R Documentation

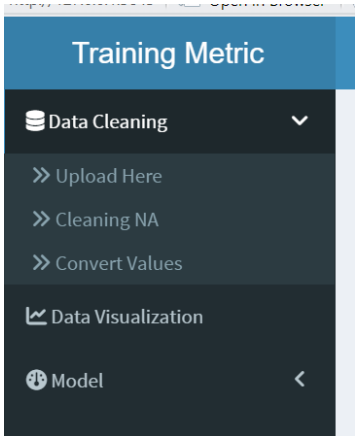
Menu 1 : Data Cleaning

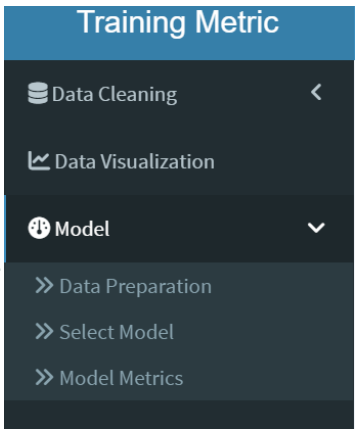
1. Tab : Upload Here
2. Tab : Cleaning Na
3. Tab : Convert Values

UI Structure in General

UI

```r
library(shiny)
library(shinydashboard)
library(shinythemes)
library(shiny)
library(tidyverse)
library(dplyr)
library(tidyr)
library(data.table)
library(DT)

source("trainers_classes.R") -> classes & objects from the modelling

ui <- dashboardPage(
  dashboardHeader(title="Training Metric Program"),
  dashboardSidebar(
    sidebarMenu(
      ## 01. DATA CLEANING
      menuItem("Data Cleaning", tabName = "preps", icon = icon("database"),
               menuSubItem("Upload Here", tabName="uploadcsv"),
               menuSubItem("Cleaning NA", tabName="cleaning"),
               menuSubItem("Convert Values", tabName="convertVal")),
                                                  (1)
      ## 02. DATA VISUALIZAION
      menuItem("Data Visualization", tabName = "vis", icon = icon("line-chart")),
      ## 03. MODEL METRICS
      menuItem("Model", tabName = "model", icon = icon("dashboard"),
               menuSubItem("Data Preparation", tabName="modelpreparation"),
               menuSubItem("Select Model", tabName="selectmodel"),
               menuSubItem("Model Metrics", tabName="metrics"))
    )
  ),
```





01. Tab 1 : Upload Here

UI

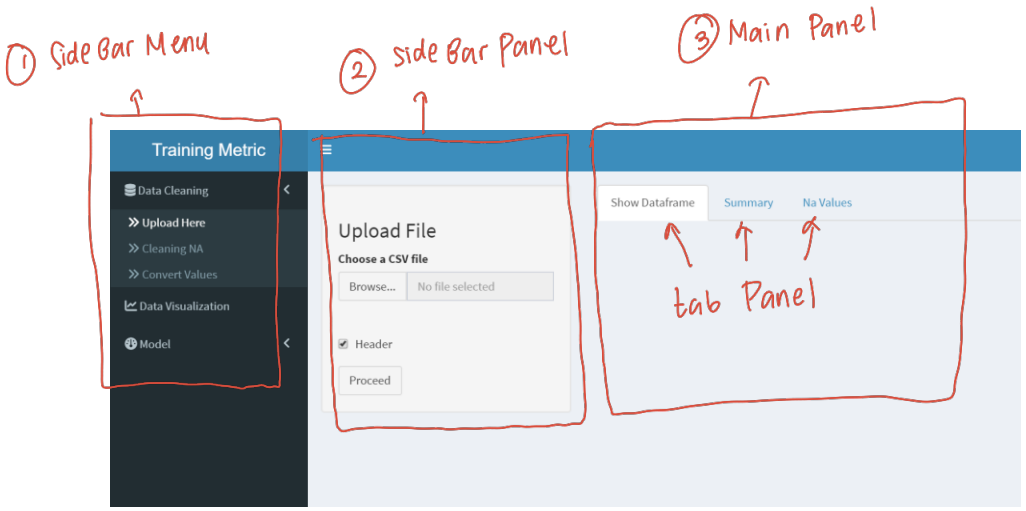
```
ui <- dashboardPage(
  dashboardHeader(title="Training Metric Program"),
  dashboardSidebar(
    sidebarMenu(
      ## 01. DATA CLEANING
      menuItem("Data Cleaning", tabName = "preps", icon = icon("database"),
               menuSubItem("Upload Here", tabName="uploadcsv"),
               menuSubItem("Cleaning NA", tabName="cleaning"),
               menuSubItem("Convert Values", tabName="convertVal")),

      ## 02. DATA VISUALIZAION
      menuItem("Data Visualization", tabName = "vis", icon = icon("line-chart")),
      ## 03. MODEL METRICS
      menuItem("Model", tabName = "model", icon = icon("dashboard"),
               menuSubItem("Data Preparation", tabName="modelpreparation"),
               menuSubItem("Select Model", tabName="selectmodel"),
               menuSubItem("Model Metrics", tabName="metrics"))
    )
  ),
```

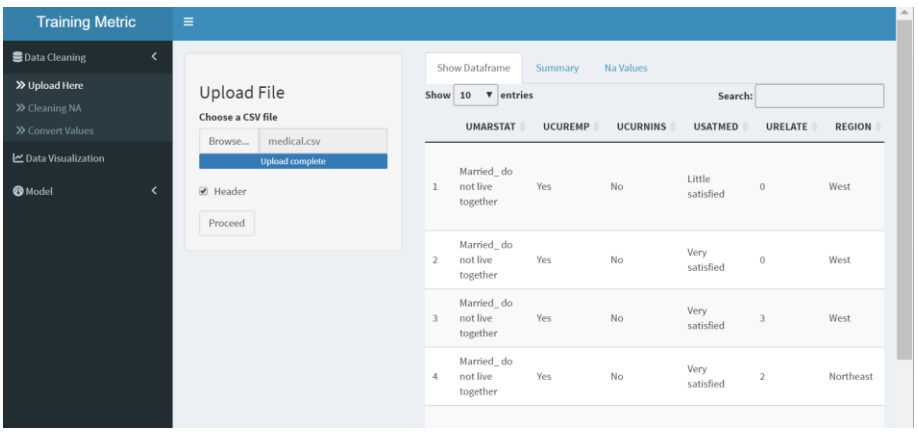
tabName

UI

```
dashboardBody(
  tabItems(
    ## 01. DATA CLEANING - TAB: UPLOAD CSV
    tabItem(tabName = "uploadcsv",
            sidebarLayout(
              sidebarPanel(
                h3("Upload File"),
                fileInput(inputId = "file", label="Choose a CSV file", accept=".csv"),
                checkboxInput("header", "Header", TRUE),
                actionButton("UploadButton", "Proceed")
              ),
              mainPanel(
                tabsetPanel(
                  tabPanel("Show Dataframe",DT::dataTableOutput("data")),
                  tabPanel("Summary",verbatimTextOutput("summary")),
                  tabPanel("Na Values",tableOutput("NaData"))
                )
              )
            )
    ),
```

tabName → it is like an id for the tab menu
So whenever we'd like to design the layout of 1 tab menu,
we call the tabName ID





01. Tab 1 : Upload Here

UI

```
dashboardBody(
  tabItems(
    ## 01. DATA CLEANING - TAB: UPLOAD CSV
    tabItem(tabName = "uploadcsv",
            sidebarLayout(
              sidebarPanel(
                h3("Upload File"),
                fileInput(inputId = "file", label="Choose a CSV file", accept=".csv"),
                checkboxInput("header", "Header", TRUE),
                actionButton("UploadButton", "Proceed")
              ),
              mainPanel(
                tabsetPanel(
                  tabPanel("Show Dataframe",DT::dataTableOutput("data")),
                  tabPanel("Summary",verbatimTextOutput("summary")),
                  tabPanel("Na Values",tableOutput("NaData"))
                )
              )
            )
    ),
```

(1) Input ID = "file"

UI → we are designing a panel where users can upload their csv file and naming it "file" to pass it to server

Server → calling the id : "file"

(2) actionButton "Upload Button"

in server, we are using an "observeEvent" as if : whenever the users click "Proceed" it will run the code inside the "observeEvent"

SERVER

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

  cleanedData <- reactiveVal()
  # 01. DATA PREPARATION - TAB: UPLOAD CSV
  data <- reactive({
    req(input$file)
    # Get the file extension
    ext <- tools::file_ext(input$file$name)
    # Check if the file extension is CSV
    if(ext == "csv") {
      fread(input$file$datapath, header = TRUE, stringsAsFactors = FALSE)
    } else {
      stop("Invalid file format. Please upload a CSV file.")
    }
  })
  observeEvent(data(), {
    cleanedData(data())
  })

  observeEvent(input$UploadButton, {
    # TAB 1 : DF.HEAD (5)
    headDf <- reactive({ cleanedData() %>% slice(1:5) })
    output$data <- DT::renderDataTable({ headDf() %>% datatable(options=list(scrollX=TRUE)) })
    # TAB 2 : DF STR
    summaryDf <- reactive({str(cleanedData())})
    output$summary <- renderPrint({summaryDf()})
    # TAB 3 : SHOW NA DATA
    na_counts <- reactive({ colSums(is.na(cleanedData())) })
    na_cols <- names(na_counts())[na_counts() > 0]
    na_counts <- na_counts()[na_cols]
    NaDf <- data.frame(ColNames = na_cols, NAs = na_counts)
    output$NaData <- renderTable(NaDf)
  })
```

(3) (4) (5)

In UI, we are giving the ID of each output, so in server we can call which output ID to show.

UI	Server
(3) dataTable Output	renderDataTable
(4) verbatimText Output	renderPrint
(5) tableOutput	renderTable

(6) ReactiveVal

We assign the csv from users' uploaded csv to variable "data". But you may notice that we assign it in reactive function :

data ← reactive({ ... })

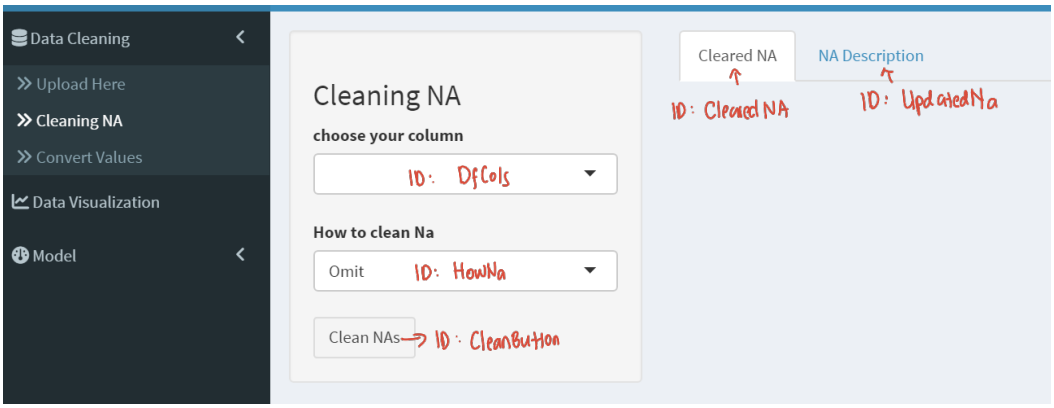
cleanedData ← reactiveVal()
is to store the dataframe of "data".

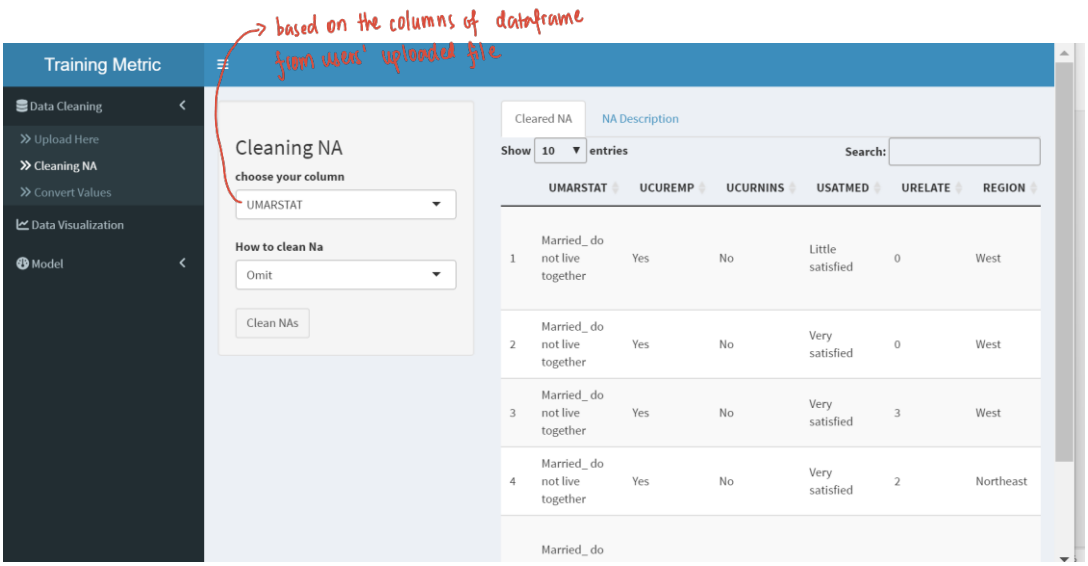
ps: read the page of reactiveVal :)

01. Tab 2 : Cleaning NA

UI

```
## 01. DATA CLEANING - TAB: CLEANING
tabItem(tabName="cleaning",
        sidebarLayout(
          sidebarPanel(
            h3("Cleaning NA"),
            selectInput(inputId = "DfCols",label="choose your column",choices=NULL,selected = NULL),
            selectInput(inputId = "HowNa",label="How to clean Na",choices=c("Omit", "Replace with Mean", "Replace with Median"),selected = "Omit"),
            actionButton("CleanButton", "Clean NAs")
          ),
          mainPanel(
            tabsetPanel(
              tabPanel("Cleared NA", DT::dataTableOutput("ClearedNA")),
              tabPanel("NA Description", tableOutput("UpdatedNa"))
            )
          )

        )),
```





01. Tab 2 : Cleaning NA

UI:

```
## 01. DATA CLEANING - TAB: CLEANING
tabItem(tabName="cleaning",
        sidebarLayout(
          sidebarPanel(
            h3("Cleaning NA"),
(1)         selectInput(inputId = "DfCols",label="choose your column",choices=NULL,selected = NULL),
(2)         selectInput(inputId = "HowNa",label="How to clean Na",choices=c("Omit", "Replace with Mean", "Replace with Median"),selected = "Omit"),
            actionButton("CleanButton", "Clean NAs")
          ),
          mainPanel(
            tabsetPanel(
              tabPanel("Cleared NA", DT::dataTableOutput("ClearedNA")),
              tabPanel("NA Description", tableOutput("UpdatedNa"))
            )
          )
        )),
```

(1) Dfcols : NULL
it is not defined in UI because we want it to appear according to the columns of df. Thus we need to define it in server

(2) If the options aren't referred to the columns of df. then the choices are defined in UI

SERVER :

```
# 01. DATA PREPARATION - TAB: CLEANING NA
# TAB 1 : DF.HEAD (5)
(1) observe({
      updateSelectInput(session, "DfCols", label="choose your column", choices=names(cleanedData()), selected=names(cleanedData())[1])
    })

    observeEvent(input$CleanButton, {
(1)   col <- input$DfCols
      how_na <- input$HowNa

      data_cleaned <- cleanedData()

      # Clean data based on user input
(2)   if (how_na == "Omit"){
        data_cleaned <- data_cleaned %>% drop_na({{col}})
      }
      else if (how_na == "Replace with Mean"){
        data_cleaned <- data_cleaned %>% mutate({{col}} := ifelse(is.na({{col}}), mean({{col}}, na.rm = TRUE), {{col}}))
      }
      else if (how_na == "Replace with Median"){
        data_cleaned <- data_cleaned %>% mutate({{col}} := ifelse(is.na({{col}}), median({{col}}, na.rm = TRUE), {{col}}))
      }

      # Update the reactive value with cleaned data
      cleanedData(data_cleaned)

      # Render cleaned data table
      output$ClearedNA <- DT::renderDataTable({data_cleaned %>% slice(1:5)%>% datatable(options=list(scrollX=TRUE))})

      # TAB 2 : SHOW REMAINING NA DATA
      na_counts <- reactive({ colSums(is.na(data_cleaned)) })
      na_cols <- names(na_counts())[na_counts() > 0]
      na_counts <- na_counts()[na_cols]
      NaDf <- data.frame(ColNames = na_cols, NAs = na_counts)
      output$UpdatedNa <- renderTable(NaDf)
    })
```

→ reactive val that stores "data()"

cols

y input of whatever users pick

(1) dfcols are defined here & use cleanedData() columns

(2) if-else conditions according to what the selected options are.

cleanedData() → previously it stores "data()"

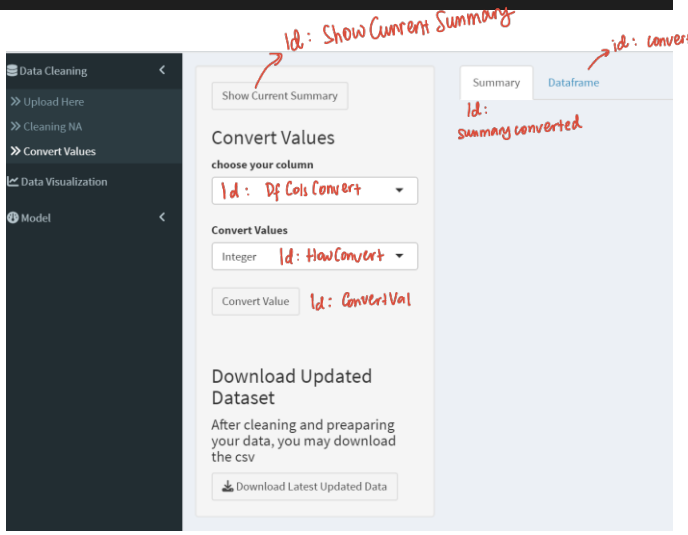
→ now that users clean their NA values in their df, we'd like to have an updated dataframe, so we update cleanedData(data_cleaned)

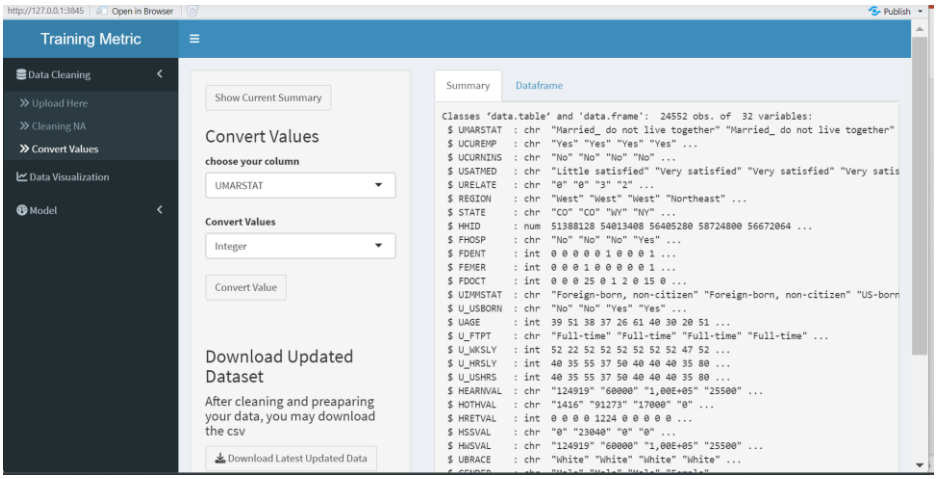
⇒ cleanedData() now stores a df w/ cleaned NA values

01. Tab 3 : Convert Values

UI

```
## 01. DATA CLEANING - TAB: CONVERT VALUES
tabItem(tabName="convertVal",
        sidebarLayout(
          sidebarPanel(
            actionButton("ShowCurrentSummary", "Show Current Summary"),
            h3("Convert Values"),
            selectInput(inputId = "DfColsConvert",label="choose your column",choices=NULL,selected = NULL),
            selectInput(inputId = "HowConvert",label="Convert Values",choices=c("String", "Integer", "Numeric","Factor","Date"),selected = "Integer"),
            actionButton("ConvertVal", "Convert Value"),
            br(),
            br(),
            br(),
            h3("Download Updated Dataset"),
            h4("After cleaning and preaparing your data, you may download the csv"),
            downloadButton("downloadData", "Download Latest Updated Data")
          ),
          mainPanel(
            tabsetPanel(
              tabPanel("Summary",verbatimTextOutput("summaryconverted")),
              tabPanel("Dataframe", DT::dataTableOutput("convertdata"))
            )
          )
        )),
```





everytime users click the button "convert value" the summary changes and keeps updated accordingly

01. Tab 3 : Convert Values

UI

```
## 01. DATA CLEANING - TAB: CONVERT VALUES
tabItem(tabName="convertVal",
        sidebarLayout(
          sidebarPanel(
            actionButton("ShowCurrentSummary", "Show Current Summary"),
            h3("Convert Values"),
            selectInput(inputId = "DfColsConvert",label="choose your column",choices=NULL,selected = NULL),
            selectInput(inputId = "HowConvert",label="Convert Values",choices=c("String", "Integer", "Numeric","Factor","Date"),selected = "Integer"),
            actionButton("ConvertVal", "Convert Value"),
            br(),
            br(),
            br(),
            h3("Download Updated Dataset"),
            h4("After cleaning and preaparing your data, you may download the csv"),
            downloadButton("downloadData", "Download Latest Updated Data")
          ),
          mainPanel(
            tabsetPanel(
              tabPanel("Summary",verbatimTextOutput("summaryconverted")),
              tabPanel("Dataframe", DT::dataTableOutput("convertdata"))
            )
          )
        )),
```

① Show Current Summary :
Basically it shows the datatypes of each columns before converting

② DfColsConvert
Choices & selected are defined in server

SERVER

```
# 01. DATA PREPARATION - TAB: CONVERT VALUES
observeEvent(input$ShowCurrentSummary, {
  output$summaryconverted <- renderPrint({str(cleanedData())})
  output$convertdata <- DT::renderDataTable({cleanedData() %>% slice(1:5)%>% datatable(options=list(scrollX=TRUE))})
})

observe({
  updateSelectInput(session, "DfColsConvert", label="choose your column", choices=names(cleanedData()), selected=names(cleanedData())[1])
})

observeEvent(input$ConvertVal, {
  col_convert <- input$DfColsConvert
  how_convert <- input$HowConvert

  currentCleanedDf <- cleanedData()

  if (how_convert=="Integer"){
    currentCleanedDf[[col_convert]] <- as.integer(currentCleanedDf[[col_convert]])
  } else if (how_convert == "String") {
    currentCleanedDf[[col_convert]] <- as.character(currentCleanedDf[[col_convert]])
  } else if (how_convert == "Numeric") {
    currentCleanedDf[[col_convert]] <- as.numeric(currentCleanedDf[[col_convert]])
  } else if (how_convert == "Factor") {
    currentCleanedDf[[col_convert]] <- as.factor(currentCleanedDf[[col_convert]])
  } else if (how_convert == "Date") {
    currentCleanedDf[[col_convert]] <- as.Date(currentCleanedDf[[col_convert]],format = "%Y%m%dT%H%M%S")
  }
  cleanedData(currentCleanedDf)
  output$summaryconverted <- renderPrint({str(cleanedData())})
  output$convertdata <- DT::renderDataTable({cleanedData() %>% slice(1:5)%>% datatable(options=list(scrollX=TRUE))})

  # still not working yet
  output$downloadData <-downloadHandler(
    filename = function(){"updateddata.csv"},
    content = function(file) {
      write.csv(currentCleanedDf, file)
    }
  )
})
```

OUTPUT DEBUG CONSOLE TERMINAL

updated cleanedData()

③ conditions accordingly to the selected options

update cleanedData()

still not working yet : (

cleanedData(currentCleanedDf) → now we're updating cleanedData, while previously it stores the df w/ cleaned NA values, now it stores converted values of df.

Menu 2 : Dat Viz

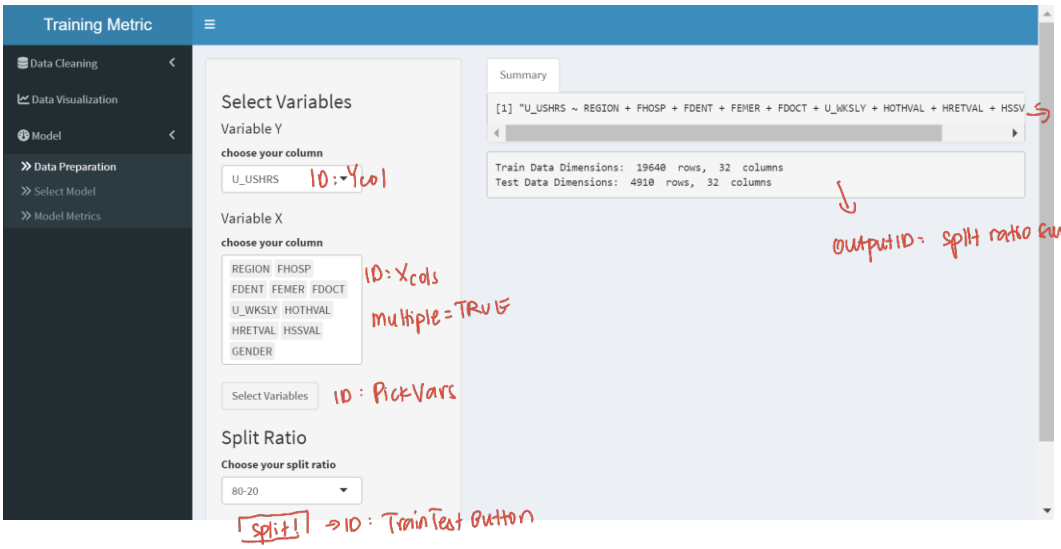
Menu 3 : Model

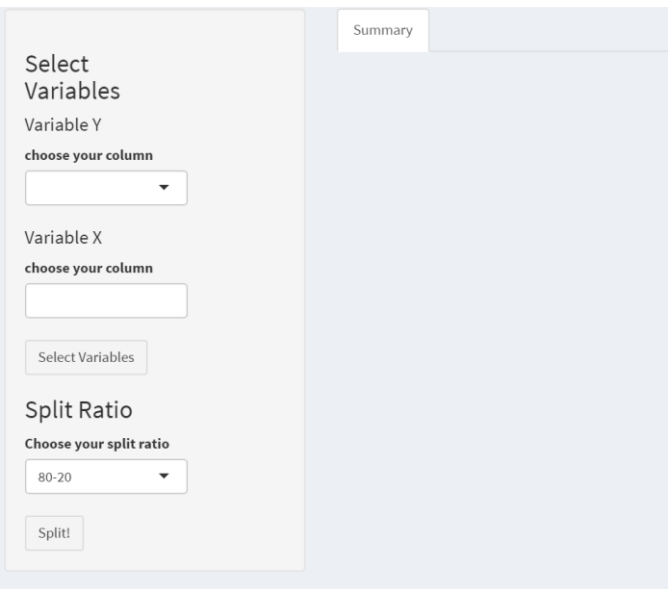
1. Tab : Data Preparation
2. Tab : Select Model
3. Tab : Model Metrics

03. Tab 1 : Data Preparation

UI

```
## 03. MODEL - TAB: DATA PREPARATION
tabItem(tabName = "modelpreparation",
        sidebarLayout(
          sidebarPanel(
            fluidRow(
              column(width = 8,
                     h3("Select Variables"),
                     h4("Variable Y"),
                     selectInput(inputId = "YCol", label = "Choose Your Y Variable", choices = NULL, selected = NULL),
                     h4("Variable X"),
                     selectInput(inputId = "XCols", label = "Choose your X Variables", choices = NULL, multiple = TRUE),
                     actionButton("PickVars", "Select Variables")
              ),
            ),
            fluidRow(
              column(width = 8,
                     h3("Split Ratio"),
                     selectInput(inputId = "TrainTestSplit", label = "Choose your split ratio", choices = c("70-30", "80-20", "90-10"), selected = "80-20"),
                     actionButton("TrainTestButton", "Split!")
              )
            )
          ),
          mainPanel(
            tabsetPanel(
              tabPanel("Summary", verbatimTextOutput("formulasummary"),verbatimTextOutput("splitratiosummary"))
            )
          )
        )
),
```





03. Tab 1 : Data Preparation

UI

```
## 03. MODEL - TAB: DATA PREPARATION
tabItem(tabName = "modelpreparation",
        sidebarLayout(
          sidebarPanel(
            fluidRow(
              column(width = 8,
                     h3("Select Variables"),
                     h4("Variable Y"),
                     selectInput(inputId = "YCol", label = "Choose Your Y Variable", choices = NULL, selected = NULL),
                     h4("Variable X"),
                     selectInput(inputId = "XCols", label = "Choose your X Variables", choices = NULL, multiple = TRUE),
                     actionButton("PickVars", "Select Variables")
              ),
            ),
            fluidRow(
              column(width = 8,
                     h3("Split Ratio"),
                     selectInput(inputId = "TrainTestSplit", label = "Choose your split ratio", choices = c("70-30", "80-20", "90-10"), selected = "80-20"),
                     actionButton("TrainTestButton", "Split!")
              )
            )
          ),
          mainPanel(
            tabsetPanel(
              tabPanel("Summary", verbatimTextOutput("formulasummary"),verbatimTextOutput("splitratiosummary"))
            )
          )
        )
),
```

SERVER

⚠ Reactive Val:

we need to carry / store: formula, train_data, test_data, label column_

Because we will use them as inputs for trainers_class objects

```
# 03. MODEL - TAB: DATA PREPARATION
# Choosing Y Var
observe({
  updateSelectInput(session, "YCol", label="choose your column", choices=names(cleanedData()), selected=names(cleanedData())[1])
})
# Choosing X Var
observe({
  updateSelectInput(session, "XCols", label="choose your column", choices=names(cleanedData()), selected=names(cleanedData())[1])
})

# input for training model classes
formula <- reactiveVal() # for the model
formula_asstring <- reactiveVal() # to show in UI
train_data <- reactiveVal()
test_data <- reactiveVal()
label_column <- reactiveVal()


# Making Formula
observeEvent(input$PickVars, {
  y_var <- input$YCol
  x_vars <- input$XCols
  # Create the formula using the selected variables
  formula_str <- paste(y_var, "~", paste(x_vars, collapse = " + "))
  # Store the formula
  formula_asstring((formula_str))
  formula((as.formula(formula_str)))
  # store variable Y
  label_column(y_var)
  output$formulasummary <- renderPrint({formula_str})
})

# Split Data
observeEvent(input$TrainTestButton, {
  split_ratio <- as.numeric(strsplit(input$TrainTestSplit, "-")[[1]]) / 100
```

how to form a formula

SERVER

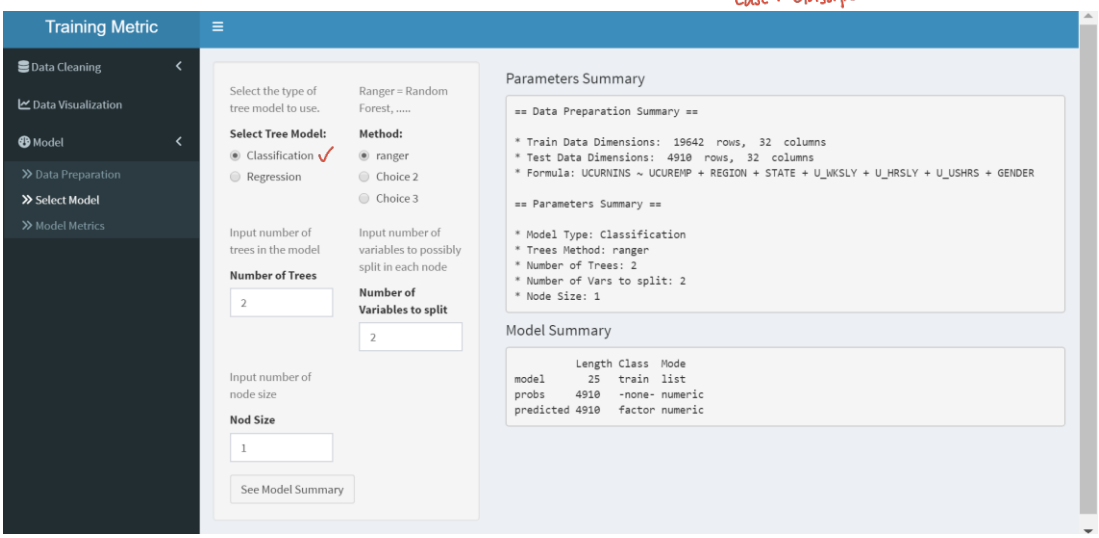
```
# Split Data
observeEvent(input$TrainTestButton, {
  split_ratio <- as.numeric(strsplit(input$TrainTestSplit, "-")[[1]]) / 100
  if (input$TrainTestSplit == "70-30") {
    train <- cleanedData()[1:round(split_ratio[1] * nrow(cleanedData())), ]
    test <- cleanedData()[(round(split_ratio[1] * nrow(cleanedData())) + 1):nrow(cleanedData()), ]
  } else if (input$TrainTestSplit == "80-20") {
    train <- cleanedData()[1:round(split_ratio[1] * nrow(cleanedData())), ]
    test <- cleanedData()[(round(split_ratio[1] * nrow(cleanedData())) + 1):nrow(cleanedData()), ]
  } else if (input$TrainTestSplit == "90-10") {
    train <- cleanedData()[1:round(split_ratio[1] * nrow(cleanedData())), ]
    test <- cleanedData()[(round(split_ratio[1] * nrow(cleanedData())) + 1):nrow(cleanedData()), ]
  }
  train_data(train)
  test_data(test)
  output$splitratiosummary <- renderPrint({
    cat("Train Data Dimensions: ", dim(train)[1], " rows, ", dim(train)[2], " columns\n")
    cat("Test Data Dimensions: ", dim(test)[1], " rows, ", dim(test)[2], " columns\n")
  })
})
```

03. Tab 2 : Select Model

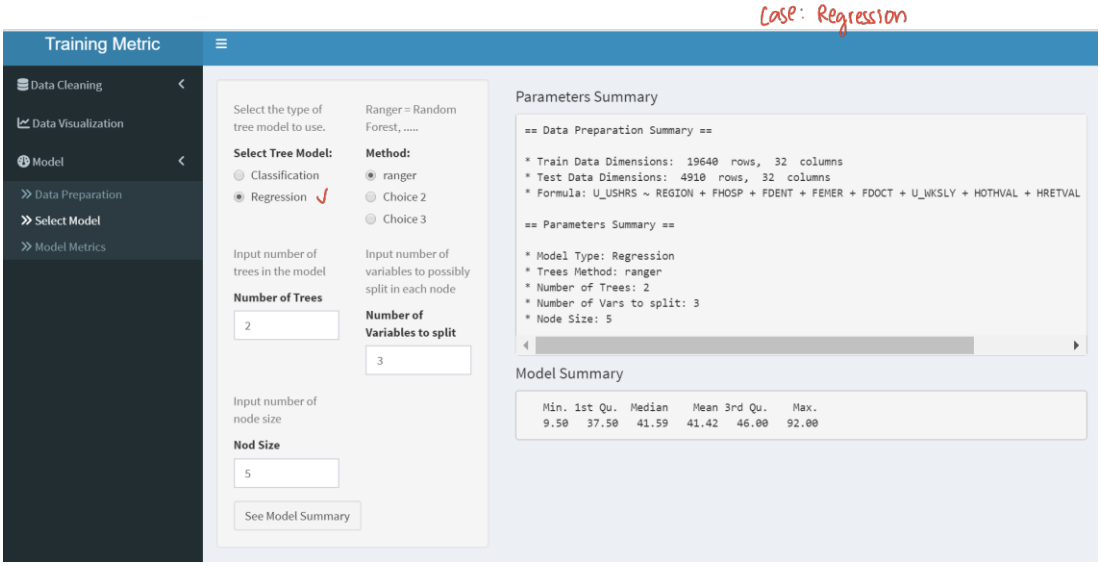
UI

```
## 03. MODEL - TAB: SELECT MODEL
tabItem(tabName = "selectmodel",
        sidebarLayout(
          sidebarPanel(
            fluidRow(
              column(width = 6,
                     helpText("Select the type of tree model to use."),
                     radioButtons("model_type", "Select Tree Model:",
                                  choices = c("Classification", "Regression"),
                                  selected = "")),
              column(width = 6,
                     helpText("Ranger = Random Forest, ....."),
                     radioButtons("trees_method", "Method:",
                                  choices = c("ranger", "Choice 2", "Choice 3"),
                                  selected = ""))),
            fluidRow(
              column(width = 6,
                     helpText("Input number of trees in the model"),
                     numericInput("numtrees","Number of Trees", value = 0)),
              column(width = 6,
                     helpText("Input number of variables to possibly split in each node"),
                     numericInput("m_try","Number of Variables to split", value = 0))),
            fluidRow(
              column(width = 6,
                     helpText("Input number of node size"),
                     numericInput("minnodesize","Nod Size", value = 0))),
            fluidRow(
              column(width = 12,
                     actionButton("seeModelSummary", "See Model Summary")))
          ),
          mainPanel(
            h4("Parameters Summary"),
            verbatimTextOutput("summaryParameters"),
            h4("Model Summary"),
            verbatimTextOutput("summaryModel")
          )
        )
)
```

Case : Classification



Case: Regression



03. Tab 2 : Select Model

UI

```
## 03. MODEL - TAB: SELECT MODEL
tabItem(tabName = "selectmodel",
        sidebarLayout(
          sidebarPanel(
            fluidRow(
              column(width = 6,
                     helpText("Select the type of tree model to use."),
                     radioButtons("model_type", "Select Tree Model:",
                                  choices = c("Classification", "Regression"),
                                  selected = "")),
              column(width = 6,
                     helpText("Ranger = Random Forest, ....."),
                     radioButtons("trees_method", "Method:",
                                  choices = c("ranger", "Choice 2", "Choice 3"),
                                  selected = ""))),
            fluidRow(
              column(width = 6,
                     helpText("Input number of trees in the model"),
                     numericInput("numtrees","Number of Trees", value = 0)),
              column(width = 6,
                     helpText("Input number of variables to possibly split in each node"),
                     numericInput("m_try","Number of Variables to split", value = 0))),
            fluidRow(
              column(width = 6,
                     helpText("Input number of node size"),
                     numericInput("minnodesize","Nod Size", value = 0))),
            fluidRow(
              column(width = 12,
                     actionButton("seeModelSummary", "See Model Summary")))
          ),
          mainPanel(
            h4("Parameters Summary"),
            verbatimTextOutput("summaryParameters"),
            h4("Model Summary"),
            verbatimTextOutput("summaryModel")
          )
        )
)
```

ReactiveVal

```
# input for training model classes
modelType <- reactiveVal()
method <- reactiveVal()
num_trees <- reactiveVal()
mtry <- reactiveVal()
min_node_size <- reactiveVal()

# variables for the classes
my_trainer <- reactiveVal()
result <- reactiveVal()
```

these are ReactiveVal for storing parameters that users will pick , and they will be carried for the inputs for modelling

Server : Parameters Summary

```
# 03. MODEL - TAB: SELECT MODEL
output$summaryParameters <- renderPrint({
  # from previous data Preps
  cat("== Data Preparation Summary == \n \n")
  cat("* Train Data Dimensions: ", dim(train_data())[1], " rows, ", dim(train_data())[2], " columns\n")
  cat("* Test Data Dimensions: ", dim(test_data())[1], " rows, ", dim(test_data())[2], " columns\n")
  cat("* Formula:", formula_asstring(),"\n \n")

  # parameters summary
  cat("== Parameters Summary == \n \n")
  model_type <- input$model_type
  modelType(model_type) -> update ReactiveVal
  cat("* Model Type:", model_type," \n")

  trees_method <- input$trees_method
  method(trees_method) -> update ReactiveVal
  cat("* Trees Method:", trees_method," \n")

  numtrees <- input$numtrees
  num_trees(numtrees) -> update ReactiveVal
  cat("* Number of Trees:", numtrees," \n")

  m_try <- input$m_try
  mtry(m_try) -> update ReactiveVal
  cat("* Number of Vars to split:", m_try," \n")

  minnodesize <- input$minnodesize
  min_node_size(minnodesize) -> update ReactiveVal
  cat("* Node Size:", minnodesize," \n")
})
```

reactiveVal that previously stores the splitted dataframe to train & test data

basically it just shows the selected parameters by users and all of those parameters are stored in reactiveVal

03. Tab 2 : Select Model

UI

```
## 03. MODEL - TAB: SELECT MODEL
tabItem(tabName = "selectmodel",
        sidebarLayout(
          sidebarPanel(
            fluidRow(
              column(width = 6,
                     helpText("Select the type of tree model to use."),
                     radioButtons("model_type", "Select Tree Model:",
                                  choices = c("Classification", "Regression"),
                                  selected = "")),
              column(width = 6,
                     helpText("Ranger = Random Forest, ....."),
                     radioButtons("trees_method", "Method:",
                                  choices = c("ranger", "Choice 2", "Choice 3"),
                                  selected = ""))),
            fluidRow(
              column(width = 6,
                     helpText("Input number of trees in the model"),
                     numericInput("numtrees","Number of Trees", value = 0)),
              column(width = 6,
                     helpText("Input number of variables to possibly split in each node"),
                     numericInput("m_try","Number of Variables to split", value = 0))),
            fluidRow(
              column(width = 6,
                     helpText("Input number of node size"),
                     numericInput("minnodesize","Nod Size", value = 0))),
            fluidRow(
              column(width = 12,
                     actionButton("seeModelSummary", "See Model Summary")))
          ),
          mainPanel(
            h4("Parameters Summary"),
            verbatimTextOutput("summaryParameters"),
            h4("Model Summary"),
            verbatimTextOutput("summaryModel")
          )
        )
```

these are reactiveVal to store the result from the modelling script (Trainers_classification / Trainers_regression)

Reactive Val

```
# input for training model classes
modelType <- reactiveVal()
method <- reactiveVal()
num_trees <- reactiveVal()
mtry <- reactiveVal()
min_node_size <- reactiveVal()

# variables for the classes
my_trainer <- reactiveVal()
result <- reactiveVal()
```

SERVER: MODEL SUMMARY

```
observeEvent(input$seeModelSummary, {
  my_trainer_shiny <- my_trainer()
  result_shiny <- result()

  if (input$model_type == "Classification"){
    my_trainer_shiny <- Trainer_classification$new(train_data(), test_data(), label_column())
    result_shiny <- my_trainer_shiny$create_random_forest(train_data(),test_data(),
                                                          formula = formula(),method = method(),
                                                          num_trees = num_trees(),importance = "impurity",
                                                          mtry = mtry(),min_node_size = min_node_size())
  } else {
    my_trainer_shiny <- Trainer_regression$new(train_data(), test_data(), label_column())
    result_shiny <- my_trainer_shiny$create_random_forest(train_data(),test_data(),
                                                          formula = formula(),method = method(),
                                                          num_trees = num_trees(),mtry = mtry(),
                                                          min_node_size = min_node_size())

  }
  my_trainer(my_trainer_shiny)
  result(result_shiny)
  output$summaryModel <- renderPrint({summary(result())})
})
```

if classification

regression

update ReactiveVal.

these are basically how the models are called from the classes/objects from trainers_classes.R and the input parameters for the classes are the ReactiveVal of formula(), method()... etc that already store users' input for parameters.

03. Tab 3 : Model Metrics

UI

```
## 03. MODEL - TAB: MODEL-METRICS
tabItem(tabName= "metrics",
        sidebarLayout(
          sidebarPanel(width = 3,
            actionButton("GenerateMetrics", "Generate Metrics")
          ),
          mainPanel(
            tabsetPanel(
              tabPanel("Metrics",id = "metrics_tab",
                       fluidRow( # summary of the metrics
                         column(width = 6,verbatimTextOutput("metricsummary1")),
                         column(width = 6,verbatimTextOutput("metricsummary2"))
                       ),
                       fluidRow( # PLOTS
                         column(width = 6,plotOutput("metricplot1")),
                         column(width = 6,plotOutput("metricplot2"))
                       ),
                       fluidRow(
                         column(width = 6,plotOutput("metricplot3")),
                         column(width = 6,plotOutput("metricplot4"))
                       ),
                       fluidRow(
                         column(width = 6,plotOutput("metricplot5")),
                         column(width = 6,plotOutput("metricplot6"))
                       )
              )
            ))
        ))
```

Classification

Regression

03. Tab 3 : Model Metrics

UI

```
## 03. MODEL - TAB: MODEL METRICS
tabItem(tabName= "metrics",
        sidebarLayout(
          sidebarPanel(width = 3,
            actionButton("GenerateMetrics", "Generate Metrics")
          ),
          mainPanel(
            tabsetPanel(
              tabPanel("Metrics",id = "metrics_tab",
                       fluidRow( # summary of the metrics
                         column(width = 6,verbatimTextOutput("metricsummary1")),
                         column(width = 6,verbatimTextOutput("metricsummary2"))
                       ),
                       fluidRow( # PLOTS
                         column(width = 6,plotOutput("metricplot1")),
                         column(width = 6,plotOutput("metricplot2"))
                       ),
                       fluidRow(
                         column(width = 6,plotOutput("metricplot3")),
                         column(width = 6,plotOutput("metricplot4"))
                       ),
                       fluidRow(
                         column(width = 6,plotOutput("metricplot5")),
                         column(width = 6,plotOutput("metricplot6"))
                       )
              )
            ))
        ))
))
```

Reactive Val

```
# input for training model classes
modelType <- reactiveVal()
method <- reactiveVal()
num_trees <- reactiveVal()
mtry <- reactiveVal()
min_node_size <- reactiveVal()

# variables for the classes
my_trainer <- reactiveVal()
result <- reactiveVal()
```

Server : classification

Calling all functions from trainer_classification

output metrics summary

output for plots

```
# 03. MODEL - TAB: MODEL METRICS
observeEvent(input$GenerateMetrics, {

  if (input$model_type == "Classification"){
    pred <- result()$predicted
    # metric specific [metricsummary1] & metric simple [metricsummary2]
    metrics_specific <- my_trainer()$calc_balanced_metrics(my_trainer()$labels,pred)
    metrics_simple <- my_trainer()$calc_metrics(my_trainer()$labels, pred)
    # plot metrics [metricplot 1-4] & learning curve [metricplot5]
    learning_curve <- my_trainer()$calculate_learning_curve(my_trainer()$train_data, my_trainer()$test_data, my_trainer()$label_column)
    learning_curve_plot <- my_trainer()$plot_metrics(learning_curve)
    plot_metrics <- my_trainer()$plot_metrics(learning_curve, train_sizes = seq(0.01, 0.9, by = 0.05))
    # plot probs [metricplot6]
    plot_prob <- my_trainer()$plot_probs(result()$probs,my_trainer()$label_column)

    # output metricsummary1 & metricsummary2
    output$metricsummary1 <- renderPrint({
      cat("Metrics Specific: \n")
      cat("* precision_pos: ", metrics_specific$precision_pos," \n")
      cat("* recall_pos: ", metrics_specific$recall_pos," \n")
      cat("* f1_pos: ", metrics_specific$f1_pos," \n")
      cat("* precision_neg: ", metrics_specific$precision_neg," \n")
      cat("* f1_neg: ", metrics_specific$f1_neg," \n")
      cat("* balanced_precision: ", metrics_specific$balanced_precision," \n")
      cat("* balanced_recall: ", metrics_specific$balanced_recall," \n")
      cat("* balanced_f1: ", metrics_specific$balanced_f1," \n")
    })
    output$metricsummary2 <- renderPrint({
      cat("Metrics Simple: \n")
      cat("* accuracy: ", metrics_simple$accuracy," \n")
      cat("* recall: ", metrics_simple$recall," \n")
      cat("* precision: ", metrics_simple$precision," \n")
      cat("* f1: ", metrics_simple$f1," \n")
    })
    # output plots
    output$metricplot1 <- renderPlot({plot_metrics$acc}) # plot metrics acc
    output$metricplot2 <- renderPlot({plot_metrics$prec}) # plot metrics prec
    output$metricplot3 <- renderPlot({plot_metrics$rec}) # plot metrics rec
    output$metricplot4 <- renderPlot({plot_metrics$f1}) # plot metrics f1
    output$metricplot5 <- renderPlot({learning_curve_plot}) # plot learning curve
    output$metricplot6 <- renderPlot({plot_prob}) # plot probs
```

OUTPUT DEBUG CONSOLE TERMINAL

03. Tab 3 : Model Metrics

UI

```
## 03. MODEL - TAB: MODEL-METRICS
tabItem(tabName= "metrics",
        sidebarLayout(
          sidebarPanel(width = 3,
            actionButton("GenerateMetrics", "Generate Metrics")
          ),
          mainPanel(
            tabsetPanel(
              tabPanel("Metrics",id = "metrics_tab",
                       fluidRow( # summary of the metrics
                         column(width = 6,verbatimTextOutput("metricsummary1")),
                         column(width = 6,verbatimTextOutput("metricsummary2"))
                       ),
                       fluidRow( # PLOTS
                         column(width = 6,plotOutput("metricplot1")),
                         column(width = 6,plotOutput("metricplot2"))
                       ),
                       fluidRow(
                         column(width = 6,plotOutput("metricplot3")),
                         column(width = 6,plotOutput("metricplot4"))
                       ),
                       fluidRow(
                         column(width = 6,plotOutput("metricplot5")),
                         column(width = 6,plotOutput("metricplot6"))
                       )
              )
            ))
        ))
))
```

Reactive Val

```
# input for training model classes
modelType <- reactiveVal()
method <- reactiveVal()
num_trees <- reactiveVal()
mtry <- reactiveVal()
min_node_size <- reactiveVal()

# variables for the classes
my_trainer <- reactiveVal()
result <- reactiveVal()
```

Server : Regression

calling all funcs from trainer_regression

output metrics summary

output renderplot

```
} else {
  # metrics calculation [metricsummary 1 & 2]
  metrics_calculation <- my_trainer()$calc_metrics(my_trainer()$model)

  # plot metrics [metric plot 1 - 4]
  metrics <- my_trainer()$calculate_learning_curve(my_trainer()$train_data, my_trainer$test_data)
  plot_metrics <- my_trainer()$plot_metrics(metrics)

  # plot actual vs predicted [metric plot 5 & 6]
  plots_ap <- my_trainer()$actual_vs_predicted(my_trainer()$predictions)
  ap <- plots_ap$actual_predicted
  residuals <- plots_ap$resiudals_plot

  # output metricsummary1 & metricsummary2
  output$metricsummary1 <- renderPrint({
    cat("Metrics Calculation - Train: \n")
    cat("* train_mse: ", metrics_calculation$train_mse," \n")
    cat("* train_rmse: ", metrics_calculation$train_rmse," \n")
    cat("* train_mae: ", metrics_calculation$train_mae," \n")
    cat("* train_rsquared: ", metrics_calculation$train_rsquared," \n")
  })
  output$metricsummary2 <- renderPrint({
    cat("Metrics Calculation - Test: \n")
    cat("* test_mse: ", metrics_calculation$test_mse," \n")
    cat("* test_rmse: ", metrics_calculation$test_rmse," \n")
    cat("* test_mae: ", metrics_calculation$test_mae," \n")
    cat("* test_rsquared: ", metrics_calculation$test_rsquared," \n")
  })
  # output plots
  output$metricplot1 <- renderPlot({plot_metrics$p_mse}) # plot metrics acc
  output$metricplot2 <- renderPlot({plot_metrics$p_mae}) # plot metrics prec
  output$metricplot3 <- renderPlot({plot_metrics$p_rmse}) # plot metrics rec
  output$metricplot4 <- renderPlot({plot_metrics$p_rsquared}) # plot metrics f1
  output$metricplot5 <- renderPlot({ap}) # plot actual-predicted
  output$metricplot6 <- renderPlot({residuals}) # plot residuals
}
```

NOTES : REACTIVEVAL

The main reason why we need ReactiveVal() for storing / assigning variables is because we are not able to call variable outside the "observe" / "observeEvent" function, as it may give us an error

- cleanedData()
  - (1) "Upload Here" menu : data() → store original df from users
    - → then : cleanedData(data) → store it in reactiveVal
  - (2) "Cleaning NA" menu :
    - → in observeEvent : "data_cleaning" → store a cleaned NA df
    - → then : cleanedData(data_cleaning) → store an updated / cleaned NA df
  - (3) "Convert Values" menu :
    - → in observeEvent : "currentCleanedDf" → store a converted values df
    - → then : cleanedData("currentCleanedDf" → store an updated df

Thus, either user convert df first or clean NA first or even skip all the cleaning data preparations, cleanedData() always carries the updated Df

ReactiveVal for Parameters

(1) formula()
(2) train_data()
(3) test_data()
(4) label_column()
(5) model_type()
(6) method()
(7) num_trees()
(8) mtry()
(9) min_node_size()

ReactiveVal for Class & Objects

(1) my_trainer()
(2) result()

How to Test on medical-csv

CLASSIFICATION :

factor

→ FORMULA : UCURNIS ~ UCUREMP + REGION + STATE + U_WKSLY + U_HRSL + U_USHRS + GENDER → factor

- → Datasplit : 80-20
- → Tree Model : classification
- → method : ranger
- → num of trees : 2
- → num of vars to split : 2
- → nod size : 1

Convert values as the yellow highlighted

REGRESSION :

factor

(2) → FORMULA : U_USHRS ~ REGION + FHOSP + FDENT + FEMER + FDOCT + U_WKSLY + HOTHVAL + HRETVAL + HSSVAL + GENDER

HOTHVAL → num, HRETVAL → num, HSSVAL → num, GENDER → factor

- → Datasplit : 80-20
- → Tree Model : Regression
- → method : ranger
- → num of trees : 2
- → num of vars to be split : 3
- → nod size : 5

(1) Convert values

(2) Clear NAs → HOTHVAL & HSSVAL