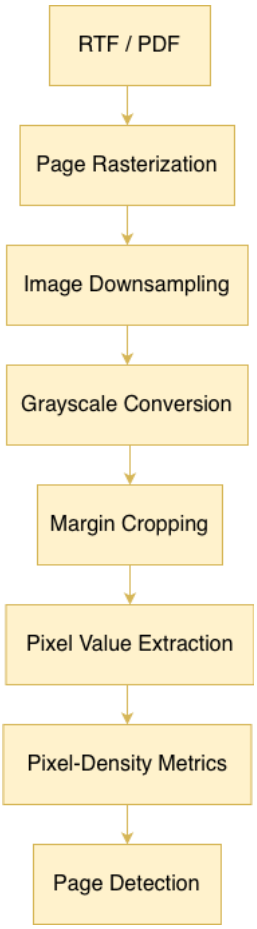

Figure 7. Stepwise workflow implemented in R Shiny for detecting abnormal page breaks.

APPLICATION OVERVIEW

Figure 8 shows the overall layout of the R Shiny app. The sidebar panel on the left allows users to upload files, select the scan mode (full scanning by default), and adjust parameters. Once configured, clicking the “Start Scan” button on the left. Then the app displays the results in the main panel on the right, which is organized into three layers:

1. The top layer provides a page viewer, allowing users to freely browse all pages or focus on those flagged as suspicious.
2. The middle layer presents QC metrics and page status (OK or Check).
3. The bottom layer displays horizontal and vertical dark pixel density projections as supporting evidence.

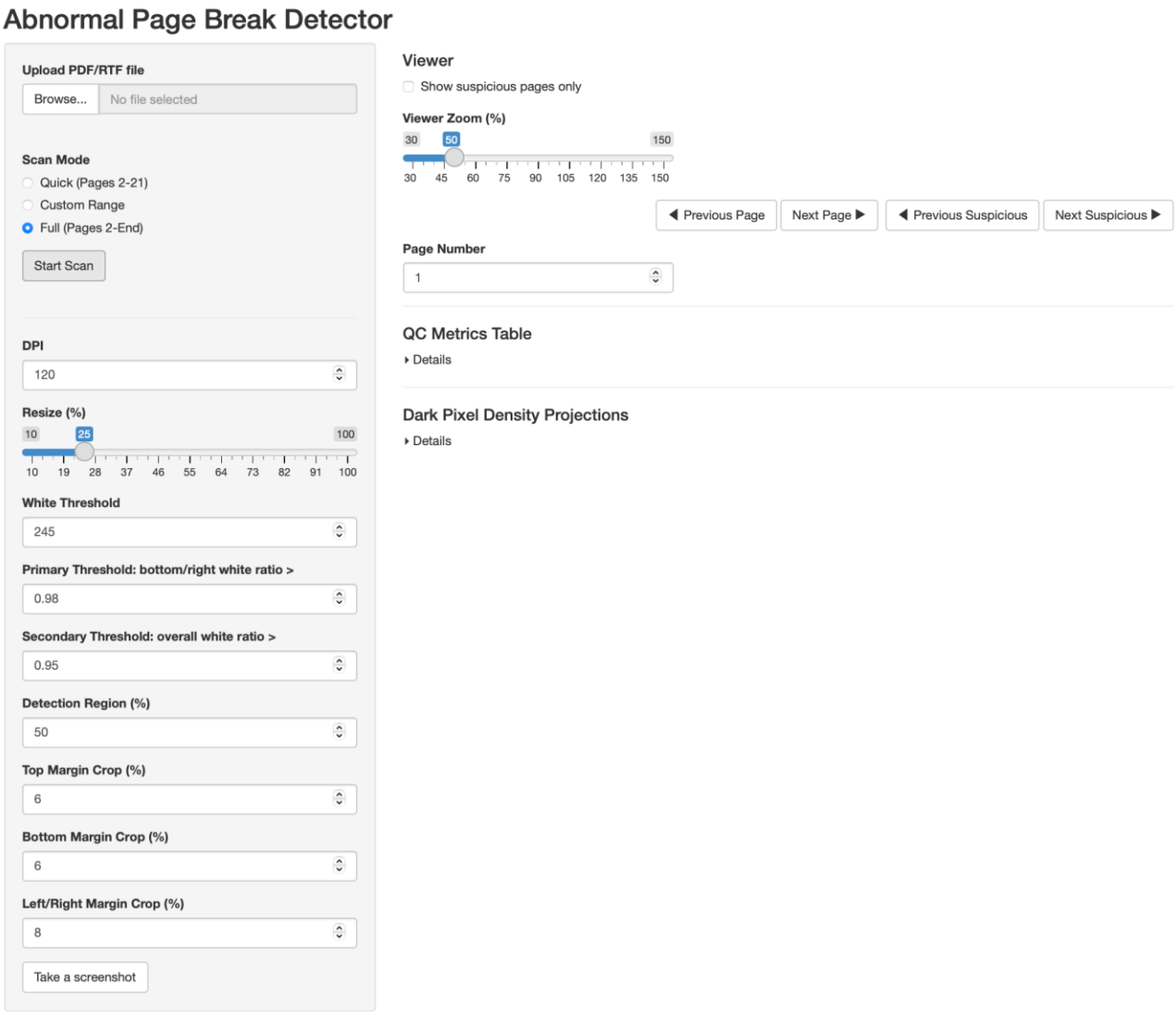

Figure 8. Application interface in initial state (before file upload).

R SHINY CODE STRUCTURE

The application follows the standard Shiny UI-server architecture. The UI defines inputs and outputs, while the server performs the main processing and analysis by calling helper functions for file format conversion, image processing, and metric calculations.

```
# Packages
library(...)

### --- Helper Functions --- ###
...
```

```r
### --- UI --- ###
ui <- fluidPage(
  titlePanel(...),
  sidebarLayout(
    sidebarPanel(...),    #file upload and parameter settings
    mainPanel(...)        #page viewer/QC table/projection plots
  )
)

### --- Server --- ###
server <- function(input, output, session) {
  #file processing
  #page-level analysis
  #abnormal page detection
  #results rendering (viewer, QC table, and projection plots)
}

# Launch App
shinyApp(ui = ui, server = server)
```

HELPER FUNCTIONS

Each step in the pipeline shown in Figure 7 is implemented in R, as presented below. The {magick} package is used for rasterization, downsampling, grayscale conversion, and cropping, and {pdftools} is used to retrieve PDF page numbers. Helper functions are organized hierarchically, with `analyze_page()` as the main page-level analysis function (Figure 9).

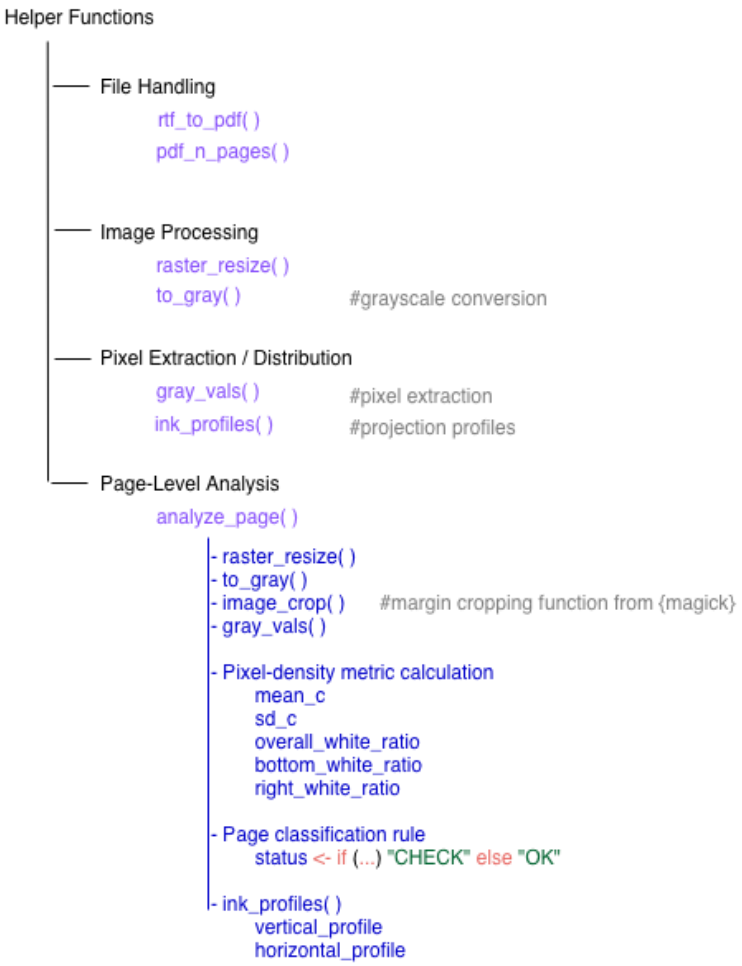

Figure 9. Structure of helper functions in the Shiny application.

(1) RTF/PDF File Handling

RTF files are converted to PDF before rendering, while PDF files are rendered directly. For RTF inputs, a temporary copy is created and processed with LibreOffice in headless mode. The converted PDF is saved to a temporary directory and then used for downstream page rendering.

```
#convert RFT to PDF (LibreOffice)
rtf_to_pdf <- function(rtf_path) {
  soffice <- "/Applications/LibreOffice.app/Contents/MacOS/soffice"

  tmp <- tempfile(fileext = ".rtf")
  file.copy(rtf_path, tmp, overwrite = TRUE)
```

```
  out_dir <- tempfile("rtf_to_pdf_")
  dir.create(out_dir, recursive = TRUE, showWarnings = FALSE)

  system2(
    soffice,
    args = c(
      "--headless",
      "--nologo",
      "--nolockcheck",
      "--convert-to",
      "pdf:writer_pdf_Export",
      "--outdir",
      out_dir,
      normalizePath(tmp, winslash = "/", mustWork = TRUE)
    )
  )

  pdfs <- list.files(out_dir, pattern = "\\.pdf$", full.names = TRUE)
  pdfs[1]
}

#get total number of pages in a PDF file
pdf_n_pages <- function(path){
  if (requireNamespace("pdftools", quietly = TRUE)) {
    as.integer(pdftools::pdf_info(path)$pages)
  } else {
    NA_integer_
  }
}
```

(2) Page Rasterization and Image Downsampling

Page rasterization converts each PDF page into an image. Image downsampling then reduces the resolution for faster processing. Both steps are implemented in the helper function `raster_resize()`.

```
raster_resize <- function(path, page, dpi, resize_pct){
  #page rasterization
  img <- image_read_pdf(path, pages = page, density = dpi)

  #image downsampling
  if (!is.null(resize_pct) && resize_pct > 0 && resize_pct < 100) {
    img <- image_resize(img, paste0(resize_pct, "%"))
  }
  img
}
```

(3) Grayscale Conversion

This step converts the RGB image to a grayscale image using the `image_convert()` function from the {magick} package.

```
to_gray <- function(img){
  image_convert(img, colorspace = "gray")
}
```

(4) Margin Cropping

The margins are cropped using the `image_crop()` function from the {magick} package to remove the top, bottom, left, and right edges of the page.

```
analyze_page <- function(...){
  img <- raster_resize(path, page, dpi, resize_pct)
  g <- to_gray(img)

  info <- image_info(g)
  w <- info$width
  h <- info$height

  #crop top (% of height)
  top_h  <- max(1, floor(h * top_crop_thr / 100))
  #crop bottom (% of height)
  bot_h  <- max(1, floor(h * bot_crop_thr / 100))
  #crop left/right (% of width)
  side_w <- max(1, floor(w * side_crop_thr / 100))

  #center region for analysis
  c_img <- image_crop(
    g,
    geometry_area(
      w - 2 * side_w,
      h - top_h - bot_h,
      side_w,
      top_h
    )
  )

  ...
}
```

(5) Pixel Value Extraction

The `image_data()` function from {magick} returns the grayscale image as a three-dimensional array. Since the image is already in grayscale, only one channel is present. Pixel values are stored as hexadecimal strings (for example, "A1", "FF", or "00").

The array is flattened into a vector using `as.vector()`, and the values are converted to decimal integers using `strtoi(..., 16L)`. For example, "FF" becomes 255 and "00" becomes 0, giving pixel intensities from 0 (black) to 255 (white).

```
gray_vals <- function(gray_img){
  a <- image_data(gray_img, channels = "gray")
  strtoi(as.vector(a[1, , ]), 16L)
}
```

(6) Pixel-Density Metrics

After grayscale pixel values are extracted from the cropped center region, they are used to compute a predefined set of pixel-density metrics.

```
analyze_page <- function(...) {

  #margin cropping
  ...

  #center region as numeric pixel values
  center <- gray_vals(c_img)

  #reference metrics from cropped center region
  mean_c <- mean(center)
  sd_c <- sd(center)

  #secondary indicator from cropped center region
  overall_white_ratio <- mean(center > white_thr)

  #primary indicators from cropped center region
  c_info <- image_info(c_img)
  cw <- c_info$width
  ch <- c_info$height

  bot <- gray_vals(image_crop(c_img, geometry_area(
    cw,
    floor(ch * pct_region / 100),
    0,
    ch - floor(ch * pct_region / 100)
  )))

  rgt <- gray_vals(image_crop(c_img, geometry_area(
    floor(cw * pct_region / 100),
    ch,
    cw - floor(cw * pct_region / 100),
    0
  )))

  bottom_white_ratio <- mean(bot > white_thr)
  right_white_ratio  <- mean(rgt > white_thr)

  ...
}
```

Vertical and horizontal projections are computed in a separate helper function `ink_profiles()` and then used within `analyze_page()`.

```
# Step 1
ink_profiles <- function(gray_img){
  a <- image_data(gray_img, channels = "gray")

  gray <- apply(a[1, , ], c(1, 2), function(x) strtoi(x, 16L))
  ink <- 255 - gray

  list(
    vertical = apply(ink, 2, mean),    #top to bottom
    horizontal = apply(ink, 1, mean)   #left to right
  )
}


# Step 2
analyze_page <- function(...) {
  #margin cropping
  ...
  c_img <- image_crop(...)

  #vertical and horizontal ink profiles from grayscale image
  prof <- ink_profiles(c_img)

  list(
    ...
    vertical_profile = prof$vertical,
    horizontal_profile = prof$horizontal
  )

  ...
}
```

(7) Page Detection

Page detection is implemented using a rule-based classification based on the calculated pixel-density metrics. The primary indicators (bottom/right white ratio) detect potential abnormal page layouts, while the supporting indicator (overall white ratio) provides confirmation. Pages meeting the detection criteria are labeled "CHECK"; otherwise they are labeled "OK".

```
analyze_page <- function(...) {
  ...
  status <- if (
    bottom_white_ratio > sus_thr ||
    right_white_ratio  > sus_thr ||
    overall_white_ratio > w_thr
  ) "CHECK" else "OK"
  ...
}
```

In this work, the thresholds (sus_thr and bg_thr) are set to 0.98 and 0.95, respectively, and can be adjusted in the Shiny app interface as needed.

UI

The user interface uses a standard layout, with a sidebar on the left and a main panel on the right. The sidebar includes file upload and parameter controls, while the main panel displays the rendered pages, QC results table, and projection plots. Detailed code is provided in Appendix 1.

```
ui <- fluidPage(
  titlePanel(...),

  sidebarLayout(

    ### --- Input Controls --- ###
    sidebarPanel(
      ...  #file upload and parameter settings
    ),

    ### ---  Output Display --- ###
    mainPanel(
      ...
      uiOutput("viewer_ui"),    #page viewer
      tableOutput("qc_table"), #QC metrics table
      plotOutput("proj_plot")  #projection plots
      ...
    )
  )
)
```

SERVER

The server coordinates file processing, page selection, page-level analysis, and result rendering. In the server code, we highlight the `analyze_page()` function because it performs the core pixel density analysis. Its outputs are used to generate the QC metrics table and projection plots for abnormal page detection.

```
server <- function(input, output, session){

  ### --- File Handling --- ###
  pdf_path <- reactive({
    req(input$file)
    ext <- tolower(tools::file_ext(input$file$name))

    if (ext == "pdf") {
      input$file$datapath
    } else if (ext == "rtf") {
      rtf_to_pdf(input$file$datapath)
    } else {
```

```
    stop("Only PDF or RTF file is supported")
  }
})

n_pages <- reactive({ pdf_n_pages(pdf_path()) })

...

### --- Scan Range Selection --- ###
scan_pages <- reactive({
  n <- n_pages()

  if (input$scan_mode == "quick") {
    ...
  } else if (input$scan_mode == "range") {
    ...
  } else {
    ...
  }
})


### --- Page-level QC Analysis --- ###
qc_cache <- reactiveVal(NULL)

qc_df <- eventReactive(input$run, {
  path  <- pdf_path()
  pages <- scan_pages()

  lst <- lapply(pages, function(p){
    analyze_page(
        path = path,
        page = p,
        dpi = input$dpi,
        resize_pct = input$resize_pct,
        pct_region = input$pct_region,
        white_thr = input$white_thr,
        sus_thr = input$sus_thr,
        w_thr = input$w_thr,
        top_crop_thr = input$top_crop_thr,
        bot_crop_thr = input$bot_crop_thr,
        side_crop_thr = input$side_crop_thr
    )
  })

  qc_cache(lst)

  do.call(rbind, lapply(lst, function(x) {
    data.frame(
     Page = x$page,
    'Bottom White Ratio' = x$bottom_white_ratio,
    'Right White Ratio' = x$right_white_ratio,
    'Overall White Ratio' = x$overall_white_ratio,
    'Mean*' = x$mean_c,
    'SD*' = x$sd_c,
     Status = x$status,
     stringsAsFactors = FALSE,
```

```
      check.names = FALSE
    )
  }))
})

output$qc_table <- renderTable({
  req(qc_df())
  qc_df()
})


### --- Suspicious Page Summary --- ###
sus_pages <- reactive({
  df <- qc_df()
  df$Page[df$Status == "CHECK"]
})

output$sus_ui <- renderUI({
  sp <- sus_pages()

  if (!length(sp)) return(tags$div("No suspicious pages"))

  tagList(
    tags$div(paste0("Suspicious pages: ", paste(sp, collapse = ", "))),
    selectInput("sus_pick", "Select suspicious page", choices = sp,
                selected = sp[1]),
    actionButton("goto_sus", "Go to Viewer")
  )
})

observeEvent(input$goto_sus, { ... })


### --- Viewer navigation --- ###
viewer_list <- reactive({
  if (!isTRUE(input$sus_only)) {
    ...
  } else {
    sus_pages()
  }
})

viewer_idx <- reactiveVal(1)

observeEvent(input$viewer_page, { ... })
observeEvent(input$goto_sus, { ... })
observeEvent(input$prev_page, { ... })
observeEvent(input$next_page, { ... })

...

### --- Page Rendering --- ###
output$viewer_ui <- renderUI({
  req(pdf_path())

  #determine page to display
  if (isTRUE(input$sus_only)) {
```

```
      ...
    } else {
      ...
    }

    #render selected page
    img <- tryCatch(
      raster_resize(pdf_path(), p, input$dpi, input$resize_pct),
      error = function(e) NULL
    )
    if (is.null(img)) return(tags$div(...))

    #display page image
    ...
  })


  ### --- Projection Plotting --- ###
  output$proj_plot <- renderPlot({
    req(qc_cache())
    lst <- qc_cache()
    p <- as.integer(input$pro_page)

    if (is.na(p) || p < 1) return(NULL)

    i <- which(vapply(lst, function(x) x$page, integer(1)) == p)
    if (!length(i)) return(NULL)

    v <- lst[[i]]$vertical_profile
    h <- lst[[i]]$horizontal_profile

    ...

    plot(v,
         type = "l",
         main = paste("Page", p, "Vertical Ink Distribution"),
         ...)
    plot(h,
         type = "l",
         main = paste("Page", p, "Horizontal Ink Distribution"),
         ...)
  })

}
```

APPLICATION EXAMPLES

This section presents two example scenarios to demonstrate how the application behaves under different conditions.

EXAMPLE 1: INVALID FILE INPUT

When a file other than PDF or RTF is uploaded, the application displays error message, as shown in Figure 10.

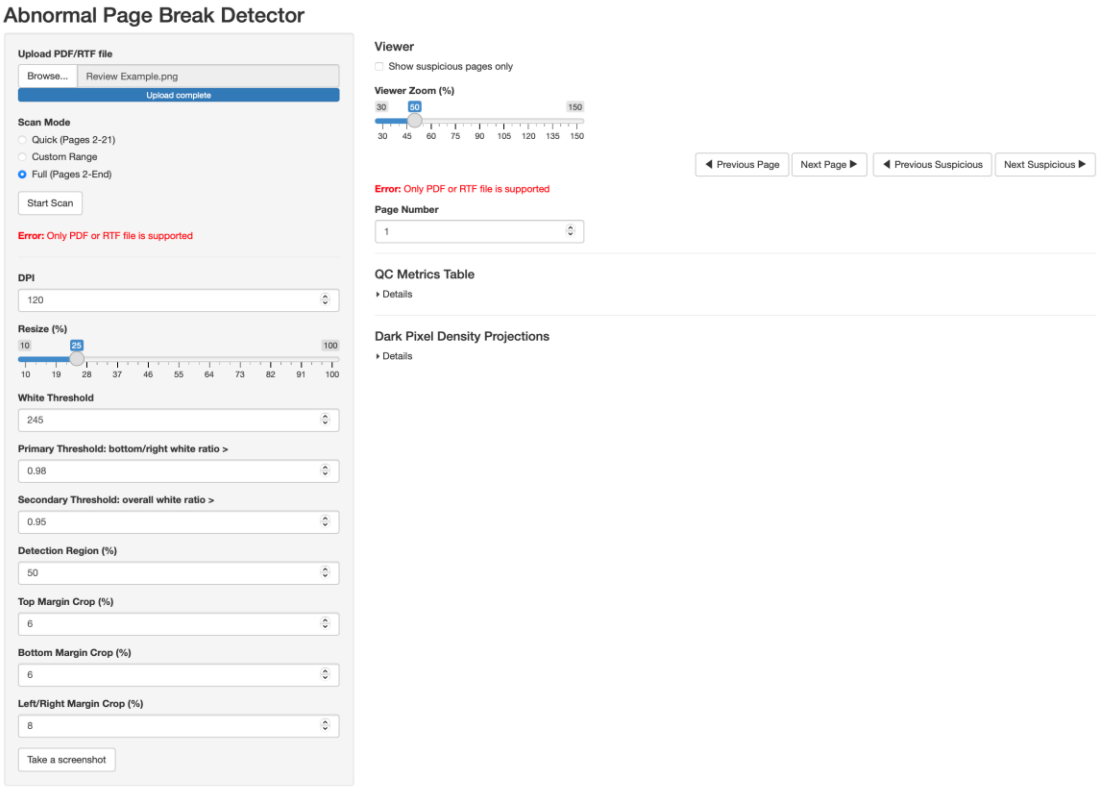

Figure 10. Error messages displayed in the application when a non-PDF/RTF file is uploaded, and processing is not performed.

EXAMPLE 2: ABNORMAL AND NORMAL PAGES

In this example, a mock TLF RTF file is uploaded to demonstrate the normal workflow of the application. Figures 11a and 11b show examples of an abnormal page and a normal page.

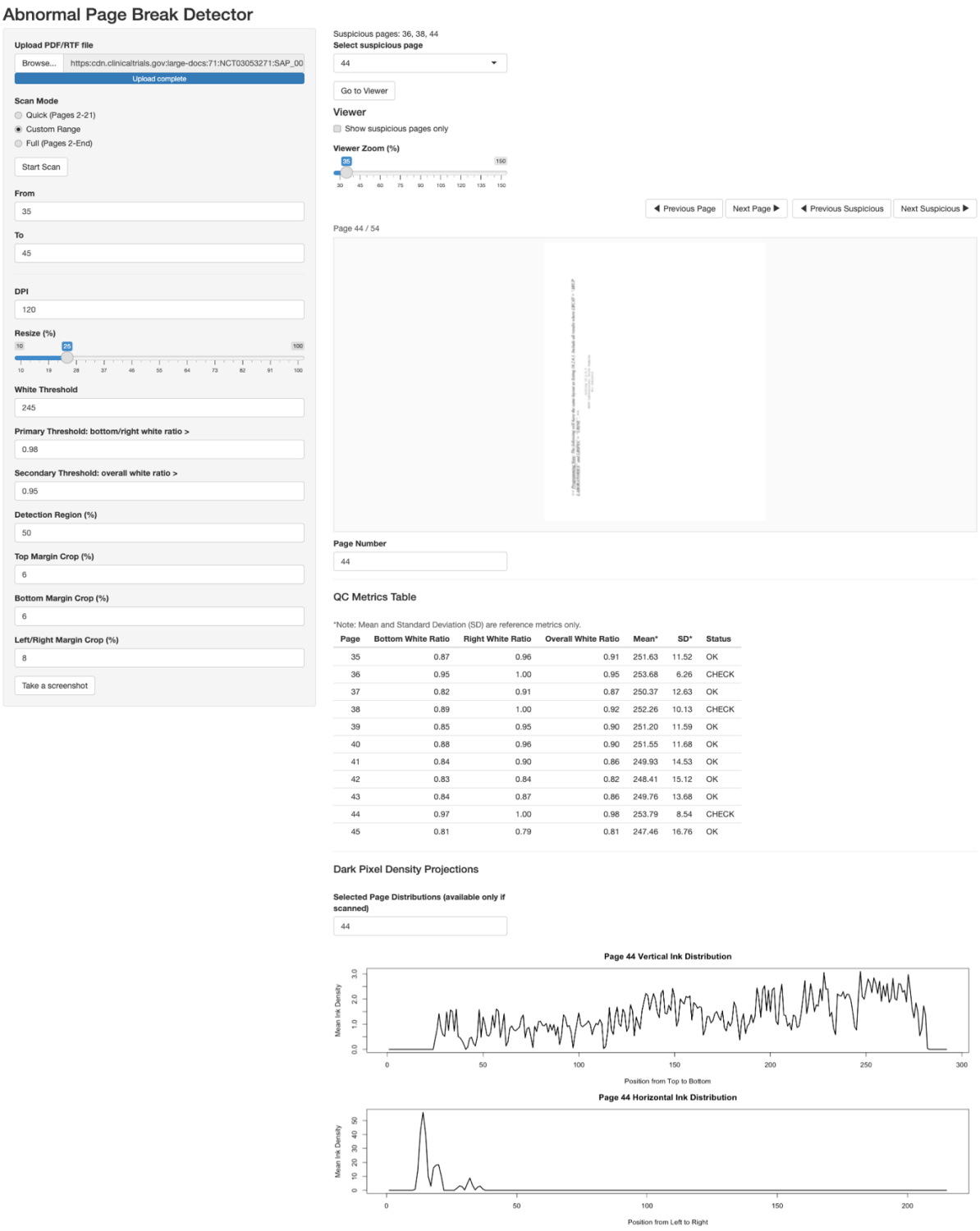

Figure 11a. Application output showing a detected abnormal page break on Page 44.

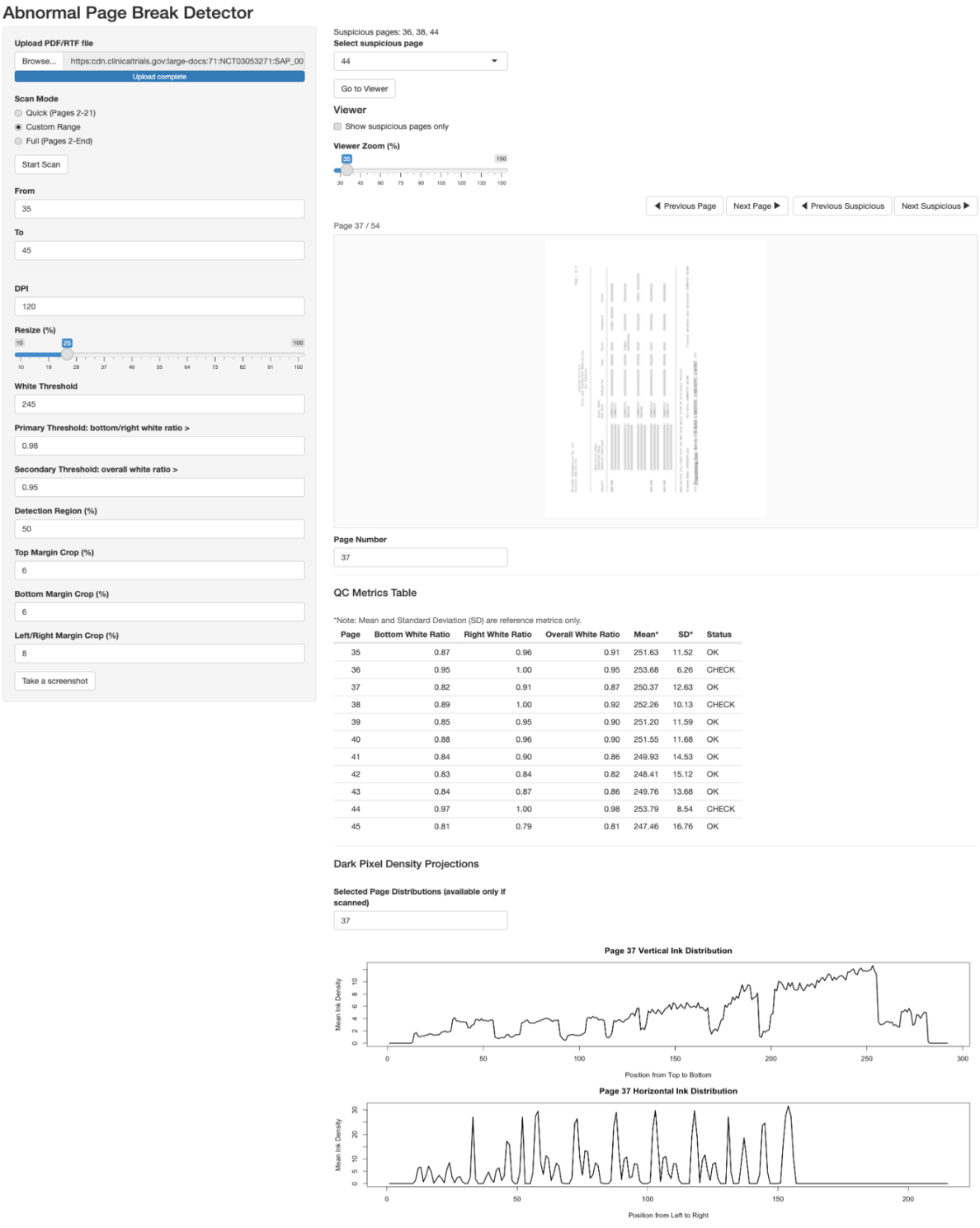

Figure 11b. Application output showing a normal page on Page 37.

We first upload an RTF file and specify a scan range (e.g., pages 35 to 45). After clicking “Start Scan”, the main panel displays: the list of suspicious pages; the page preview in the Viewer (page 1 by default); the QC metrics table for all pages within the selected range; and the dark pixel density projection plots for a selected page (page 2 by default).

The page number in the Viewer and the Plot layers are independent. Clicking “Go to Viewer” navigates to the selected suspicious page in the Viewer and updates the plots accordingly.

PRACTICAL RECOMMENDATIONS

It is recommended to perform an initial scan with a lower resizing level to improve efficiency and speed. If more detailed inspection is needed, the resizing percentage can be increased for smaller, selected page ranges to improve detection sensitivity.

Several parameters in the framework are configurable, including thresholds for overall, bottom, and right white-space ratios, as well as detection region and margin cropping percentages. These parameters can be adjusted based on company standards or study-specific layouts.

FUTURE DIRECTIONS

Future work may explore more adaptive approaches for determining thresholds to better accommodate variability across companies and document layouts. For example, machine learning models trained on historical TLF outputs could be used to estimate appropriate threshold values, reducing reliance on fixed rules.

CONCLUSION

This work demonstrates how practical QC challenges can be addressed through simple yet robust statistical reasoning, grayscale pixel-density analysis derived from color science principles, and interactive R Shiny implementation. It highlights how document structure and visual signal interpretation can be extended to QC strategies, providing a practical and scalable solution to improve the efficiency and reliability of TLF layout review.

REFERENCES

Otsu N (1979). A threshold selection method from gray-level histograms. *Automatica*, 11, 285–296.

Gonzalez RC (2009). *Digital Image Processing*. Pearson Education India.

ScienceDirect (n.d.). Gray-level distribution. https://www.sciencedirect.com/topics/computer-science/gray-level-distribution (accessed March 27, 2026)

Wikipedia (n.d.). Thresholding (image processing). https://en.wikipedia.org/wiki/Thresholding_(image_processing) (accessed March 27, 2026)

Ooms J (n.d.). *pdftools: Text Extraction, Rendering and Converting of PDF Documents*. https://cran.r-project.org/package=pdftools (accessed March 27, 2026)

Ooms J (n.d.). *magick: Advanced Graphics and Image Processing in R*. https://cran.r-project.org/package=magick (accessed March 27, 2026)

Urbanek S (n.d.). *base64enc: Tools for Base64 Encoding*. https://cran.r-project.org/package=base64enc (accessed March 27, 2026)

Attali D (n.d.). *shinyscreenshot: Capture Screenshots of Shiny Applications*. https://cran.r-project.org/package=shinyscreenshot (accessed March 27, 2026)

CONTACT INFORMATION

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

Yi Guo
Pfizer Inc.
yi.guo@pfizer.com

APPENDIX 1. SHINY USER INTERFACE (UI) CODE

```
ui <- fluidPage(

  tags$style(HTML("...")), #custom CSS

  titlePanel("Abnormal Page Break Detector"),

  sidebarLayout(
    sidebarPanel(
      fileInput("file", "Upload PDF/RTF file"),
      radioButtons(
        "scan_mode",
        "Scan Mode",
        choices = c(
          "Quick (Pages 2-21)" = "quick",
          "Custom Range" = "range",
          "Full (Pages 2-End)" = "full"
        ),
        selected = "full"
      ),
      actionButton("run", "Start Scan"),
      br(),
      br(),
      uiOutput("range_ui"),
      tags$hr(),

      #rendering and threshold parameters
      numericInput("dpi", "DPI", 120, ...),
      sliderInput("resize_pct", "Resize (%)", 10, 100, 25),
      numericInput("white_thr", "White Threshold", 245, 0, 255),
      numericInput(
        "sus_thr",
        "Primary Threshold: bottom/right white ratio >",
        0.98, 0, 1, 0.01),
      numericInput(
        "w_thr",
        "Secondary Threshold: overall white ratio >",
        0.95, 0, 1, 0.01),
      numericInput("pct_region", "Detection Region (%)", 50, 0, 100, 1),
      numericInput("top_crop_thr", "Top Margin Crop (%)", 6, 0, 100),
      numericInput("bot_crop_thr", "Bottom Margin Crop (%)", 6, 0, 100),
      numericInput("side_crop_thr", "Left/Right Margin Crop (%)", 8, 0, 100),
    ),

    mainPanel(
      div(class = "top-sus", uiOutput("sus_ui")),

      #top layer: page viewer
      h4("Viewer"),
      checkboxInput("sus_only", "Show suspicious pages only", FALSE),
      sliderInput("viewer_zoom", "Viewer Zoom (%)", 30, 150, 50, 5),
      div(
        class = "...",
        div(
          actionButton("prev_page", "◀ Previous Page"),
```

```
          actionButton("next_page", "Next Page ▶")
        ),
        div(
          actionButton("prev_sus", "◀ Previous Suspicious"),
          actionButton("next_sus", "Next Suspicious ▶")
        )
      ),
      uiOutput("viewer_ui"),
      div(class = "viewer-footer", numericInput("viewer_page", "Page Number",
                1, min = 1)),

      hr(class = "section-gap"),

      #middle layer: QC metrics table
      h4("QC Metrics Table"),
      tags$details(uiOutput("note"), tableOutput("qc_table")),

      hr(),

      #bottom layer: projection plots
      h4("Dark Pixel Density Projections"),
      tags$details(
        numericInput(
          "pro_page",
          "Selected Page Distributions (available only if scanned)",
          2,
          min = 1
        ),
        plotOutput("proj_plot", height = "500px")
      ),

      imageOutput("preview", width = "50%", height = "200")

    )
  )
)
```