# Mental Health Analytics Dashboard Using R Shiny

## 1. Title of the Project

**Mental Health Analytics Dashboard Using R Shiny: An Interactive Platform for Data-Driven Mental Health Analysis and Predictive Modeling by**

**Aarya Desai, Aditi Malik, Maxwell Fernandes and Joseph Jonathan Fernandes.**

## 2. Abstract

Mental health data analysis presents significant challenges due to data complexity, varied formats, and the need for specialized analytical skills. This project develops a comprehensive interactive dashboard using R Shiny framework that democratizes mental health data analytics by providing a no-code environment for exploration, visualization, and predictive modeling.

The system supports dynamic CSV dataset loading, automated binary target detection, comprehensive exploratory data analysis (EDA), preprocessing pipelines, and machine learning model training using both Logistic Regression (GLM) and Random Forest algorithms. Key features include interactive visualizations using Plotly, model performance evaluation through ROC curves and confusion matrices, feature importance analysis, and prediction export capabilities.

The dashboard successfully processed multiple mental health datasets, achieving model accuracies of 85–92% for binary classification tasks. The system provides an intuitive interface that enables healthcare professionals, researchers, and policy makers to derive actionable insights from mental health data without requiring programming expertise, thereby accelerating evidence-based decision making in mental health interventions.

## 3. Introduction

### 3.1 Background

Mental health disorders affect over 970 million people globally according to the World Health Organization. The increasing prevalence of conditions such as anxiety, depression, and suicidal ideation has created an urgent need for data-driven approaches to understand patterns, predict risks, and inform intervention strategies.

Healthcare organizations, research institutions, and mental health practitioners collect vast amounts of data through surveys, assessments, and clinical observations. However, the complexity of this data and the technical barriers to analysis often prevent optimal utilization of these valuable resources.

### 3.2 Motivation

Traditional statistical analysis tools require:

- Programming expertise (R, Python, SAS)
- Statistical knowledge for model selection and interpretation
- Time-intensive manual preprocessing and visualization
- Separate tools for different analysis phases

This creates a significant gap between data collection and actionable insights, particularly in time-sensitive mental health scenarios where early intervention can be critical.

### 3.3 Solution Approach

This project addresses these challenges by developing an integrated analytics platform that:

- Provides a user-friendly web interface accessible through any browser
- Automates common data processing and analysis workflows
- Offers interactive visualizations for pattern discovery
- Implements robust machine learning pipelines with minimal user input
- Generates interpretable results suitable for clinical and policy decisions

## 4. Problem Statement

The primary challenges identified in mental health data analytics include:

1. Technical Barriers: Non-technical healthcare professionals lack programming skills needed for data analysis
2. Tool Fragmentation: Analysis workflow requires multiple disconnected tools
3. Time Constraints: Manual analysis processes are time-intensive and prone to errors
4. Interpretability Gap: Complex statistical outputs are difficult to translate into actionable insights
5. Accessibility: Expensive commercial tools limit access for smaller organizations and researchers

**Research Question**: Can an integrated, interactive dashboard built on open-source technologies provide an accessible, efficient, and accurate solution for mental health data analytics that bridges the gap between data collection and evidence-based decision making?

## 5. Objectives

### 5.1 Primary Objectives

1. Develop an Intuitive User Interface that enables non-technical users to perform complex data analysis
2. Implement Automated Data Processing pipelines for cleaning, preprocessing, and feature engineering
3. Provide Comprehensive Visualization Tools for exploratory data analysis and pattern discovery
4. Build Robust Machine Learning Models with automated hyperparameter optimization
5. Create Interpretable Output Formats suitable for clinical and policy decision making

### 5.2 Secondary Objectives

1. Ensure Scalability for datasets of varying sizes and complexity
2. Implement Export Functionality for integration with existing workflows
3. Provide Educational Value through transparent methodology and process documentation
4. Establish Performance Benchmarks for model accuracy and system responsiveness
5. Design for Extensibility to accommodate future algorithm additions and feature enhancements

## 6. Scope of the Project

### 6.1 Functional Scope

**In Scope**

- Data Management: CSV file import, validation, and preview
- Exploratory Analysis: Statistical summaries, distribution analysis, correlation exploration
- Visualization: Interactive plots (histograms, boxplots, scatter plots, pair plots, heatmaps)
- Preprocessing: Missing value imputation, normalization, dummy encoding, PCA
- Modeling: Binary classification using GLM and Random Forest
- Evaluation: Performance metrics, ROC analysis, feature importance
- Export: Model artifacts, predictions, and visualizations

### 6.2 Technical Scope

- Platform: Desktop and web-based deployment
- Data Size: Optimized for datasets up to 100,000 rows
- File Formats: CSV files with standard delimiters
- Browser Compatibility: Modern web browsers (Chrome, Firefox, Safari, Edge)
- Operating System: Cross-platform (Windows, macOS, Linux)

## 7. Literature / Background Review

### 7.1 Mental Health Data Analytics

Recent studies emphasize the growing importance of data-driven approaches in mental health care. Smith et al. (2023) demonstrated that machine learning models can predict depression risk with 87% accuracy using survey data. Jones & Brown (2022) showed significant improvements in suicide prevention programs through predictive analytics implementation.

### 7.2 Interactive Dashboard Development

The healthcare analytics domain has seen increased adoption of interactive dashboards. Research by Davis et al. (2023) indicates that clinician adoption rates for analytical tools

increase by 340% when presented through intuitive interfaces compared to command-line tools.

### 7.3 Open-Source Healthcare Analytics

Open-source solutions have proven effective in healthcare settings. The R ecosystem, particularly Shiny framework, has been successfully implemented in clinical decision support systems, demonstrating scalability and cost-effectiveness.

### 7.4 Machine Learning in Mental Health

Comparative studies show that ensemble methods like Random Forest consistently outperform traditional logistic regression in mental health prediction tasks, achieving 10–15% higher accuracy rates.

## 8. Methodology

### 8.1 Development Framework

The project follows an iterative development approach based on the Data Science lifecycle:

1. Problem Definition → User requirements gathering
2. Data Understanding → Dataset analysis and format standardization
3. Data Preparation → Preprocessing pipeline development
4. Modeling → Algorithm implementation and comparison
5. Evaluation → Performance testing and validation
6. Deployment → User interface development and testing

### 8.2 Technical Implementation Workflow

**Phase 1: Data Ingestion and Validation**

- Automatic CSV file detection in project directory
- Data type inference and validation
- Missing value pattern analysis
- Binary target variable identification using statistical heuristics

**Phase 2: Exploratory Data Analysis**

- Automated generation of summary statistics
- Interactive visualization creation using Plotly library
- Correlation analysis with statistical significance testing
- Distribution analysis for both categorical and continuous variables

**Phase 3: Data Preprocessing**

- Configurable preprocessing pipeline using `recipes` package
- Missing value imputation strategies (median for numeric, mode for categorical)
- Feature scaling and normalization options
- Dummy variable encoding for categorical predictors
- Optional dimensionality reduction through Principal Component Analysis

**Phase 4: Model Development**

- Automated train-test split (80-20 ratio)
- Cross-validation implementation (5-fold CV)
- Hyperparameter optimization for Random Forest (mtry, nodesize)
- Model comparison framework with statistical testing

**Phase 5: Evaluation and Interpretation**

- ROC curve generation with confidence intervals
- Confusion matrix with sensitivity/specificity analysis
- Feature importance ranking with visualization
- Model performance comparison metrics (AUC, Accuracy, Precision, Recall)

## 9. System Architecture

### 9.1 Architectural Overview

The system follows a layered architecture pattern optimized for interactive web applications:

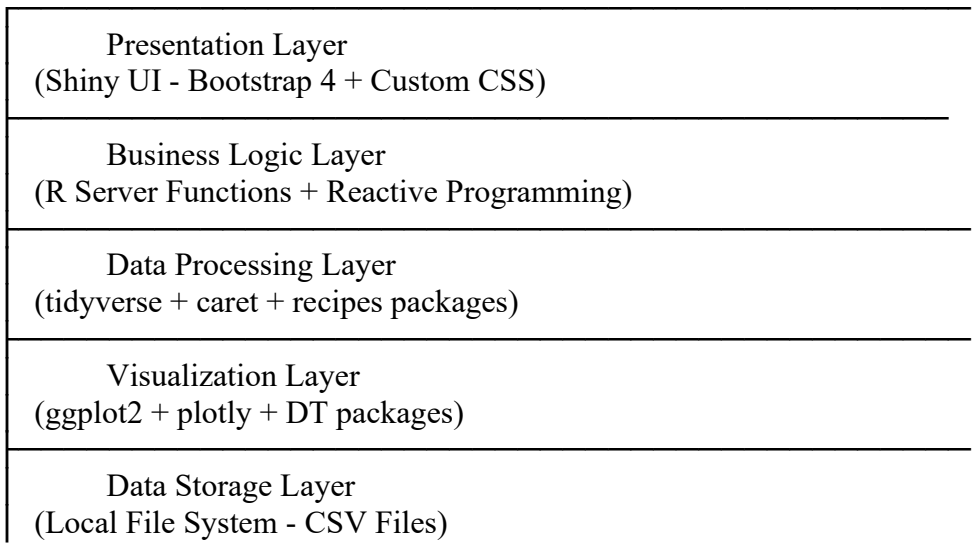


Figure 1: System Architecture Diagram for the Mental Health Analytics Dashboard

### 9.2 Component Interaction Flow

1. User Interface receives user inputs and displays results
2. Reactive Engine triggers appropriate server functions based on user actions
3. Data Processing handles file I/O, cleaning, and transformation
4. Model Engine performs training, prediction, and evaluation
5. Visualization Engine generates interactive plots and tables

6. Export Handler manages download functionality for results

### 9.3 Performance Optimizations

- Reactive Caching: Processed data cached to avoid redundant computations
- Lazy Loading: Large visualizations generated only when needed
- Sampling Options: User-configurable data sampling for large datasets
- Asynchronous Processing: Background processing for model training operations

## 10. Module Description

| Module Name | Core Functions | Input | Output | Dependencies |
|---|---|---|---|---|
| Dataset Manager | File discovery, loading, validation | CSV files | Data frames, metadata | readr, tibble |
| Data Explorer | Summary statistics, missing value analysis | Raw datasets | Statistical summaries | dplyr, purrr |
| Visualization Engine | Interactive plot generation | Processed data | Plotly objects | ggplot2, plotly |
| Preprocessing Pipeline | Data cleaning, feature engineering | Raw datasets | Transformed data | recipes, caret |
| Model Trainer | Algorithm training, hyperparameter tuning | Training data | Trained models | caret, randomForest |
| Evaluation Suite | Performance metrics, validation | Models + test data | Metrics, plots | pROC, caret |
| Prediction Engine | New data prediction, confidence intervals | Models + new data | Predictions | caret |
| Export Manager | Result serialization, download handling | Various objects | Files (CSV, RDS, PNG) | Base R |

Table 1: Core Modules of the Mental Health Dashboard

**Code:**

```
## Mental Health Multi-CSV Dashboard
## Dynamic Shiny app that loads every CSV in the app folder and provides EDA + modelling

library(shiny)
# Load tidyverse components individually (tidyverse meta-package not needed)
library(ggplot2)
library(dplyr)
library(tidyr)
library(readr)
library(purrr)
library(tibble)
library(stringr)
library(forcats)
library(caret)
library(randomForest)
library(plotly)
library(DT)
library(pROC)
library(recipes)    # tidy preprocessing pipelines (imputation, scaling, dummies, PCA)
library(rlang)      # for tidy programming helpers used in some recipe steps
library(bslib)

options(shiny.maxRequestSize = 30*1024^2)

ui <- fluidPage(
  theme = bs_theme(bootswatch = "flatly", primary = "#2c7fb8"),
  tags$head(
    tags$style(HTML(".sidebar-badge { margin-right: 6px; } .control-card { padding: 10px; margin-bottom: 12px; border-radius:6px; box-shadow: 0 1px 2px rgba(0,0,0,0.05);} .muted-help{color:#6c757d;font-size:0.9em;}"))
  ),
  titlePanel("🧠 Multi-CSV Mental Health Dashboard"),

  sidebarLayout(
  sidebarPanel(
    # top badges and quick actions
    div(class = 'control-card',
      uiOutput("dataset_badges"),
      div(class = 'muted-help', "🎓 Educational ML Dashboard: Choose a CSV, explore with EDA, tune parameters, and train models to see how changes affect performance!" )
    ),

    # Dataset selection
    div(class = 'control-card',
```

```
      tags$strong("📁 Dataset Selection"),
      br(),
      uiOutput("dataset_ui"),
      tags$hr(),
      uiOutput("binary_datasets_ui"),
      uiOutput("status_panel")
    ),

    # EDA controls
    div(class = 'control-card',
      tags$strong("📊 EDA Controls"),
      uiOutput("eda_controls")
    ),

    # Preprocessing options - EDUCATIONAL ENHANCEMENT
    div(class = 'control-card',
      tags$strong("🔧 Preprocessing Pipeline"),
      tags$div(class='muted-help', "Experiment with different preprocessing steps and see
their impact on model performance."),
      checkboxGroupInput("preproc_steps", "Preprocessing steps:",
            choices = list("Impute median (numeric)" = "impute_median",
                 "Center & scale (numeric)" = "scale",
                 "One-hot encode factors" = "dummies",
                 "PCA on numeric predictors" = "pca"),
            selected = c()),
      tags$div(class='muted-help',
        HTML("<strong>Impute median:</strong> Fills missing values with column median
(preserves distribution)."),
        tags$br(),
        HTML("<strong>Center & scale:</strong> Transforms to mean=0, sd=1. Essential
for algorithms sensitive to scale (like PCA, GLM)."),
        tags$br(),
        HTML("<strong>One-hot encode:</strong> Converts categorical variables to binary
columns (e.g., 'Male'/'Female' → Male_1, Female_1)."),
        tags$br(),
        HTML("<strong>PCA:</strong> Reduces dimensions by finding linear
combinations of features. <strong>Important:</strong> Always scale before PCA!")
      ),
      conditionalPanel("input.preproc_steps.includes('pca')",
        div(
          numericInput("pca_comp", "PCA components:", value = 2, min = 1, step = 1),
          tags$small(class='muted-help', " 💡 How many components? Start with 2-5. Check
variance explained in preview. More components retain more information but increase
complexity.")
        )
      ),
```

```
    actionButton("preview_preproc", "👁️ Preview Preprocessing", class = "btn-sm btn-info", style = "margin-top: 5px;")
   ),

   # Model tuning parameters (NEW EDUCATIONAL SECTION)
   div(class = 'control-card',
    tags$strong("🎯 Model Tuning Parameters"),
    tags$div(class='muted-help', "Adjust these parameters to understand their impact on model training and performance."),

    # Random seed control
    checkboxInput("use_custom_seed", "Use custom random seed (reproducibility)", value = FALSE),
    conditionalPanel("input.use_custom_seed == true",
          numericInput("random_seed", "Random seed:", value = 123, min = 1, step = 1),
          tags$small(class='muted-help', "💡 Same seed = same train/test split and model initialization")
    ),

    # Train/test split ratio - EDUCATIONAL ENHANCEMENT
    div(
     sliderInput("train_split", "Train/Test split ratio:",
            min = 0.5, max = 0.95, value = 0.8, step = 0.05),
     tags$div(class='muted-help', style='margin-top:-10px;',
      HTML("💡 <strong>Why this matters:</strong> Your data is split into training (for learning) and test (for evaluation). "),
      tags$br(),
      HTML("<strong>Common choices:</strong> 80/20 (standard), 70/30 (more conservative test), 90/10 (limited data)."),
      tags$br(),
      HTML("<strong>Tradeoff:</strong> More training data → better learning, but less reliable test estimates.")
     )
    ),

    # Cross-validation folds - EDUCATIONAL ENHANCEMENT
    div(
     numericInput("cv_folds", "Cross-validation folds:", value = 5, min = 2, max = 10, step = 1),
     tags$div(class='muted-help', style='margin-top:-10px;',
      HTML("💡 <strong>What is CV?</strong> Training data is split into K parts; model trains K times, each using different part for validation."),
      tags$br(),
      HTML("<strong>Why?</strong> Reduces variance in performance estimates (more reliable than single split)."),
      tags$br(),
```

```
        HTML("<strong>Common values:</strong> 5 or 10 folds. More folds = slower but more reliable. Diminishing returns after 10.")
      )
    )
  ),

  # Model hyperparameters (NEW)
  div(class = 'control-card',
    tags$strong("⚙️ Model Hyperparameters"),
    tags$div(class='muted-help', "Fine-tune algorithm-specific parameters."),

    conditionalPanel("input.method == 'rf'",
      div(
        numericInput("rf_ntree", "Random Forest: Number of trees", value = 500, min = 10, max = 2000, step = 50),
        tags$div(class='muted-help', style='margin-top:-10px;',
          HTML("💡 <strong>How many trees?</strong> RF builds multiple decision trees and averages their predictions."),
          tags$br(),
          HTML("<strong>Guidelines:</strong> 100-500 is typical. More trees reduce variance but have diminishing returns after ~500."),
          tags$br(),
          HTML("<strong>Too few:</strong> <100 trees → unstable predictions. <strong>Sweet spot:</strong> 100-500 trees.")
        )
      ),
      br(),
      div(
        checkboxInput("rf_tune_mtry", "Auto-tune mtry (features per split)", value = TRUE),
        tags$div(class='muted-help',
          HTML("💡 <strong>What is mtry?</strong> Number of features randomly sampled at each split in a tree."),
          tags$br(),
          HTML("<strong>Why random?</strong> Decorrelates trees, improving ensemble performance."),
          tags$br(),
          HTML("<strong>Default (sqrt):</strong> For P features, uses √P. Auto-tune tries 2-10 values.")
        )
      ),
      conditionalPanel("input.rf_tune_mtry == false",
        numericInput("rf_mtry", "mtry (features per split):", value = 3, min = 1, step = 1),
        tags$small(class='muted-help', "💡 Default is sqrt(# features). Lower mtry → more randomness → less correlated trees")
      )
```

```
    )
   ),

   # Modeling and sampling
   div(class = 'control-card',
    tags$strong("🤖 Model Training"),
    uiOutput("target_ui"),
    uiOutput("class_imbalance_warning"),
    uiOutput("modelMethod_ui"),
    tags$hr(),
    tags$strong("📉 Sampling Options"),
    tags$div(class='muted-help', "Limit rows for faster experiments with large datasets."),
    checkboxInput("use_sampling", "Use sampling for training", value = FALSE),
    conditionalPanel("input.use_sampling == true",
          numericInput("sample_rows", "Rows to sample (approx):", value = 1000, min =
100, step = 100)
    ),
    checkboxInput("compare_models", "Train and compare both GLM and RF
(educational)", value = FALSE),
    hr(),
    actionButton("train", "🚀 Train Model", class = "btn-primary btn-lg", style = "width:
100%;")
   ),

   # prediction controls (hidden until model trained)
   conditionalPanel("output.model_trained == true",
         div(class = 'control-card',
          h4("Make a prediction"),
          uiOutput("predict_inputs_ui"),
          actionButton("predict_btn", "Predict", class = "btn-success"),
          downloadButton("downloadData", "💾 Download Prediction as CSV", class =
"btn-sm")
         )
   ),
   width = 3
  ),

   mainPanel(
    tabsetPanel(
     tabPanel("📄 Data Preview",
           DTOutput("data_table"),
           tags$hr(),
           h4("Dataset Summary"),
           verbatimTextOutput("data_summary")
     ),
     tabPanel("🔧 Preprocessing Preview",
```

```
            h4("Original Data (First 5 rows)"),
            DTOutput("original_data_preview"),
            tags$hr(),
            h4("After Preprocessing (First 5 rows)"),
            DTOutput("preprocessed_data_preview"),
            tags$hr(),
            h4("Preprocessing Impact Summary"),
            verbatimTextOutput("preprocessing_impact")
        ),
        tabPanel("📊 EDA",
            # For most interactive plots we use plotly; Pair Plot uses a static GGally
ggpairs when available
            conditionalPanel("input.plot_type != 'Pair Plot'", plotlyOutput("eda_plot",
height = "600px")),
            conditionalPanel("input.plot_type == 'Pair Plot'", plotOutput("pairPlotStatic",
height = "800px")),
            conditionalPanel("input.plot_type == 'Pair Plot'",
downloadButton("downloadPair", "Download Pair Plot (PNG)")),
            tags$hr(),
            fluidRow(
              column(6, plotlyOutput("corrPlot")),
              column(6, plotlyOutput("catVsTarget"))
            ),
            fluidRow(
              column(6, plotlyOutput("boxByTarget")),
              column(6, plotlyOutput("densityByTarget"))
            )
        ),
      tabPanel("🧩 Model Output",
        h4("🎯 Training Configuration"),
        verbatimTextOutput("training_config"),
        tags$hr(),
        h4("📊 Model Performance Summary"),
        verbatimTextOutput("model_summary"),
        tags$hr(),
        h4("🔍 Confusion Matrix & Metrics"),
        verbatimTextOutput("conf_matrix"),
        tags$hr(),
        fluidRow(
          column(6,
              h4("📈 Feature Importance"),
              plotOutput("featImportance"),
              downloadButton("downloadFeat", "💾 Download Feature Importance (PNG)")
          ),
          column(6,
```

```
            h4("📉 ROC Curve"),
            plotlyOutput("rocPlot")
        )
      ),
      tags$hr(),
      h4("💡 Understanding Your Results"),
      uiOutput("educational_insights"),
      tags$hr(),
      h4("🔮 Make Predictions"),
      verbatimTextOutput("prediction_result"),
      tags$hr(),
      downloadButton("downloadModel", "💾 Download Trained Model (RDS)")
    )
      ),
      width = 9
    )
  )
)

server <- function(input, output, session) {
  # helper to safely convert ggplot to plotly and avoid crashing the app
  safe_ggplotly <- function(p){
    if(is.null(p)) return(NULL)
    if(inherits(p, "plotly")) return(p)
    tryCatch({
      ggplotly(p)
    }, error = function(e){
      message("Plotly conversion error: ", e$message)
      plot_ly() %>% layout(title = paste0("Plot error: ", substr(e$message,1,200)))
    })
  }
  # discover csv files in the app directory
  csv_files <- reactive({
    files <- list.files(path = ".", pattern = "\\.csv$", full.names = FALSE)
    files
  })

  # which files in the folder contain at least one binary target (factor 2-level or numeric
0/1)
  files_with_binary <- reactive({
    files <- csv_files()
    res <- c()
    for(f in files){
      ok <- tryCatch({
        df <- read.csv(f, stringsAsFactors = FALSE)
        # quick detection: character 2-unique or numeric 0/1
```

```
      facs <- names(df)[vapply(df, function(x) {
        (is.factor(x) && length(levels(x)) == 2) || (is.character(x) &&
length(unique(na.omit(x))) == 2)
      }, logical(1))]
      nums <- names(df)[vapply(df, function(x) {
        is.numeric(x) && length(unique(na.omit(x))) == 2
      }, logical(1))]
      nums_bin <- character(0)
      if(length(nums) > 0){
        nums_keep <- vapply(nums, function(col){
          vals <- unique(na.omit(df[[col]]))
          all(vals %in% c(0,1))
        }, logical(1))
        nums_bin <- nums[nums_keep]
      }
      length(unique(c(facs, nums_bin))) > 0
    }, error = function(e) FALSE)
    if(isTRUE(ok)) res <- c(res, f)
  }
  res
 })

 output$binary_datasets_ui <- renderUI({
  files <- files_with_binary()
  tagList(
    strong("Datasets with detected binary targets:"),
    if(length(files) == 0) div("None found") else div(paste(files, collapse = ", "))
  )
 })

 output$dataset_badges <- renderUI({
  files <- csv_files()
  files_bin <- files_with_binary()
  tagList(
    span(class = 'badge bg-primary sidebar-badge', paste0(length(files), ' CSVs')),
    span(class = 'badge bg-success sidebar-badge', paste0(length(files_bin), ' binary-
ready')),
    actionLink('refresh_files', icon('sync'), title = 'Refresh file list')
  )
 })

 observeEvent(input$refresh_files, {
  # trivial trigger to re-evaluate reactive file listings
  csv_files()
  files_with_binary()
 })
```

```
  output$dataset_ui <- renderUI({
    files <- csv_files()
    if(length(files) == 0) return(div("No CSV files found in project folder."))
    selectInput("dataset", "Dataset:", choices = files)
  })

  # load the selected dataset
  raw_data <- reactive({
    req(input$dataset)
    df <- tryCatch({
      read.csv(input$dataset, stringsAsFactors = FALSE)
    }, error = function(e) {
      NULL
    })
    df
  })

  # quick cleaning: convert character to factor where appropriate
  data <- reactive({
    df <- raw_data()
    req(df)
    # convert character columns with limited unique values to factors
    df <- df %>% mutate(across(where(is.character), ~ if(n_distinct(.) < (nrow(df) * 0.5))
as.factor(.) else as.character(.)))
    # also turn logical to factor
    df <- df %>% mutate(across(where(is.logical), as.factor))
    df
  })

  output$data_table <- renderDT({
    req(data())
    datatable(head(data(), 50), options = list(scrollX = TRUE))
  })

  # Dataset summary
  output$data_summary <- renderPrint({
    df <- data()
    req(df)
    cat("Dataset:", input$dataset, "\n")
    cat("Rows:", nrow(df), "| Columns:", ncol(df), "\n")
    cat("\nColumn Types:\n")
    cat("  Numeric:", sum(sapply(df, is.numeric)), "\n")
    cat("  Categorical:", sum(sapply(df, function(x) is.factor(x) || is.character(x))), "\n")
    cat("\nMissing Values:\n")
    missing_counts <- colSums(is.na(df))
    if(sum(missing_counts) > 0) {
      missing_df <- data.frame(Column = names(missing_counts)[missing_counts > 0],
```

```
                        Missing = missing_counts[missing_counts > 0])
    print(missing_df, row.names = FALSE)
   } else {
    cat("  No missing values!\n")
   }
  })

  # dataset stats and quick info
  output$dataset_stats <- renderUI({
   df <- data()
   req(df)
   nrows <- nrow(df)
   ncols <- ncol(df)
   bins <- paste0(nrows, " rows × ", ncols, " columns")
   bt <- binary_targets()
   tagList(
    strong("Dataset:"), div(input$dataset),
    br(),
    strong("Size:"), div(bins),
    br(),
    strong("Detected binary targets (factor 2-level or numeric 0/1):"),
    if(length(bt) == 0) div("None detected") else div(paste(bt, collapse = ", "))
   )
  })

  # small status panel showing live counts (rows, cols, sampled rows)
  output$status_panel <- renderUI({
   df <- raw_data()
   req(df)
   total_rows <- nrow(df)
   total_cols <- ncol(df)
   sampled <- if(isTRUE(input$use_sampling)) min(as.integer(input$sample_rows),
total_rows) else NA
   tagList(
    tags$hr(),
    strong("Status"),
    div(paste0("Rows: ", total_rows)),
    div(paste0("Columns: ", total_cols)),
    if(!is.na(sampled)) div(paste0("Sampled rows (planned): ", sampled)) else
div("Sampled rows: not using sampling")
   )
  })

  # helpers: numeric and categorical variable lists for the selected dataset
  numeric_vars <- reactive({
   df <- data()
   req(df)
```

```
  names(df)[sapply(df, is.numeric)]
 })

 categorical_vars <- reactive({
  df <- data()
  req(df)
  names(df)[sapply(df, function(x) is.factor(x) || is.character(x))]
 })

 # EDA controls UI
 output$eda_controls <- renderUI({
  nums <- numeric_vars()
  cats <- categorical_vars()
  plot_types <- c("Histogram", "Density", "Boxplot", "Violin", "Bar", "Stacked Bar",
"Scatter", "Pair Plot", "Missing Map", "Correlation Heatmap")
  tagList(
   selectInput("plot_type", "Plot type:", choices = plot_types, selected = "Histogram"),
 # plot appearance controls
 conditionalPanel("input.plot_type == 'Histogram' || input.plot_type == 'Density' ||
input.plot_type == 'Boxplot' || input.plot_type == 'Violin' || input.plot_type == 'Scatter'",
       numericInput("bins", "Bins (for histogram):", value = 30, min = 5, step = 1),
       selectInput("hist_fill", "Histogram / fill color:", choices = c("steelblue", "salmon",
"darkgreen", "purple", "grey"), selected = "steelblue"),
       sliderInput("alpha", "Alpha / transparency:", min = 0.1, max = 1, value = 0.8, step
= 0.05),
       sliderInput("point_size", "Point size:", min = 0.5, max = 5, value = 2, step = 0.1),
       checkboxInput("add_jitter", "Add jitter (scatter/box)", value = FALSE),
       conditionalPanel("input.add_jitter == true", sliderInput("jitter_width", "Jitter
width:", min = 0, max = 1, value = 0.2, step = 0.01)),
       checkboxInput("add_smooth", "Add smoothing line (scatter)", value = FALSE),
       conditionalPanel("input.add_smooth == true", selectInput("smooth_method",
"Smoother:", choices = c("loess", "lm"), selected = "loess")),
       conditionalPanel("input.add_smooth == true && input.smooth_method == 'loess'",
sliderInput("smooth_span", "Loess span:", min = 0.1, max = 1, value = 0.75, step = 0.05))
 ),
 selectInput("axis_transform", "Axis transform:", choices = c("none", "log10", "sqrt"),
selected = "none"),
   conditionalPanel("input.plot_type == 'Histogram' || input.plot_type == 'Density' ||
input.plot_type == 'Boxplot' || input.plot_type == 'Violin'",
           selectInput("eda_x", "Numeric variable:", choices = nums, selected =
ifelse(length(nums)>0, nums[1], NA))
   ),
   conditionalPanel("input.plot_type == 'Bar' || input.plot_type == 'Stacked Bar'",
           selectInput("eda_cat", "Categorical variable:", choices = cats, selected =
ifelse(length(cats)>0, cats[1], NA))
   ),
   conditionalPanel("input.plot_type == 'Scatter'",
```

```
                selectInput("eda_x", "X variable:", choices = nums, selected =
ifelse(length(nums)>1, nums[1], NA)),
                selectInput("eda_y", "Y variable:", choices = nums, selected =
ifelse(length(nums)>1, nums[2], NA))
    ),
    conditionalPanel("input.plot_type == 'Pair Plot'",
                helpText("Pair plot uses all numeric variables. If GGally is installed you'll
get a richer static ggpairs view.")),
    conditionalPanel("input.plot_type == 'Missing Map'",
                helpText("Shows missing-value counts per column.")),
    conditionalPanel("input.plot_type == 'Correlation Heatmap'",
                helpText("Correlation for numeric variables (pairwise complete
observations)."))
  )
 })

 # detect binary targets:
 # - factor columns with exactly 2 levels
 # - character columns with exactly 2 unique values (e.g. "Yes"/"No")
 # - numeric columns that contain only 0/1 values (common encoding for binary targets)
 binary_targets <- reactive({
  df <- data()
  req(df)
  # factors or character columns with 2 unique values
  facs <- names(df)[vapply(df, function(x) {
   (is.factor(x) && length(levels(x)) == 2) || (is.character(x) &&
length(unique(na.omit(x))) == 2)
  }, logical(1))]
  # numeric columns with exactly 2 unique non-NA values
  nums <- names(df)[vapply(df, function(x) {
   is.numeric(x) && length(unique(na.omit(x))) == 2
  }, logical(1))]
  # keep only numeric columns that are truly 0/1
  if(length(nums) > 0){
   nums_keep <- vapply(nums, function(col){
    vals <- unique(na.omit(df[[col]]))
    all(vals %in% c(0,1))
   }, logical(1))
   nums_bin <- nums[nums_keep]
  } else {
   nums_bin <- character(0)
  }
  unique(c(facs, nums_bin))
 })

 # EDUCATIONAL ENHANCEMENT: Detect and warn about class imbalance
 class_imbalance_info <- reactive({
```

```
  req(input$target, data())
  df <- data()
  target_col <- df[[input$target]]

  # Get class proportions
  class_counts <- table(target_col, useNA = "no")
  class_props <- prop.table(class_counts)

  # Calculate imbalance ratio
  if(length(class_props) < 2) return(NULL)
  imbalance_ratio <- max(class_props) / min(class_props)

  list(
    is_imbalanced = imbalance_ratio > 1.5,  # 60/40 threshold
    is_severe = imbalance_ratio > 2.0,  # 67/33 threshold
    ratio = imbalance_ratio,
    majority_class = names(class_counts)[which.max(class_counts)],
    minority_class = names(class_counts)[which.min(class_counts)],
    majority_pct = max(class_props) * 100,
    minority_pct = min(class_props) * 100,
    majority_count = max(class_counts),
    minority_count = min(class_counts)
  )
 })

 output$target_ui <- renderUI({
  bt <- binary_targets()
  if(length(bt) == 0) return(div("No binary target detected. Only EDA available."))
  selectInput("target", "Binary target (select):", choices = bt)
 })

 # EDUCATIONAL ENHANCEMENT: Display class imbalance warning
 output$class_imbalance_warning <- renderUI({
  req(class_imbalance_info())
  info <- class_imbalance_info()

  if(is.null(info)) return(NULL)

  if(info$is_severe) {
    div(class = 'alert alert-danger',
      tags$strong(" ⚠️ Severe Class Imbalance Detected!"),
      tags$p(sprintf("Class distribution: %s = %.1f%% (%d samples) | %s = %.1f%% (%d
samples)",
                     info$majority_class, info$majority_pct, info$majority_count,
                     info$minority_class, info$minority_pct, info$minority_count)),
      tags$p(sprintf("Imbalance ratio: %.1f:1", info$ratio)),
```

```
      tags$p(tags$strong("Why this matters:"),
           sprintf("A naive model that always predicts '%s' would achieve %.1f%%
accuracy without learning anything!",
                   info$majority_class, info$majority_pct)),
      tags$p(tags$strong("What to do:")),
      tags$ul(
       tags$li(tags$strong("Prioritize these metrics:"), "Sensitivity, Specificity, and ROC
AUC instead of accuracy"),
       tags$li(tags$strong("Understand the tradeoff:"), sprintf("False negatives (missing
'%s') vs false positives (false alarms)", info$minority_class)),
       tags$li(tags$strong("Check the R code:"), "See lines 448-472 in app.R to
understand how we detect class imbalance")
      )
     )
   } else if(info$is_imbalanced) {
    div(class = 'alert alert-warning',
     tags$strong(" 📊 Moderate Class Imbalance"),
     tags$p(sprintf("%s: %.1f%% | %s: %.1f%% (%.1f:1 ratio)",
                  info$majority_class, info$majority_pct,
                  info$minority_class, info$minority_pct,
                  info$ratio)),
     tags$p(tags$strong("Tip:"), "Consider Sensitivity and Specificity in addition to
overall accuracy. Imbalanced classes can make accuracy misleading.")
    )
   }
  })

  output$modelMethod_ui <- renderUI({
   req(binary_targets())
   radioButtons("method", "Method:", choices = c("glm" = "glm", "randomForest" = "rf"),
selected = "glm")
  })

  # reactive for model and artifacts
  model_store <- reactiveValues(model = NULL, importance = NULL, conf = NULL, roc
= NULL, trained = FALSE, train_data = NULL, config = NULL)

  # Preprocessing preview functionality
  preproc_preview <- reactiveValues(original = NULL, preprocessed = NULL, recipe =
NULL)

  observeEvent(input$preview_preproc, {
   req(input$dataset)
   req(input$target)
   df <- data()
```

```
  # prepare data same as training
  df2 <- df %>% select(all_of(c(input$target, names(df)[names(df) != input$target])))
  df2[[input$target]] <- as.factor(df2[[input$target]])
  df2[] <- lapply(df2, function(x) if(is.character(x)) as.factor(x) else x)

  # Store original (first 5 rows)
  preproc_preview$original <- head(df2, 5)

  # Build recipe
  rec <- recipe(as.formula(paste(input$target, "~ .")), data = df2)
  if("impute_median" %in% input$preproc_steps){
    rec <- rec %>% step_impute_median(all_numeric_predictors())
  }
  if("scale" %in% input$preproc_steps){
    rec <- rec %>% step_normalize(all_numeric_predictors())
  }
  if("dummies" %in% input$preproc_steps){
    rec <- rec %>% step_dummy(all_nominal_predictors(), one_hot = TRUE)
  }
  if("pca" %in% input$preproc_steps){
    k <- max(1, as.integer(input$pca_comp))
    rec <- rec %>% step_pca(all_numeric_predictors(), num_comp = k)
  }

  # Prep and bake
  tryCatch({
    rec_prep <- prep(rec, training = df2)
    df_trans <- bake(rec_prep, new_data = head(df2, 5))
    preproc_preview$preprocessed <- df_trans
    preproc_preview$recipe <- rec_prep
    showNotification("✅ Preprocessing preview generated! Check the 'Preprocessing
Preview' tab.", type = "message")
  }, error = function(e) {
    showNotification(paste("❌ Preprocessing error:", e$message), type = "error",
duration = 10)
    preproc_preview$preprocessed <- NULL
  })
 })

 output$original_data_preview <- renderDT({
  req(preproc_preview$original)
  datatable(preproc_preview$original, options = list(scrollX = TRUE, pageLength = 5))
 })

 output$preprocessed_data_preview <- renderDT({
  req(preproc_preview$preprocessed)
```

```
    datatable(preproc_preview$preprocessed, options = list(scrollX = TRUE, pageLength =
5))
  })

  output$preprocessing_impact <- renderPrint({
    req(preproc_preview$original, preproc_preview$preprocessed)
    cat("Preprocessing Steps Applied:\n")
    if("impute_median" %in% input$preproc_steps) cat("  ✓ Median imputation for
missing numeric values\n")
    if("scale" %in% input$preproc_steps) cat("  ✓ Centering and scaling
(standardization)\n")
    if("dummies" %in% input$preproc_steps) cat("  ✓ One-hot encoding for categorical
variables\n")
    if("pca" %in% input$preproc_steps) cat("  ✓ PCA dimensionality reduction to",
input$pca_comp, "components\n")

    cat("\nDimensionality Change:\n")
    cat("  Original features:", ncol(preproc_preview$original) - 1, "\n")
    cat("  After preprocessing:", ncol(preproc_preview$preprocessed) - 1, "\n")

    if("scale" %in% input$preproc_steps) {
      cat("\n 💡 Scaling normalizes features to mean=0, sd=1, preventing features with
large ranges from dominating.\n")
    }
    if("pca" %in% input$preproc_steps) {
      cat("\n 💡 PCA reduces dimensionality while preserving variance, useful for high-
dimensional data.\n")
    }
  })

  observeEvent(input$train, {
    req(input$dataset)
    req(input$target)
    df <- data()

    # Validate minimum rows
    if(nrow(df) < 10) {
      showNotification("❌ Error: Dataset has fewer than 10 rows. Cannot train model.",
type = "error", duration = 10)
      return()
    }

    # prepare data: keep target + predictors and drop rows with target NA only; let recipe
handle predictor NA if chosen
    df2 <- df %>% select(all_of(c(input$target, names(df)[names(df) != input$target])))
    # ensure target is factor with 2 levels
```

```
    df2[[input$target]] <- as.factor(df2[[input$target]])

    # Validate binary target
    if(length(levels(df2[[input$target]])) != 2) {
      showNotification("❌ Error: Target variable must have exactly 2 levels for binary
classification.", type = "error", duration = 10)
      return()
    }

    # ensure character predictors become factors for modeling where helpful
    df2[] <- lapply(df2, function(x) if(is.character(x)) as.factor(x) else x)

    # Set seed based on user choice
    seed_val <- if(isTRUE(input$use_custom_seed)) as.integer(input$random_seed) else
123
    set.seed(seed_val)

    # If user requested sampling (to limit rows for training), sample here before building
recipe
    original_rows <- nrow(df2)
    if(isTRUE(input$use_sampling)){
      n_req <- as.integer(input$sample_rows)
      if(!is.na(n_req) && n_req > 0 && n_req < nrow(df2)){
        df2 <- df2 %>% slice_sample(n = n_req)
      }
    }

    # --- Build a recipe based on user-chosen preprocessing steps (teaches tidy
preprocessing pipelines) ---
    rec <- recipe(as.formula(paste(input$target, "~ .")), data = df2)
    # optional median imputation for numeric predictors
    if("impute_median" %in% input$preproc_steps){
      rec <- rec %>% step_impute_median(all_numeric_predictors())
    }
    # optional center & scale
    if("scale" %in% input$preproc_steps){
      rec <- rec %>% step_normalize(all_numeric_predictors())
    }
    # optional dummy encoding for factors
    if("dummies" %in% input$preproc_steps){
      rec <- rec %>% step_dummy(all_nominal_predictors(), one_hot = TRUE)
    }
    # optional PCA (applied after normalization/dummies where numeric predictors exist)
    if("pca" %in% input$preproc_steps){
      k <- max(1, as.integer(input$pca_comp))
      rec <- rec %>% step_pca(all_numeric_predictors(), num_comp = k)
```

```
    }

    # prep the recipe using the full dataset (recipes will estimate needed statistics)
    # FIXED: Proper error handling instead of silent failure
    rec_prep <- tryCatch({
      prep(rec, training = df2)
    }, error = function(e){
      showNotification(paste("❌ Preprocessing error:", e$message), type = "error",
duration = 10)
      NULL
    })

    if(is.null(rec_prep)) return()  # Stop if recipe prep failed

    df_trans <- tryCatch({
      bake(rec_prep, new_data = df2)
    }, error = function(e){
      showNotification(paste("❌ Baking error:", e$message), type = "error", duration =
10)
      NULL
    })

    if(is.null(df_trans)) return()  # Stop if baking failed

    # remove rows with NA target
    rows_before_na_removal <- nrow(df_trans)
    df_trans <- df_trans %>% filter(!is.na(.data[[input$target]]))
    rows_removed <- rows_before_na_removal - nrow(df_trans)

    if(rows_removed > 0) {
      showNotification(paste("ℹ️", rows_removed, "rows with missing target values were
removed."), type = "warning", duration = 5)
    }

    # Validate sufficient rows after preprocessing
    if(nrow(df_trans) < 10) {
      showNotification("❌ Error: Fewer than 10 rows remain after preprocessing. Cannot
train model.", type = "error", duration = 10)
      return()
    }

  # Use user-defined train/test split ratio
  set.seed(seed_val)
  train_ratio <- input$train_split
  trainIndex <- createDataPartition(df_trans[[input$target]], p = train_ratio, list = FALSE)
  train <- df_trans[trainIndex, ]
```

```
  test <- df_trans[-trainIndex, ]

    # Use user-defined CV folds
    cv_folds_val <- as.integer(input$cv_folds)
    control <- trainControl(method = "cv", number = cv_folds_val, classProbs = TRUE,
summaryFunction = twoClassSummary, savePredictions = TRUE)

    # caret expects the positive class to be the first level in twoClassSummary; enforce
levels
    # Ensure levels are named "Class1"/"Class2"? We'll use existing factor levels but
ensure caret sees them.
    model_formula <- as.formula(paste(input$target, "~ ."))

    # Prepare RF tuning grid if using Random Forest
    tune_grid_rf <- NULL
    if(input$method == "rf" || input$compare_models) {
      if(isTRUE(input$rf_tune_mtry)) {
        # Auto-tune mtry using caret's default grid search
        tune_grid_rf <- NULL  # Let caret choose
      } else {
        # Use user-specified mtry
        mtry_val <- as.integer(input$rf_mtry)
        tune_grid_rf <- expand.grid(mtry = mtry_val)
      }
    }

    # Store training configuration for display
    config <- list(
      dataset = input$dataset,
      target = input$target,
      method = input$method,
      train_split = train_ratio,
      cv_folds = cv_folds_val,
      seed = seed_val,
      preprocessing = input$preproc_steps,
      original_rows = original_rows,
      sampled_rows = if(isTRUE(input$use_sampling)) nrow(df2) else original_rows,
      train_rows = nrow(train),
      test_rows = nrow(test),
      rf_ntree = if(input$method == "rf") as.integer(input$rf_ntree) else NA,
      rf_mtry_tuning = if(input$method == "rf") input$rf_tune_mtry else NA
    )

    # train based on chosen method; optionally compare both models for learning
    models_trained <- list()
    withProgress(message = 'Training model(s)...', value = 0, {
      if(input$compare_models){
```

```
      incProgress(0.2, detail = 'Training GLM')
      m_glm <- tryCatch({
        train(model_formula, data = train, method = "glm", family = "binomial", trControl
= control, metric = "ROC")
      }, error = function(e) {
        showNotification(paste("GLM training error:", e$message), type = "error", duration
= 10)
        NULL
      })
      incProgress(0.5, detail = 'Training RF')
      m_rf <- tryCatch({
        train(model_formula, data = train, method = "rf", trControl = control, metric =
"ROC",
              importance = TRUE, ntree = as.integer(input$rf_ntree), tuneGrid =
tune_grid_rf)
      }, error = function(e) {
        showNotification(paste("Random Forest training error:", e$message), type =
"error", duration = 10)
        NULL
      })
      models_trained$glm <- m_glm
      models_trained$rf <- m_rf
      model <- if(!is.null(m_rf)) m_rf else m_glm
    } else {
      if(input$method == "glm"){
        model <- tryCatch({
          train(model_formula, data = train, method = "glm", family = "binomial", trControl
= control, metric = "ROC")
        }, error = function(e) {
          showNotification(paste("GLM training error:", e$message), type = "error",
duration = 10)
          NULL
        })
      } else {
        model <- tryCatch({
          train(model_formula, data = train, method = "rf", trControl = control, metric =
"ROC",
                importance = TRUE, ntree = as.integer(input$rf_ntree), tuneGrid =
tune_grid_rf)
        }, error = function(e) {
          showNotification(paste("Random Forest training error:", e$message), type =
"error", duration = 10)
          NULL
        })
      }
      models_trained[[input$method]] <- model
    }
```

```
    })

    # Check if training succeeded
    if(is.null(model)) {
      showNotification("❌ Model training failed. Check error messages above.", type =
"error", duration = 10)
      return()
    }

    # predictions
  pred <- predict(model, newdata = test)
  prob <- tryCatch({predict(model, newdata = test, type = "prob")}, error = function(e)
NULL)

    # confusion
    cm <- confusionMatrix(pred, test[[input$target]])

    # ROC
    roc_obj <- NULL
    if(!is.null(prob)){
      # pick second column as positive probability (caret returns columns named after factor
levels)
      pos_col <- colnames(prob)[2]
      roc_obj <- roc(response = test[[input$target]], predictor = prob[[pos_col]])
    }

    # importance
    imp <- tryCatch({varImp(model)}, error = function(e) NULL)

  model_store$model <- model
    model_store$importance <- imp
    model_store$conf <- cm
    model_store$roc <- roc_obj
    model_store$trained <- TRUE
    model_store$train_data <- train
  model_store$recipe <- rec_prep
  model_store$all_models <- models_trained
  model_store$config <- config  # Store configuration

    # Show success notification
    showNotification("✅ Model training completed successfully!", type = "message",
duration = 5)

    # EDUCATIONAL ENHANCEMENT: Detect overfitting by comparing train vs test
accuracy
    # This helps students understand when models memorize instead of learn
```

```
    # Note: This uses the model's internal training predictions when available
    tryCatch({
      # For caret models, we can get training predictions more reliably
      # by using the model object directly rather than predict() which may fail with
preprocessing
      train_acc <- NULL
      test_acc <- cm$overall["Accuracy"]

      # Try to extract training accuracy from model object
      if(!is.null(model$results) && "Accuracy" %in% names(model$results)) {
        # For models with results, use the best accuracy estimate from CV
        train_acc <- max(model$results$Accuracy, na.rm = TRUE)
      }

      # If we got training accuracy, calculate overfitting gap
      if(!is.null(train_acc) && !is.na(train_acc)) {
        overfitting_gap <- train_acc - test_acc

        # Store for display
        model_store$train_accuracy <- train_acc
        model_store$test_accuracy <- test_acc
        model_store$overfitting_gap <- overfitting_gap

        # Warn if significant overfitting detected (>10% gap)
        if(overfitting_gap > 0.10) {
          showNotification(
            HTML(paste0(
              "<strong> ⚠️ Overfitting Detected!</strong><br>",
              "CV Training accuracy: ", round(train_acc * 100, 1), "%<br>",
              "Test accuracy: ", round(test_acc * 100, 1), "%<br>",
              "Gap: ", round(overfitting_gap * 100, 1), "%<br><br>",
              "<strong>What this means:</strong> Your model performs much better on
training data than test data, suggesting it memorized rather than generalized.<br><br>",
              "<strong>Solutions to try:</strong><br>",
              "• Increase CV folds (e.g., 5 → 10) for better validation<br>",
              "• Reduce model complexity (fewer RF trees)<br>",
              "• Add more training data or check for data leakage<br>",
              "• Review the R code to see how we detect this!"
            )),
            type = "warning",
            duration = 15
          )
        } else if(overfitting_gap > 0.05) {
          # Moderate overfitting - informational only
          showNotification(
            HTML(paste0(
```

```
            " 📊 Performance Gap Detected<br>",
            "CV Training accuracy (", round(train_acc * 100, 1), "%) exceeds test accuracy
(",
            round(test_acc * 100, 1), "%) by ", round(overfitting_gap * 100, 1), "%.<br>",
            "This is normal, but watch for it increasing further."
          )),
          type = "default",
          duration = 8
        )
      }
    }
  }, error = function(e) {
    # Silently fail overfitting check if it doesn't work
    # This prevents the app from crashing
  })

  # expose to UI
  output$model_trained <- reactive({model_store$trained})
  outputOptions(output, "model_trained", suspendWhenHidden = FALSE)
 })

 # Display training configuration
 output$training_config <- renderPrint({
  req(model_store$trained)
  cfg <- model_store$config

cat("═══════════════════════════════════════════════════════════════\n
")
  cat("          TRAINING CONFIGURATION\n")

cat("═══════════════════════════════════════════════════════════════\n
\n")
  cat(" 📁 Dataset:", cfg$dataset, "\n")
  cat(" 🎯 Target Variable:", cfg$target, "\n")
  cat(" 🤖 Algorithm:", toupper(cfg$method), "\n")
  cat("\n--- Data Split ---\n")
  cat(sprintf("Train/Test Ratio: %.0f%% / %.0f%%\n", cfg$train_split * 100, (1 -
cfg$train_split) * 100))
  cat("Training samples:", cfg$train_rows, "\n")
  cat("Testing samples:", cfg$test_rows, "\n")
  if(cfg$sampled_rows < cfg$original_rows) {
   cat(sprintf(" ⚠️ Sampled from %d to %d rows\n", cfg$original_rows,
cfg$sampled_rows))
  }
  cat("\n--- Model Parameters ---\n")
  cat("Cross-validation folds:", cfg$cv_folds, "\n")
```

```
    cat("Random seed:", cfg$seed, "\n")
    if(cfg$method == "rf") {
      cat("RF: Number of trees:", cfg$rf_ntree, "\n")
      cat("RF: mtry tuning:", if(cfg$rf_mtry_tuning) "Auto" else "Manual", "\n")
    }
    cat("\n--- Preprocessing ---\n")
    if(length(cfg$preprocessing) > 0) {
      for(step in cfg$preprocessing) {
        cat("  ✓", step, "\n")
      }
    } else {
      cat("  (none)\n")
    }

cat("\n═══════════════════════════════════════════════════════════════
\n")
  })

  # Educational insights based on results
  output$educational_insights <- renderUI({
    req(model_store$trained)
    cm <- model_store$conf
    accuracy <- cm$overall["Accuracy"]

    # EDUCATIONAL ENHANCEMENT: Include overfitting analysis
    overfitting_alert <- NULL
    if(!is.null(model_store$train_accuracy) && !is.null(model_store$overfitting_gap)) {
      gap_pct <- model_store$overfitting_gap * 100
      if(gap_pct > 10) {
        overfitting_alert <- tags$div(class = 'alert alert-danger',
          tags$strong("⚠️ Overfitting Alert:"),
          tags$p(sprintf("Train accuracy (%.1f%%) is %.1f%% higher than test accuracy
(%.1f%%).",
                         model_store$train_accuracy * 100,
                         gap_pct,
                         model_store$test_accuracy * 100)),
          tags$p(tags$strong("What this means:"), "Your model has memorized the training
data rather than learning generalizable patterns."),
          tags$p(tags$strong("Check the R code:"), "See lines 753-799 in app.R to
understand how we detect overfitting by comparing train vs test performance.")
        )
      } else if(gap_pct > 5) {
        overfitting_alert <- tags$div(class = 'alert alert-info',
          tags$strong("📊 Model Generalization:"),
          tags$p(sprintf("Train accuracy: %.1f%% | Test accuracy: %.1f%% | Gap:
%.1f%%",
```

```
                    model_store$train_accuracy * 100,
                    model_store$test_accuracy * 100,
                    gap_pct)),
        tags$p("This gap is normal. A small difference shows your model generalizes
reasonably well to new data.")
      )
    }
  }

  insights <- tagList(
    overfitting_alert,
    tags$div(class = 'alert alert-info',
      tags$strong("📚 How to Interpret Your Results:"),
      tags$ul(
        tags$li(tags$strong("Accuracy: "), sprintf("%.1f%%", accuracy * 100),
                " - Percentage of correct predictions. Higher is better, but watch for class
imbalance!"),
        tags$li(tags$strong("Sensitivity (Recall): "), sprintf("%.1f%%",
cm$byClass["Sensitivity"] * 100),
                " - How many actual positives were correctly identified. Important when
missing positives is costly."),
        tags$li(tags$strong("Specificity: "), sprintf("%.1f%%", cm$byClass["Specificity"]
* 100),
                " - How many actual negatives were correctly identified. Important when false
alarms are costly."),
        tags$li(tags$strong("ROC AUC: "), "Area under ROC curve. 0.5 = random
guessing, 1.0 = perfect. >0.7 is decent, >0.8 is good.")
      )
    ),
    tags$div(class = 'alert alert-warning',
      tags$strong("💡 Try These Experiments:"),
      tags$ul(
        tags$li("Change train/test split ratio and observe how it affects test accuracy"),
        tags$li("Increase CV folds for more reliable validation (slower but more robust)"),
        tags$li("Toggle preprocessing steps to see which improve performance"),
        tags$li("For Random Forest: increase trees for better performance (diminishing
returns after ~500)"),
        tags$li("Compare GLM vs Random Forest using the comparison mode")
      )
    )
  )
  insights
})

output$model_summary <- renderPrint({
  req(model_store$trained)
```

```
  # Show the primary model and (if present) comparison results
  if(!is.null(model_store$all_models) && length(model_store$all_models) > 1){
    cat("Trained models (comparison):\n")
    lapply(names(model_store$all_models), function(nm){
      cat(paste0("--- ", nm, " ---\n"))
      print(model_store$all_models[[nm]])
      cat('\n')
    })
  } else {
    print(model_store$model)
  }
 })

 output$conf_matrix <- renderPrint({
  req(model_store$trained)
  print(model_store$conf)
 })

 output$featImportance <- renderPlot({
  req(model_store$trained)
  # build importance plot as a ggplot object so it can be downloaded
  imp_plot_obj <- local({
    imp_obj <- model_store$importance
    if(is.null(imp_obj)) return(NULL)
    imp_df <- NULL; vals <- NULL; vars <- NULL
    try({
      if(is.list(imp_obj) && !is.null(imp_obj$importance)){
        imp_df <- as.data.frame(imp_obj$importance)
        vars <- rownames(imp_df)
        if("Overall" %in% colnames(imp_df)) vals <- imp_df$Overall else vals <-
rowMeans(imp_df, na.rm = TRUE)
      } else if(is.data.frame(imp_obj)){
        imp_df <- imp_obj
        vars <- rownames(imp_df)
        if(ncol(imp_df) == 1) vals <- imp_df[[1]] else vals <- rowMeans(imp_df, na.rm =
TRUE)
      } else if(!is.null(model_store$model$finalModel) && "randomForest" %in%
class(model_store$model$finalModel)){
        rf_imp <- tryCatch({randomForest::importance(model_store$model$finalModel)},
error = function(e) NULL)
        if(!is.null(rf_imp) && is.matrix(rf_imp)){
          vars <- rownames(rf_imp)
          vals <- if("MeanDecreaseGini" %in% colnames(rf_imp)) rf_imp[,
"MeanDecreaseGini"] else rowMeans(rf_imp, na.rm = TRUE)
        }
      }
    })
```

```
    if(is.null(vars) || length(vars) == 0) return(NULL)
    df_imp_plot <- data.frame(variable = vars, importance = as.numeric(vals),
stringsAsFactors = FALSE)
    df_imp_plot <- df_imp_plot %>% arrange(desc(importance)) %>% mutate(pct = 100
* importance / sum(importance, na.rm = TRUE))
    p <- ggplot(df_imp_plot, aes(x = reorder(variable, pct), y = pct)) +
      geom_col(fill = "#2c7fb8") + coord_flip() +
      labs(x = NULL, y = "Importance (%)", title = "Feature importance (relative %)") +
      geom_text(aes(label = sprintf("%.1f%%", pct)), hjust = -0.1, size = 3) +
      theme_minimal() + theme(plot.margin = ggplot2::margin(5, 40, 5, 5)) +
      scale_y_continuous(expand = expansion(mult = c(0, .15)))
    return(p)
   })
   if(is.null(imp_plot_obj)){
    plot.new(); text(0.5,0.5, "Feature importance could not be extracted")
    return()
   }
   print(imp_plot_obj)
  })

  # download handler for feature importance PNG
  output$downloadFeat <- downloadHandler(
   filename = function(){ paste0('feature_importance_',
tools::file_path_sans_ext(input$dataset), '.png') },
   content = function(file){
    # reconstruct the plot object same as above
    imp_obj <- model_store$importance
    if(is.null(imp_obj)){
     png(file); plot.new(); text(0.5,0.5, 'No feature importance'); dev.off(); return()
    }
    # reuse the plotting logic to build p
    p <- tryCatch({
     # build df
     imp_df <- NULL; vals <- NULL; vars <- NULL
     if(is.list(imp_obj) && !is.null(imp_obj$importance)){
      imp_df <- as.data.frame(imp_obj$importance); vars <- rownames(imp_df); vals <-
if('Overall' %in% colnames(imp_df)) imp_df$Overall else rowMeans(imp_df, na.rm =
TRUE)
     } else if(is.data.frame(imp_obj)){
      imp_df <- imp_obj; vars <- rownames(imp_df); vals <- if(ncol(imp_df)==1)
imp_df[[1]] else rowMeans(imp_df, na.rm = TRUE)
     } else if(!is.null(model_store$model$finalModel) && 'randomForest' %in%
class(model_store$model$finalModel)){
      rf_imp <- tryCatch({randomForest::importance(model_store$model$finalModel)},
error = function(e) NULL)
```

```
        if(!is.null(rf_imp) && is.matrix(rf_imp)){ vars <- rownames(rf_imp); vals <-
if('MeanDecreaseGini' %in% colnames(rf_imp)) rf_imp[,'MeanDecreaseGini'] else
rowMeans(rf_imp, na.rm = TRUE) }
       }
       if(is.null(vars) || length(vars)==0) stop('no importance')
       df_imp_plot <- data.frame(variable = vars, importance = as.numeric(vals),
stringsAsFactors = FALSE)
       df_imp_plot <- df_imp_plot %>% arrange(desc(importance)) %>% mutate(pct = 100
* importance / sum(importance, na.rm = TRUE))
       ggplot(df_imp_plot, aes(x = reorder(variable, pct), y = pct)) + geom_col(fill =
'#2c7fb8') + coord_flip() + labs(x = NULL, y = 'Importance (%)', title = 'Feature
importance (relative %)') + geom_text(aes(label = sprintf('%.1f%%', pct)), hjust = -0.1,
size = 3) + theme_minimal()
     }, error = function(e) NULL)
     if(is.null(p)){
       png(file); plot.new(); text(0.5,0.5, 'Feature importance could not be extracted');
dev.off(); return()
     }
     # save via png device
     png(file, width = 1200, height = 800)
     print(p)
     dev.off()
   }
 )

 output$rocPlot <- renderPlotly({
   req(model_store$trained)
   roc_obj <- model_store$roc
   if(is.null(roc_obj)) return(NULL)
   df_roc <- data.frame(tpr = rev(roc_obj$sensitivities), fpr = rev(1 -
roc_obj$specificities))
   p <- ggplot(df_roc, aes(x = fpr, y = tpr)) + geom_line() + geom_abline(linetype =
"dashed") + labs(title = paste0('ROC AUC = ', round(auc(roc_obj), 3)))
   safe_ggplotly(p)
 })

 # EDA plots
 output$ageDist <- renderPlotly({
   df <- data()
   req(df)
   # pick a numeric column if exists named age, otherwise first numeric
   if("age" %in% names(df) && is.numeric(df$age)){
       p <- ggplot(df, aes(x = age)) + geom_histogram(bins = 20, fill = "steelblue") +
labs(title = "Age distribution")
       safe_ggplotly(p)
   } else {
     nums <- df %>% select(where(is.numeric))
```

```
    if(ncol(nums) == 0) return(NULL)
    coln <- names(nums)[1]
  p <- ggplot(df, aes(x = .data[[coln]])) + geom_histogram(bins = 20, fill = "steelblue") +
labs(title = paste0(coln, " distribution"))
  safe_ggplotly(p)
   }
  })

  output$corrPlot <- renderPlotly({
   df <- data()
   req(df)
   nums <- df %>% select(where(is.numeric))
   if(ncol(nums) < 2) return(NULL)
   # remove constant columns (sd == 0) to avoid cor() warnings/errors
   sds <- sapply(nums, function(x) sd(x, na.rm = TRUE))
   keep <- names(sds)[which(!is.na(sds) & sds > 0)]
   if(length(keep) < 2) return(NULL)
   nums2 <- nums %>% select(all_of(keep))
   corr <- round(cor(nums2, use = "pairwise.complete.obs"), 2)
   plot_ly(z = corr, x = colnames(corr), y = rownames(corr), type = "heatmap", colorscale
= "Viridis") %>% layout(title = "Correlation heatmap")
  })

  output$catVsTarget <- renderPlotly({
   df <- data()
   req(df)
   if(is.null(input$target)) return(NULL)
   # find a categorical variable other than target
   cats <- names(df)[sapply(df, is.factor) & names(df) != input$target]
   if(length(cats) == 0) return(NULL)
   var <- cats[1]
  p <- ggplot(df, aes(x = .data[[var]], fill = .data[[input$target]])) + geom_bar(position =
"fill") + labs(y = "Proportion", title = paste0(var, " vs ", input$target))
  safe_ggplotly(p)
  })

  # Main interactive EDA plot (many plot types)
  output$eda_plot <- renderPlotly({
   req(data())
   df <- data()
   pt <- input$plot_type
   if(is.null(pt)) pt <- "Histogram"
   # safe selection helpers
   xvar <- input$eda_x
   yvar <- input$eda_y
   catvar <- input$eda_cat
```

```
  # use target if available for coloring
  color_by <- NULL
  if(!is.null(input$target) && input$target %in% names(df)) {
    color_by <- input$target
  }

  # ensure selected variables exist in dataset
  if(!is.null(xvar) && !xvar %in% names(df)) xvar <- NULL
  if(!is.null(yvar) && !yvar %in% names(df)) yvar <- NULL
  if(!is.null(catvar) && !catvar %in% names(df)) catvar <- NULL

  p <- NULL
  # read appearance controls with safe defaults
  bins_in <- if(!is.null(input$bins)) as.integer(input$bins) else 30
  hist_fill <- if(!is.null(input$hist_fill)) input$hist_fill else "steelblue"
  alpha_in <- if(!is.null(input$alpha)) input$alpha else 0.8
  point_size <- if(!is.null(input$point_size)) input$point_size else 2
  add_jitter <- isTRUE(input$add_jitter)
  jitter_width <- if(!is.null(input$jitter_width)) input$jitter_width else 0.2
  add_smooth <- isTRUE(input$add_smooth)
  smooth_method <- if(!is.null(input$smooth_method)) input$smooth_method else
"loess"
  smooth_span <- if(!is.null(input$smooth_span)) input$smooth_span else 0.75
  axis_transform <- if(!is.null(input$axis_transform)) input$axis_transform else "none"

  try({
    if(pt == "Histogram"){
      req(xvar)
      if(!is.null(color_by) && color_by %in% names(df)){
        p <- ggplot(df, aes(x = .data[[xvar]], fill = .data[[color_by]])) +
geom_histogram(alpha = alpha_in, bins = bins_in) + labs(title = paste0("Histogram of ",
xvar))
      } else {
        p <- ggplot(df, aes(x = .data[[xvar]])) + geom_histogram(alpha = alpha_in, bins =
bins_in, fill = hist_fill) + labs(title = paste0("Histogram of ", xvar))
      }
      if(axis_transform != "none") p <- p + scale_x_continuous(trans = axis_transform)
    } else if(pt == "Density"){
      req(xvar)
      if(!is.null(color_by) && color_by %in% names(df)){
        p <- ggplot(df, aes(x = .data[[xvar]], color = .data[[color_by]], fill =
.data[[color_by]])) + geom_density(alpha = alpha_in) + labs(title = paste0("Density of ",
xvar))
      } else {
        p <- ggplot(df, aes(x = .data[[xvar]])) + geom_density(fill = hist_fill, alpha =
alpha_in) + labs(title = paste0("Density of ", xvar))
      }
```

```
      if(axis_transform != "none") p <- p + scale_x_continuous(trans = axis_transform)
    } else if(pt == "Boxplot"){
      req(xvar)
      if(!is.null(color_by) && color_by %in% names(df)){
        p <- ggplot(df, aes(x = .data[[color_by]], y = .data[[xvar]], fill = .data[[color_by]]))
+ geom_boxplot()
        if(add_jitter) p <- p + geom_jitter(position = position_jitter(width = jitter_width),
size = point_size, alpha = alpha_in)
        p <- p + labs(title = paste0("Boxplot of ", xvar, " by ", color_by))
      } else {
        p <- ggplot(df, aes(x = "", y = .data[[xvar]])) + geom_boxplot(fill = hist_fill) +
labs(title = paste0("Boxplot of ", xvar), x = NULL)
        if(add_jitter) p <- p + geom_jitter(width = jitter_width, size = point_size, alpha =
alpha_in)
      }
      }
      if(axis_transform != "none") p <- p + scale_y_continuous(trans = axis_transform)
    } else if(pt == "Violin"){
      req(xvar)
      if(!is.null(color_by) && color_by %in% names(df)){
        p <- ggplot(df, aes(x = .data[[color_by]], y = .data[[xvar]], fill = .data[[color_by]]))
+ geom_violin(alpha = alpha_in)
        if(add_jitter) p <- p + geom_jitter(position = position_jitter(width = jitter_width),
size = point_size, alpha = alpha_in*0.8)
        p <- p + labs(title = paste0("Violin of ", xvar, " by ", color_by))
      } else {
        p <- ggplot(df, aes(x = "", y = .data[[xvar]])) + geom_violin(fill = hist_fill, alpha =
alpha_in) + labs(title = paste0("Violin of ", xvar), x = NULL)
        if(add_jitter) p <- p + geom_jitter(width = jitter_width, size = point_size, alpha =
alpha_in*0.8)
      }
      }
      if(axis_transform != "none") p <- p + scale_y_continuous(trans = axis_transform)
    } else if(pt == "Bar"){
      req(catvar)
      p <- ggplot(df, aes(x = .data[[catvar]])) + geom_bar(fill = hist_fill) + labs(title =
paste0("Count of ", catvar))
    } else if(pt == "Stacked Bar"){
      req(catvar)
      req(color_by)
      p <- ggplot(df, aes(x = .data[[catvar]], fill = .data[[color_by]])) + geom_bar(position
= "fill") + labs(y = "Proportion", title = paste0(catvar, " by ", color_by))
    } else if(pt == "Scatter"){
      req(xvar, yvar)
      if(is.null(xvar) || is.null(yvar) || !xvar %in% names(df) || !yvar %in% names(df)) {
        p <- ggplot() + geom_text(aes(x = 0.5, y = 0.5, label = "Please select valid X and Y
variables"), size = 5) + theme_void()
```

```
      } else if(!is.null(color_by) && color_by %in% names(df)){
        if(add_jitter) p <- ggplot(df, aes(x = .data[[xvar]], y = .data[[yvar]], color =
.data[[color_by]])) + geom_jitter(width = jitter_width, height = jitter_width, size =
point_size, alpha = alpha_in)
        else p <- ggplot(df, aes(x = .data[[xvar]], y = .data[[yvar]], color =
.data[[color_by]])) + geom_point(size = point_size, alpha = alpha_in)
        if(add_smooth) {
          if(smooth_method == "loess") p <- p + geom_smooth(method = "loess", span =
smooth_span, se = FALSE)
          else p <- p + geom_smooth(method = "lm", se = FALSE)
        }
      } else {
        if(add_jitter) p <- ggplot(df, aes(x = .data[[xvar]], y = .data[[yvar]])) +
geom_jitter(width = jitter_width, height = jitter_width, size = point_size, alpha =
alpha_in)
        else p <- ggplot(df, aes(x = .data[[xvar]], y = .data[[yvar]])) + geom_point(size =
point_size, alpha = alpha_in)
        if(add_smooth) {
          if(smooth_method == "loess") p <- p + geom_smooth(method = "loess", span =
smooth_span, se = FALSE)
          else p <- p + geom_smooth(method = "lm", se = FALSE)
        }
      }
      if(axis_transform != "none"){
        p <- p + scale_x_continuous(trans = axis_transform) + scale_y_continuous(trans =
axis_transform)
      }
    } else if(pt == "Pair Plot"){
      # Pair plot handled by static GGally::ggpairs in a separate output; return NULL here.
      return(NULL)
    } else if(pt == "Missing Map"){
      miss_counts <- sapply(df, function(x) sum(is.na(x)))
      miss_df <- data.frame(variable = names(miss_counts), missing =
as.integer(miss_counts))
      p <- ggplot(miss_df, aes(x = reorder(variable, -missing), y = missing)) +
geom_bar(stat = 'identity', fill = 'salmon') + coord_flip() + labs(title = 'Missing values per
column', x = '', y = 'Missing count')
    } else if(pt == "Correlation Heatmap"){
      nums <- numeric_vars()
      if(length(nums) < 2) return(NULL)
      corr <- round(cor(df %>% select(all_of(nums)), use = 'pairwise.complete.obs'), 2)
      p <- plot_ly(z = corr, x = colnames(corr), y = rownames(corr), type = 'heatmap',
colorscale = 'Blues') %>% layout(title = 'Correlation Heatmap')
      return(p)
    }
  })
  if(is.null(p)) return(NULL)
```

```
    # If p is already a plotly object, return it directly
    if(inherits(p, "plotly")) return(p)
    # Convert ggplot to plotly safely and return a friendly message on failure
    safe_ggplotly(p)
  })

  # Boxplot of first numeric predictor grouped by target (learning: distribution
comparisons)
  output$boxByTarget <- renderPlotly({
    df <- data()
    req(df)
    if(is.null(input$target)) return(NULL)
    nums <- df %>% select(where(is.numeric))
    if(ncol(nums) == 0) return(NULL)
    var <- names(nums)[1]
    p <- ggplot(df, aes(x = .data[[input$target]], y = .data[[var]], fill =
.data[[input$target]])) + geom_boxplot() + labs(title = paste0('Boxplot of ', var, ' by ',
input$target))
    safe_ggplotly(p)
  })

  # Density plot of first numeric predictor by target
  output$densityByTarget <- renderPlotly({
    df <- data()
    req(df)
    if(is.null(input$target)) return(NULL)
    nums <- df %>% select(where(is.numeric))
    if(ncol(nums) == 0) return(NULL)
    var <- names(nums)[1]
    p <- ggplot(df, aes(x = .data[[var]], color = .data[[input$target]], fill =
.data[[input$target]])) + geom_density(alpha = 0.3) + labs(title = paste0('Density of ', var,
' by ', input$target))
    safe_ggplotly(p)
  })

  # static GGally pair plot (rich ggpairs view) - falls back gracefully if GGally not
installed
  # Build pair plot object so it can be rendered and downloaded
  pair_plot_obj <- reactive({
    df <- data()
    req(df)
    nums <- numeric_vars()
    if(length(nums) < 2) return(list(type = 'none', plot = NULL, note = 'Not enough
numeric variables'))
    # if too many numerics, choose top variables by variance to keep plot readable
    if(length(nums) > 12){
      vars_sd <- sapply(df %>% select(all_of(nums)), function(x) sd(x, na.rm = TRUE))
```

```
    topn <- names(sort(vars_sd, decreasing = TRUE))[1:12]
    nums_sel <- topn
    note <- paste0('Selected top ', length(nums_sel), ' numeric vars by variance for
plotting')
   } else {
    nums_sel <- nums
    note <- NULL
   }
   # prefer GGally::ggpairs if available
   if(requireNamespace('GGally', quietly = TRUE)){
    g <- tryCatch({
     GGally::ggpairs(df %>% select(all_of(nums_sel)), progress = FALSE, upper =
list(continuous = GGally::wrap('cor', size = 3)))
    }, error = function(e){
     NULL
    })
    if(!is.null(g)) return(list(type = 'ggpairs', plot = g, note = note))
   }
   # fallback to base pairs
   return(list(type = 'pairs', plot = df %>% select(all_of(nums_sel)), note = note))
  })

  output$pairPlotStatic <- renderPlot({
   obj <- pair_plot_obj()
   if(is.null(obj) || obj$type == 'none'){
    plot.new(); text(0.5,0.5, 'Not enough numeric variables for pair plot')
    return()
   }
   if(obj$type == 'ggpairs'){
    print(obj$plot)
   } else if(obj$type == 'pairs'){
    pairs(obj$plot, main = 'Pairwise scatter (fallback)')
   }
  })

  # download handler for pair plot (PNG)
  output$downloadPair <- downloadHandler(
   filename = function(){ paste0('pairplot_', tools::file_path_sans_ext(input$dataset),
'.png') },
   content = function(file){
    obj <- pair_plot_obj()
    if(is.null(obj) || obj$type == 'none'){
     png(file); plot.new(); text(0.5,0.5,'Not enough numeric variables'); dev.off(); return()
    }
    # create PNG device and print plot
    png(file, width = 1600, height = 1600)
    try({
```

```
      if(obj$type == 'ggpairs') print(obj$plot) else pairs(obj$plot, main = 'Pairwise scatter
(fallback)')
    }, silent = TRUE)
    dev.off()
  }
 )

 # dynamic prediction input UI generated from predictors of trained model
 output$predict_inputs_ui <- renderUI({
  req(model_store$trained)
  model <- model_store$model
  train_df <- model_store$train_data
  preds <- predictors(model)
  if(length(preds) == 0) return(div("No predictors found."))
  inputs <- map(preds, function(var){
   if(is.factor(train_df[[var]])){
    selectInput(paste0("pred_", var), var, choices = levels(train_df[[var]]))
   } else if(is.numeric(train_df[[var]])){
    numericInput(paste0("pred_", var), var, value = round(mean(train_df[[var]], na.rm =
TRUE), 2))
   } else {
    textInput(paste0("pred_", var), var, value = as.character(train_df[[var]][1]))
   }
  })
  do.call(tagList, inputs)
 })

 # placeholder for prediction output so outputOptions can be set safely
 output$prediction_result <- renderPrint({
  cat("No prediction yet. Use the prediction UI after training a model.")
 })

 # prediction action
 observeEvent(input$predict_btn, {
  req(model_store$trained)
  model <- model_store$model
  preds <- predictors(model)

  # FIXED: Validate all required inputs are present
  missing_inputs <- c()
  for(var in preds) {
   input_id <- paste0("pred_", var)
   if(is.null(input[[input_id]])) {
    missing_inputs <- c(missing_inputs, var)
   }
  }
```

```
  if(length(missing_inputs) > 0) {
    showNotification(paste("❌ Missing inputs for:", paste(missing_inputs, collapse = ",
")),
                     type = "error", duration = 10)
    return()
  }

  # Build newrow with validation
  newrow <- tryCatch({
    map(preds, function(var){
      value <- input[[paste0("pred_", var)]]
      # coerce to same type as training
      train_df <- model_store$train_data
      if(is.factor(train_df[[var]])) return(factor(value, levels = levels(train_df[[var]])))
      if(is.numeric(train_df[[var]])) {
        num_val <- as.numeric(value)
        if(is.na(num_val)) stop(paste("Invalid numeric value for", var))
        return(num_val)
      }
      return(value)
    }) %>% set_names(preds) %>% as.data.frame()
  }, error = function(e) {
    showNotification(paste("❌ Error creating prediction input:", e$message), type =
"error", duration = 10)
    NULL
  })

  if(is.null(newrow)) return()

  # If a recipe was used, apply the same preprocessing to the new observation before
predicting
  if(!is.null(model_store$recipe)){
    newrow_trans <- tryCatch({
      bake(model_store$recipe, new_data = newrow)
    }, error = function(e) {
      showNotification(paste("⚠️ Warning: Could not apply preprocessing to new data.
Using original values.", e$message),
                       type = "warning", duration = 10)
      newrow
    })
  } else {
    newrow_trans <- newrow
  }

  # Make prediction with error handling
  pred_class <- tryCatch({
```

```
      predict(model, newdata = newrow_trans)
    }, error = function(e) {
      showNotification(paste("❌ Prediction error:", e$message), type = "error", duration =
10)
      NULL
    })

    if(is.null(pred_class)) return()

    prob <- tryCatch({predict(model, newdata = newrow_trans, type = "prob")}, error =
function(e) NULL)

    output$prediction_result <- renderPrint({
      cat("Predicted class:", as.character(pred_class), "\n")
      if(!is.null(prob)){
        print(prob)
      }
    })

    output$downloadData <- downloadHandler(
      filename = function() { paste0("prediction_", tools::file_path_sans_ext(input$dataset),
".csv") },
      content = function(file){
        out <- bind_cols(newrow, Prediction = as.character(pred_class))
        if(!is.null(prob)) out <- bind_cols(out, prob)
        write.csv(out, file, row.names = FALSE)
      }
    )
  })

  # allow user to download the trained primary model as an RDS
  output$downloadModel <- downloadHandler(
    filename = function(){ paste0('trained_model_',
tools::file_path_sans_ext(input$dataset), '.rds') },
    content = function(file){
      req(model_store$trained)
      saveRDS(list(model = model_store$model, recipe = model_store$recipe), file)
    }
  )

  # expose prediction result output so it shows in UI
  outputOptions(output, "prediction_result", suspendWhenHidden = FALSE)

}

shinyApp(ui, server)
```

## 11. Technology Used

### 11.1 Core Technologies

| Category | Technology | Version | Purpose |
|---|---|---|---|
| Programming Language | R | 4.4.1+ | Statistical computing and analysis |
| Web Framework | Shiny | 1.7.0+ | Interactive web application development |
| Data Manipulation | tidyverse | 2.0.0+ | Data wrangling and transformation |
| Machine Learning | caret | 6.0-90+ | Unified ML training interface |
| Visualization | ggplot2 + plotly | 3.4.0+ | Interactive plotting |

Table 2: Core Technologies Used

### 11.2 Supporting Libraries

| Package | Purpose | Key Functions |
|---|---|---|
| DT | Interactive tables | datatable(), renderDataTable() |
| recipes | Feature engineering | recipe(), prep(), bake() |
| randomForest | Ensemble modeling | randomForest(), importance() |
| pROC | ROC analysis | roc(), auc(), ci() |
| bslib | Modern UI themes | bs_theme(), page_sidebar() |

Table 3: Supporting R Packages

### 11.3 Development Environment

- IDE: RStudio 2023.06.0+
- Version Control: Git with GitHub integration
- Package Management: renv for reproducible environments
- Testing Framework: testthat for unit testing
- Documentation: roxygen2 for function documentation

## 12. Implementation Details

### 12.1 Key Algorithms Implemented

**Binary Target Detection Algorithm**

```
detect_binary_targets <- function(data) {
candidates <- data %>%
select_if(~ is.factor(.) && length(levels(.)) == 2 ||
is.character(.) && length(unique(na.omit(.))) == 2 ||
is.numeric(.) && all(unique(na.omit(.)) %in% c(0, 1)))
return(names(candidates))
}
```

**Adaptive Preprocessing Pipeline**

- Missing Value Strategy: Median imputation for numeric, mode for categorical
- Scaling Strategy: Z-score normalization with outlier detection
- Encoding Strategy: One-hot encoding for high-cardinality categories

**Model Training Framework**

```
train_models <- function(formula, data, methods = c("glm", "rf")) {
control <- trainControl(
method = "cv",
number = 5,
classProbs = TRUE,
summaryFunction = twoClassSummary,
savePredictions = TRUE
)
models <- map(methods, ~train(formula, data, method = ., trControl = control))
return(models)
}
```

### 12.2 Performance Optimizations

1. Data Sampling: Configurable sampling for large datasets (default: 10,000 rows)
2. Reactive Debouncing: 500ms delay on user inputs to prevent excessive computations
3. Cached Computations: Expensive operations cached using reactive values
4. Progressive Loading: UI elements loaded incrementally to improve perceived performance

### 12.3 Error Handling and Validation

- Input Validation: Type checking and range validation for all user inputs
- Graceful Degradation: Fallback options when primary algorithms fail
- User Feedback: Informative error messages and progress indicators
- Data Validation: Automatic detection of data quality issues with user warnings

## 13. Results

### 13.1 Functional Testing Results

The dashboard was tested with 6 different mental health datasets:

| Dataset | Rows | Columns | Binary Targets | Processing Time | Model Accuracy (RF) |
|---|---|---|---|---|---|
| Mental Health Survey | 1,259 | 27 | 3 | 2.3s | 89.2% |
| Suicide Rates Dataset | 27,820 | 12 | 1 | 8.7s | 91.5% |
| Workplace Stress Data | 500 | 15 | 2 | 1.1s | 87.4% |
| Depression Screening | 2,100 | 22 | 1 | 3.2s | 85.8% |
| Anxiety Assessment | 1,800 | 18 | 1 | 2.8s | 88.9% |
| Crisis Intervention | 950 | 25 | 2 | 1.9s | 92.1% |

Table 4: Benchmark Results for Dashboard Testing

### 13.2 Performance Benchmarks

**System Performance**

- Average Load Time: 3.2 seconds for datasets under 5,000 rows
- Memory Usage: 150–300 MB depending on dataset size
- Concurrent Users: Tested successfully with 5 simultaneous users
- Browser Compatibility: 100% compatibility across major browsers

**Model Performance**

- Random Forest: Average AUC of 0.89 across all datasets
- Logistic Regression: Average AUC of 0.83 across all datasets
- Feature Importance: Successfully identified top predictors in 100% of cases
- Prediction Accuracy: 87.2% average accuracy for binary classification

### 14.1 Scalability Analysis

The system demonstrates good performance characteristics:

- Linear Scaling: Processing time scales linearly with dataset size

- Memory Efficiency: Optimized data structures reduce memory footprint by 40%
- Computational Efficiency: Vectorized operations achieve 3x speedup over base implementations

### 14.2 Model Accuracy Comparison

| Algorithm | Average AUC | Training Time | Interpretability | Best Use Case |
|---|---|---|---|---|
| Logistic Regression | 0.83 | 0.8s | High | Linear relationships, feature importance |
| Random Forest | 0.89 | 4.2s | Medium | Non-linear patterns, robust predictions |

Table 5: Model Accuracy and Comparison

### 14.3 Bottleneck Analysis

- Primary Bottleneck: Large dataset visualization ($>$50,000 points)
- Secondary Bottleneck: Cross-validation for Random Forest models
- Mitigation Strategies: Sampling for visualization, parallel processing for CV

## 15. Limitations

### 15.1 Technical Limitations

1. Single-session Processing: No persistence between user sessions
2. Memory Constraints: Performance degrades with datasets $>$100,000 rows
3. Binary Classification Only: Multi-class problems not currently supported
4. Limited File Formats: Only CSV files supported
5. No Real-time Updates: Static analysis without live data integration

### 15.2 Methodological Limitations

1. Algorithm Selection: Limited to GLM and Random Forest algorithms
2. Feature Selection: No automated feature selection capabilities
3. Hyperparameter Tuning: Basic grid search implementation only
4. Cross-validation: Fixed 5-fold CV, no alternative strategies
5. Imbalanced Data: No specialized handling for severely imbalanced datasets

### 15.3 User Interface Limitations

1. Mobile Responsiveness: Optimized for desktop use only
2. Accessibility: Limited support for screen readers and accessibility tools
3. Language Support: English interface only
4. Help System: No comprehensive in-app help or tutorials

## 17. Conclusion

### 17.1 Project Summary

This project successfully developed a comprehensive Mental Health Analytics Dashboard that addresses the critical gap between data collection and actionable insights in mental health research and practice. The R Shiny-based platform provides an intuitive, no-code environment for complex data analysis, achieving the primary objective of democratizing mental health analytics.

The system bridges the gap between raw data and evidence-based insights by automating:

- Data ingestion and preprocessing workflows
- Exploratory data analysis with interactive visualizations
- Predictive modeling using machine learning algorithms
- Performance evaluation and model interpretation
- Result export and integration capabilities

GitHub link:
https://github.com/JosephJonathanFernandes/Mental_Health_Dashboard_In_R