Florida Polytechnic University, *Fall 2023 Semester*

Course: Scientific Computation & Programming (COP 5090)

Professor: Rei Sanchez-Arias, Ph.D.

Scientific Computation & Programming

Final Project: *Shiny Web Application "German Predictor"*

*(**AI generated picture** with the prompt „Oilpainting of how a person from the 18th century would have imagined a mashine that learns by itself and does predictions of the future.", Source: https://openai.com/dall-e-2/)*

Submitted by:

Alexander Burkhart

E-Mail: aburkhart6899@floridapoly.edu

Jan Hery

E-Mail: jhery6811@floridapoly.edu

Submitted on:

12/03/2023

Table of Contents

1. Introduction and Motivation

1.1 Background and Motivation

In recent years, the field of data science has seen a significant increase in popularity due to the growing availability of data and the increasing recognition of its potential to extract valuable insights. In the midst of this data revolution, predictive modeling emerges as a crucial technique, enabling analysts and data scientists to predict future trends, make informed decisions, and gain a deeper understanding of complex systems. Acknowledging the importance of predictive modeling, there is a pressing need to bridge the gap between its complexity and accessibility for newcomers to the field.

Traditional approaches to teaching predictive modeling often involve complex programming languages and statistical software, creating significant barriers for individuals without a coding background. To tackle this challenge, our Shiny application serves as a solution, providing an intuitive and interactive platform that illustrates the fundamental concepts of predictive modeling. By offering a user-friendly interface, the application aims to empower students, educators, and aspiring data enthusiasts to explore, visualize, and understand the essence of predictive modeling without the need for extensive coding knowledge.

Our core motivation revolves around making data science education accessible to a broader audience. In a world where data-driven decision-making is widespread across various industries, it is crucial to equip individuals with the skills needed to navigate the complexities of predictive modeling.

1.2 Goal of Shiny Application

The first primary goal of our Shiny application is to provide users with a Exploratory Data Analysis (EDA) functionality. EDA serves as the foundation for any data-driven endeavor, allowing users to comprehend the characteristics of the selected datasets. By implementing this feature, our aim is to enable users to identify patterns, and gain valuable insights into the structure of the chosen datasets.

The second key goal of our Shiny application is to implement Linear Regression, a fundamental predictive modeling technique. Linear Regression provides a straightforward yet powerful approach to modeling the relationship between dependent and independent variables. By incorporating Linear Regression into the application, our objective is to explain this technique for users without coding experience, allowing them to grasp the core principles and application of linear modeling.

Moving forward, our third primary goal revolves around implementing K-Nearest Neighbors (KNN), introducing users to a non-linear approach in predictive modeling. The KNN algorithm relies on the proximity of data points to make predictions. By incorporating KNN into the application, we aim to highlight the flexibility and effectiveness of non-linear modeling.

Our overall goal is to develop an easy-to-use Shiny app using R that will allow users to explore these functionalities.

2. Literature Review

2.1 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a crucial step in the data analysis process that involves looking into and summarizing the main characteristics of a dataset. It serves as a fundamental phase to gain insights into the structure, patterns, and potential outliers within the data, laying the groundwork for informed decision-making in various fields.

Understanding the details of data and its variables is essential as it allows analysts and data scientists to make meaningful interpretations and predictions. EDA provides a comprehensive overview of the dataset's distribution, central tendencies, and spread. This insight increasing the robustness of subsequent analyses. Additionally, by examining the relationships between variables, analysts can uncover patterns and dependencies that form the basis for more advanced modeling techniques.

A correlation heatmap, a vital part of EDA, visually represents the strength and direction of relationships between different features in a dataset. This heatmap helps identify multicollinearity - instances where two or more variables are highly correlated - which can impact the accuracy of predictive models. Detecting and understanding these relationships are crucial for refining model inputs and preventing redundancy in the feature space.

Histograms for each feature are providing a detailed view of the data's frequency distribution. This visualization is helping in understanding the shape, central tendency, and dispersion of variables. It aids in recognizing the presence of outliers and understanding the data's symmetry.

In summary, EDA is essential for unlocking the potential of a dataset. By exploring the relationships between variables, identifying correlations, and understanding feature distributions. EDA acts as the compass that guides analysts through the data. For this reason, EDA will be one of the fundamentals for the Shiny application.

2.2 Predictive Modeling

2.2.1 Linear Regression

Linear Regression is a foundational and widely used statistical method within machine learning and statistics, serving as a fundamental building block for various predictive modeling tasks. This method aims to establish a relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data. It's a technique applicable in diverse fields, from economics and finance to social sciences and beyond.

At its core, Linear Regression tries to model the relationship between variables by assuming a linear association, where changes in the independent variables lead to proportional changes in the dependent variable. The linear equation typically takes the form of y=mx+c, where 'y' represents the dependent variable, 'x' denotes the independent variable, 'm' signifies the slope of the line, and 'c' represents the intercept on the y-axis. The primary objective lies in determining the best-fitting line that minimizes the difference between the predicted and observed values.

One of the key strengths of Linear Regression is its interpretability. The model's simplicity allows for clear comprehension and explanation of the relationship between variables. It provides valuable insights into the impact each independent variable on the dependent variable.

However, the effectiveness of Linear Regression relies on certain assumptions. These include the linearity between variables, independence of observations, absence of multicollinearity among independent variables.

The performance of Linear Regression can be further enhanced by employing techniques such as regularization (e.g., Ridge Regression, Lasso Regression) to mitigate overfitting, feature engineering to select relevant variables, and assessing model performance through metrics like R-squared, Mean Squared Error (MSE), or Root Mean Squared Error (RMSE).

Linear Regression remains a fundamental and interpretable method in predictive modeling, as well as offering valuable insights into relationships between variables.

2.2.2 K-Nearest Neighbor

K-Nearest Neighbors (KNN) is an algorithm known for its simplicity and effectiveness across various machine learning tasks, spanning classifications to regression tasks. Its fundamental principle revolves around predicting outcomes for new data points based on the consensus of the majority class or the average of the k nearest neighbors in the feature space. The value "k" in KNN describes the count of neighbors taken into consideration when deciding, serving as an important parameter in this algorithm. KNN operates as a non-linear algorithm, contributing to its versatility and applicability in different scenarios.

The intelligent selection of the "k" parameter is of high importance in KNN. A small value for "k" might succumb to overfitting, overemphasizing noise present within the dataset. A larger "k" could lead to oversimplification, potentially overlooking intricate patterns within the data. Finding the right balance is crucial for optimal performance, ensuring the algorithm captures essential facts.

To determine the most suitable "k" value, using cross-validation technique is an important tool. In this process, the dataset undergoes division into for example 10 parts or folds. Subsequently, the Root Mean Squared Error (RMSE) for each fold and various "k" values is meticulously calculated. These results are then systematically compared, with the goal to identify the most optimal "k". Post-evaluation, the algorithm does the training, using the determined optimal "k".

3. Methodology

3.1 Development Approach

Our approach to crafting the Shiny app was a collaborative endeavor that involved backend development and modular structure-building, resulting in a cohesive and user-friendly UI. The process was a joint effort between Alexander, who focused on the backend, and Jan, who concentrated on structuring modules and the UI.

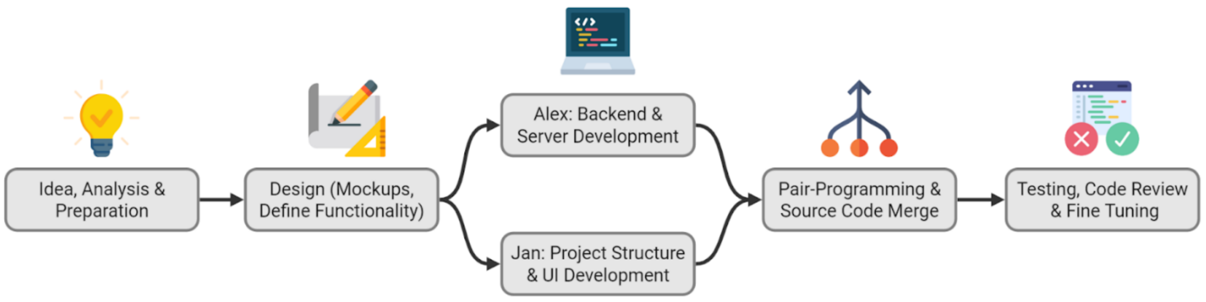

Idea, Analysis & Preparation
Design (Mockups, Define Functionality)
Alex: Backend & Server Development
Jan: Project Structure & UI Development
Pair-Programming & Source Code Merge
Testing, Code Review & Fine Tuning

Figure 1: Workflow Diagram Development Process

Alexander dedicated his efforts to the backend infrastructure and functionality. He crafted the core features, establishing the logic that powered the app's responsiveness and managed its data. His focus was on data handling, computations, and setting up elements to ensure seamless app interaction.

Meanwhile, Jan took the lead in building a modular structure for the app, commencing with the project structure and UI components. His goal was to create an inviting and easy-to-use interface. Jan utilized his expertise in UI/UX design to shape the layout, organizing the elements within the app for user-friendliness.

Our approach was grounded in careful planning and teamwork. We initiated the process by outlining mock-ups, defining functionalities, and specifying the expected app behavior. Both of us independently developed functionalities and UI components, adhering to the predetermined design and functions. In collaborative coding sessions the merging of backend functionality with the UI structure took place. Collaborative review and integration were crucial to combine the components and ensure the app operated as envisioned. An additional layer of our process involved acceptance testing of each other's implemented functionalities. For instance, Jan's code underwent review by Alexander, and vice versa. This practice of mutual assessment and feedback enhanced the reliability and quality of our individual contributions.

3.2 Shiny Framework

Shiny is a web application framework developed in the R programming language that simplifies the creation of interactive and dynamic web applications. It is particularly well-suited for data scientists, statisticians, and analysts who want to share their R-based analyses and visualizations with a broader audience. Shiny enables users to transform their R code into interactive, user-friendly web applications without requiring extensive web development skills.

At its core, Shiny consists of two main components: the user interface (UI) and the server. The UI defines the layout and appearance of the application, specifying the visual elements such as buttons, sliders, and plots. The server, on the other hand, handles the underlying computational logic, responding to user inputs and dynamically updating the UI accordingly.

Shiny supports a wide range of interactive elements, including interactive plots, tables, and custom HTML widgets, providing flexibility in creating diverse and engaging applications. Moreover, it facilitates the integration of R's powerful statistical and data analysis capabilities into the interactive web environment.

3.3 Datasets

3.3.1 Mtcars

The "mtcars" dataset in R is a built-in dataset containing data on various attributes of 32 different automobile models from a 1974 Motor Trend magazine. Each row in the dataset represents a different car model, and there are 11 columns providing information on different features and characteristics of these cars.

- **mpg**: Miles per gallon (fuel efficiency of the car)
- **cyl**: Number of cylinders in the engine (4, 6, or 8)
- **disp**: Displacement, which refers to the engine's total volume in cubic inches
- **hp**: Horsepower, a measure of the engine's power
- **drat**: Rear axle ratio, which impacts the car's acceleration
- **wt**: Weight of the car in thousands of pounds
- **qsec**: Quarter-mile time, a measure of the car's acceleration capability
- **vs**: Engine type (0 = V-shaped, 1 = straight)
- **am**: Transmission type (0 = automatic, 1 = manual).10. gear: Number of forward gears
- **carb**: Number of carburetors

3.3.2 Swiss

The "swiss" dataset is one of the built-in datasets available in R, providing information about socio-economic indicators and fertility rates in Switzerland's provinces in the late 19th century. This dataset, often used for educational and demonstration purposes, contains data points for 47 provinces in Switzerland during the years 1888-1889. It includes the following variables.

- **Fertility**: The number of live births per 1,000 inhabitants over the age of 15.
- **Agriculture**: Percentage of the province's population involved in agricultural pursuits.
- **Examination**: Percentage of the military draftees classified as "highest" level in the examination.
- **Education**: Percentage of the population considered as having completed primary education.
- **Catholic**: Percentage of the Catholic population.
- **Infant.Mortality**: Rate of infant mortality per 1,000 live births.

These variables provide insight into the socio-economic characteristics and demographic profiles of the Swiss provinces at that time.

3.3.3 Air Quality

In R, the "airquality" dataset is a built-in dataset that provides air quality measurements for New York City between May to September 1973. The dataset contains daily readings of air pollution indicators along with meteorological measurements. With its combination of air quality indicators and meteorological data, the "airquality" dataset serves as an illustrative tool for demonstrating the impact of environmental factors on air pollution and exploring relationships between different variables. The dataset consists of 154 observations on 6 variables:

- **Ozone**: The concentration of ozone in parts per billion (ppb).
- **Solar.R**: Solar radiation in Langleys (a measure of solar energy received per square meter).
- **Wind**: Wind speed in miles per hour.
- **Temp**: Temperature in degrees Fahrenheit.
- **Month**: The month of the measurement (ranging from 5 to 9 for May to September).
- **Day**: The day of the measurement within the respective month.

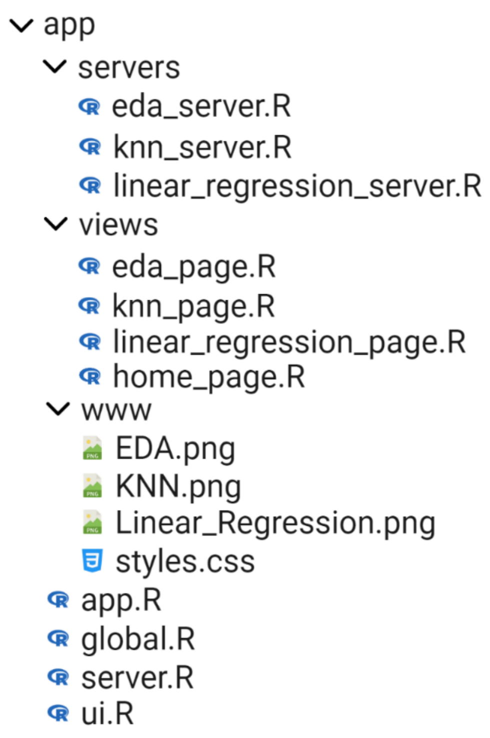
4. Technical Implementation

4.1 Project Structure

In the course of this chapter, we will have a look into the individual files and directories created as part of this project. To provide a comprehensive overview of all these components, we will first examine the project structure. In general, a modular approach has been chosen, aiming to externalize individual functions whenever possible to keep the code organized and the individual files as compact as possible. This not only simplifies error detection but also allows for easy expansion of the application with additional files and functionalities without significant integration effort.

The project adopts a modular approach to enhance clarity and maintainability. This involves breaking down the application into smaller, self-contained units, each handling a specific aspect or functionality. Objectives of this idea are:

- **Code Organization**: By dividing functions into separate modules, the codebase becomes more readable and easier to navigate. This facilitates quicker identification and resolution of errors.
- **File Size Management**: Dividing the application into smaller files prevents any single file from becoming overly large. This not only aids in maintaining a clean and concise codebase but also promotes a more granular understanding of each module's purpose.
- **Scalability**: The modular structure enhances the scalability of the application. New functionalities can be seamlessly added by introducing new modules without disrupting the existing codebase. This flexibility is particularly valuable in dynamic projects where requirements may evolve over time.



```
app
  servers
    eda_server.R
    knn_server.R
    linear_regression_server.R
  views
    eda_page.R
    knn_page.R
    linear_regression_page.R
    home_page.R
  www
    EDA.png
    KNN.png
    Linear_Regression.png
    styles.css
  app.R
  global.R
  server.R
  ui.R
```

Figure 2: General Project Structure

The project is thoughtfully organized with a clear focus on simplicity and efficiency. Within the app/ directory, different subdirectories and files are arranged to enhance modularity, readability, and overall project maintainability.

servers/ : This directory houses distinct R scripts for each predictive modeling technique. Specifically, there are eda_server.R, knn_server.R, and linear_regression_server.R. Each of these scripts manages the server-side logic for their respective predictive modeling techniques.

views/ : The views/ directory contains R scripts representing different pages of the application. In addition to the pages dedicated to specific modeling techniques (eda_page.R, knn_page.R, and linear_regression_page.R), there is a home_page.R file, which serves as the landing page and a central hub for the users to enter the and navigate through the application.

www/ : Resources contributing to the visual and functional aspects of the application are stored here. This includes images for the different predictive modeling algorithms and the styles.css file, which inherits the global visual styling of the application.

app.R: Serving as the main entry point, the app.R file orchestrates the Shiny application by integrating the server and UI components. This file plays a crucial role in defining the overall behavior of the application.

global.R: The global.R file is dedicated to housing global variables and functions that remain accessible throughout the application. This enhances code reusability and maintains a clean separation. In our case it is used to define the libraries used within the project.

server.R and ui.R: These files, server.R and ui.R, respectively handle the server-side logic and user interface components. Together, they form the backbone of the Shiny application, ensuring the seamless connection between user interactions and server responses.

This structured approach to organizing files promotes ease of development and paves the way for the straightforward addition of new functionalities. Each predictive modeling technique is encapsulated within its own set of files, fostering clarity and enabling collaborative development. The home_page.R file provides users with a central point for navigation or introductory information within the application.

4.2 File Descriptions

After the project overview has been presented, the following will focus on highlighting code snippets from individual files and explaining how predictions work from both the UI and server perspectives. Firstly, the focus will be on the server perspective, followed by presenting the program code of the UI and the user perspective.

4.2.1 Servers

The architecture contains three server files. One for EDA, one for the KNN algorithm and a last one for the Linear Regression. The EDA is structured in two parts - Input and Output. The code for the KNN and Linear Regression servers are structured in three parts. The first part is the input part, here are the UI elements initialized. The main process is taking part in the prediction part. This part of the code creates the prediction models and passes the predictions and all the necessary information to the output part. The output part handles the creation of plots and summaries for the user.

/servers/eda_server.R:

Inputs:

```
 selected_df <- reactive({
    dataframes_list[[input$dataframe]]
  })
```

A reactive expression is a function, which fetches the selected dataframe from dataframes_list_based. Reactive expressions are updated when the user input changes.

Outputs:

```
# Create correlation heatmap
  output$correlation_heatmap <- renderPlot({
    df <- selected_df()

    # Exclude non-numeric columns
    numeric_cols <- sapply(df, is.numeric)
    df_numeric <- df[, numeric_cols]

    # Check if there are at least two numeric columns and enough non-missing
rows for correlation
    if (sum(numeric_cols) >= 2 && sum(complete.cases(df_numeric)) >= 2) {
      correlations <- cor(df_numeric, use = "complete.obs")
      corrplot(correlations, method = "color", type = "upper", order =
"hclust", tl.col = "black", tl.srt = 45)
    } else {
      plot(NULL, xlim = c(0, 1), ylim = c(0, 1), main = "Correlation Heatmap",
xlab = "", ylab = "")
      text(0.5, 0.5, "Not enough numeric columns or rows for correlation", cex
= 1.2)
    }
  })
```

First a subset of the selected data is created which only contains numeric columns. Afterwards the columns are checked that there are at least two numeric ones and that there are enough rows for a correlations analysis. If these conditions are met, the correlations are calculated between the numeric columns and a correlation heatmap is created. If the conditions are not met, an empty plot with a message stating, "Not enough numeric columns or rows for correlation is created."

```
  # Display histogram
  output$histogram <- renderPlot({
    df <- selected_df()
    numeric_var <- input$numeric_var

    # Check if a numerical variable is selected and the data is non-NULL
    if (!is.null(df[[numeric_var]]) && is.numeric(df[[numeric_var]])) {
      graphics::hist(df[[numeric_var]], main = paste("Histogram of",
numeric_var), xlab = numeric_var, col = "skyblue", border = "black")
    } else {
      # If the data is NULL or not numeric, display a message or a default
plot
      plot(NULL, xlim = c(0, 1), ylim = c(0, 1), main = "Histogram", xlab =
"", ylab = "")
      text(0.5, 0.5, "Invalid data for histogram", cex = 1.2)
    }
  })
```

The output of this code is a histogram. First the data frame is received and the variable which is to show. Then the selected variable is checked for numeric and not NULL. If these conditions are met, a histogram is created. If the selected variable is NULL or not numeric, it produces an empty plot with the title "Histogram" and the message "Invalid data for histogram."

/servers/linear_regression_server.R:

Inputs:

```
  # Render dependent variable UI
  output$dependent_var_ui_2 <- renderUI({
    df <- selected_df()
    selectInput("dependent_var", "Choose the Dependent Variable", choices =
colnames(df))
  })
```

The first rendering block creates a dynamic dropdown menu. This dropdown menu allows users to select the dependent variable from the columns of the currently selected data frame.

```
  # Render independent variables UI
  output$independent_vars_ui_2 <- renderUI({
    df <- selected_df()
    checkboxGroupInput("independent_vars", "Choose the Independent Variables",
choices = colnames(df))
  })
```

The second rendering block generates a checkbox group. This checkbox group enables users to select multiple independent variables from the columns of the currently selected data frame.

Processing:

In this part of the documentation the processing part of the shiny app is described. The overall structure is explained with linear regression. A lot of code from linear regression is recycled in the KNN server.

```
output_data <- reactive({

    req(input$dependent_var, input$independent_vars)
    df <- selected_df()
    dep_var <- input$dependent_var
    ind_vars <- input$independent_vars
    vars_length <- length(ind_vars)

    # Logic to catch error if dependent and independent variables are the same
    if(sum(dep_var == ind_vars) > 0) {
      equal_error = TRUE
    } else {
      equal_error = FALSE
    }
```

The output_data reactive expression is the core computation of the prediction part. It is designed to perform several computations and data manipulations based on user-selected dependent and independent variables for linear regression analysis and visualization.

At the beginning, the code validates user inputs, ensuring that both a dependent variable and at least one independent variable are selected.

```
if (vars_length > 0 && !equal_error) {
      # Filter out any missing values in the selected variables
      filtered_df <- df[complete.cases(df[, c(dep_var, ind_vars)]), ]
      train_ind <- createDataPartition(filtered_df[,1], p = .75, list = FALSE)
      training <- filtered_df[train_ind, ]
      testing <- filtered_df[-train_ind, ]

      # Create formula for linear regression
      formula <- as.formula(paste(dep_var, "~", paste(ind_vars, collapse = " +
")))

      # Perform linear regression
      lm_model <- lm(formula, data = training)
      predicitons <- predict(lm_model, newdata = testing)
      column_name <- dep_var
      vec <- c(1:3)
      my_examples_pred <- predicitons[vec]
      my_examples_true <- testing[vec, column_name]
      MAE <- predicitons - testing[, column_name]
     }
```

Once the input validation and error handling are completed, the reactive expression moves on to process the data and build a linear regression model. This includes steps like filtering out missing values, splitting the data into training and testing sets, constructing a regression formula, performing the regression, generating predictions, and computing the Mean Absolute Error (MAE) between predicted and actual values.

```
      list(
        summary = capture.output(summary(lm_model)),
        plot_data = list(
          x = training[, ind_vars, drop = FALSE],
          y = training[, dep_var],
          lm_model = lm_model
        ),
        my_examples_pred = my_examples_pred,
        my_examples_true = my_examples_true,
        vars_length = vars_length,
        MAE_lin = (mean(abs(testing[, column_name] - predicitons))),
        indep_variables = ind_vars,
        depend_variable = dep_var,
        equal_error = equal_error
      )
      # if no independent variable is chosen
    } else {
      list(
        summary = "Choose exactly one independent variable for plotting.",
        plot_data = NULL,
        vars_length = vars_length,
        equal_error = equal_error)
```

The output from this reactive expression is a comprehensive list containing various elements:

- Summary output from the linear regression model.
- Data suitable for plotting, such as independent variables (x) and the dependent variable (y) for the training set, along with the lm_model.
- Examples of predicted and true values for a subset of the dataset.
- Length of selected independent variables and their names.
- The name of the dependent variable.
- An equal_error flag indicating whether there was an error due to duplicate selection of dependent and independent variables.

Overall, this block undertakes a series of operations, managing errors and generating essential data and model outputs for linear regression analysis and visualization within the Shiny application.

/servers/knn_server.R/

```
    # Logic to catch error if there is no independent variable
    if (vars_length > 0 && !equal_error) {
      # Filter out any missing values in the selected variables
      filtered_df <- df[complete.cases(df[, c(dep_var, ind_vars)]), ]
      train_ind <- createDataPartition(filtered_df[, 1], p = 0.75, list =
FALSE)
      training <- filtered_df[train_ind, ]
      testing <- filtered_df[-train_ind, ]

      # Create formula for KNN
      formula <- as.formula(paste(dep_var, "~", paste(ind_vars, collapse = " +
")))
```

This part is the data processing and model construction for KNN regression. It filters out missing values from the chosen variables, creating a new dataset filtered_df. The dataset is then split into training and testing subsets. Afterwards a KNN regression formula is created based on the user-selected dependent and independent variables.

```
# Set up control parameters for 10-fold cross-validation
      ctrl <- trainControl(method = "cv", number = 10)
      k_values <- 1:20
      knn_model_cv <- train(formula, data = training, method = "knn",
trControl = ctrl, preProcess = c("center", "scale"), tuneGrid = data.frame(k =
k_values))

      predicitons <- predict(knn_model_cv, testing)
      column_name <- dep_var
      MAE_KNN <- (mean(abs(testing[, column_name] - predicitons)))
      vec <- c(1:3)
      my_examples_pred <- predicitons[vec]
      my_examples_true <- testing[vec, column_name]
      }
```

To perform a 10-fold cross-validation, the code configures control parameters to employ trainControl and uses a range of k values (from 1 to 20) for KNN analysis.

Predictions are generated using the KNN model on the testing data. The Mean Absolute Error (MAE) is calculated between predicted and actual values. Additionally, a subset comprising predicted and true values are collected for a few examples within the dataset.

Output:

In the output part the code prepares the information of the prediction part to hand it over to the UI. A summary of the model, a scatterplot with the predictions of the model and the MAE is handed over to the UI. A comparison of True Value vs predicted value is as well prepared.

```
# Display summary for KNN
  output$summary_knn <- renderPrint({
    equal_error = output_data_knn()$equal_error
    if (equal_error == TRUE) {
      print("No Summary available")
    } else {
      print(output_data_knn()$knn_model_cv)
    }
  })
```

This section defines the output element labeled summary_knn. The purpose of this output element is to present a summary related to the KNN regression model.

```
  # Display MAE for KNN
  output$MAE_knn <- renderText({
    equal_error = output_data_knn()$equal_error
    if (equal_error == TRUE) {
      "No MAE calculated"
    } else {
      mae_value <- round(output_data_knn()$MAE_KNN, 4)
      formatted_text <- paste("<span style='font-weight: bold; font-size: 
16px;'>Mean Absolute Error for KNN is: ", mae_value, "</span>")
      formatted_text
    }
  })
```

Here the output element summary_knn is created. This output element presents the calculated MAE related to the KNN regression model.

Error Handling:

```
    # Logic to catch error if dependent and independent variables are the same
    if (sum(dep_var == ind_vars) > 0) {
      equal_error = TRUE
    } else {
      equal_error = FALSE
    }
```

This code is checking if the dependent variable matches any of the independent variables. If an error is detected in the input, the flag (equal_error) is set to handle these cases. This flag is used in the processing part to not try to create predictions and in the output part of the code to not try to show a summary. In case of errors in the input (like not choosing any independent variables), a different list is returned from the reactive output_data element containing an error message and necessary elements set to NULL.

```
 if (vars_length > 0 && !equal_error)
{
       ….

}
```

This error handling is inside the prediction part of the code. It makes sure that the user selects at least one independent variable and that the equal error is not TRUE.

```
    if (vars_length > 0 && !equal_error)
      {
      plot(knn_model)
      }
    else if (equal_error == TRUE) {
            plot(NULL, xlim = c(0, 1), ylim = c(0, 1), xlab = "Independent
      Variable", ylab = "Dependent Variable", main = "Scatter plot with
      Regression Line")
            text(0.5, 0.5, "No plot generated. Dependent and Independent
      Variables are the same", cex = 1.2)
          }
    else {
            plot(NULL, xlim = c(0, 1), ylim = c(0, 1), xlab = "Independent
      Variable", ylab = "Dependent Variable", main = "Scatter plot with KNN
      Regression")
            text(0.5, 0.5, "No plot generated. Either too many dimensions or
      KNN model is NULL", cex = 1.2)
    }
  })
```

This error handling is the continuation of the equal_error handling inthe output part of the app. To ensure that nothing is shown to the user when the equal_error is true or no independent variable is chosen.

4.2.2 User Interface

ui.R

The most important file in this regard is the ui.R, which represents the main structure of how our UI should look like. Here the frame of the application is defined oriented on the design of a typical admin dashboard with a header, sidebar and a body - This will be explained in the following in more detail.

```r
# Source the UI page files
source("views/home_page.R")

# Create dashboard with shinyDashboard
dashboardPage(
  skin = "purple",
  dashboardHeader(
    title = "German Predictor"
  ),
  dashboardSidebar(
    sidebarMenu(
      menuItem(" Home", tabName = "home", icon = icon("home")),
      menuItem(" Exploratory Data Analysis", ...
    )
  ),
  dashboardBody(
    # Include custom CSS styles
    includeCSS("www/styles.css"),

    tabItems(
      # Home tab
      tabItem(tabName = "home", homePageUI()),

      # EDA tab
      tabItem(tabName = "eda", source("views/eda_page.R")[1]),
      ...
    )
  )
)
```

The code begins by sourcing the UI elements from the home_page.R file, implying that the UI components specific to the home page are defined in a separate. This is one way of integrating the subpages, while it's also possible to source them directly in the tabs which is shown later within this explanation. The main structure of the Shiny dashboard is created using the dashboardPage function. It includes the following components:

Header: The dashboard header is set with the title of our application "German Predictor".

Sidebar: The dashboard sidebar is configured with a menu containing items for different tabs, each associated with a specific modeling technique or the EDA. The icons for each menu item are assigned using the icon() function from the shinydashboard package.

Body: The main body of the dashboard includes custom CSS styles, imported with includeCSS("www/styles.css"), to enhance the visual appearance of the application. Within the body, the tabItems function is used to define the content for each tab. The individual tabItem functions specify the elements within each tab.

In summary, this UI code sets up a Shiny dashboard with a sidebar menu for easy navigation between tabs, each dedicated to a specific predictive modeling technique as well as EDA and the home page. The inclusion of custom CSS styles contributes to an improved visual experience for the user.

Next, the subpages in the /views directory are discussed. This involves home_page.R, eda_page.R and knn_page.R as well as linear_regression_page.R, whereby the latter two are explained using linear regression, as they are very similar.

/views/home_page.R:

```
# homePageUI function definition
homePageUI <- function() {
  # Define the UI for the home page
  fluidPage(
    # First row
    fluidRow(
      # Welcome Title
      h1("Welcome to the German Predictor!", class = "welcome-title"),
      h2("Efficient, Precise and Always on Time.", class = "welcome-subtitle")

      # Three infoboxes
      infoBox(
        "Information about car models.",
        "Dataset 1: mtcars",
        icon = icon("car"),
        fill = TRUE,
        color = "purple",
      ), ...
    ),
    # Second row
    fluidRow(
      # Info box
      div(
        class = "explanation-box",
        box(
          title = "App Description",
          width = NULL,
          "Check out our new app...
        )
      )
    ),
    # Third row
    fluidRow(
      # Three columns with boxes
      column(width = 4,
             box(
               title = "Exploratory Data Analysis (EDA)",
               width = NULL,
               "Investigate the data you use for predictions.",
               status = "primary",
               solidHeader = TRUE,
               img(src = "EDA.png", width = "100%")  # Image for EDA
             )
      ),
      ...
    )
  )
}
```

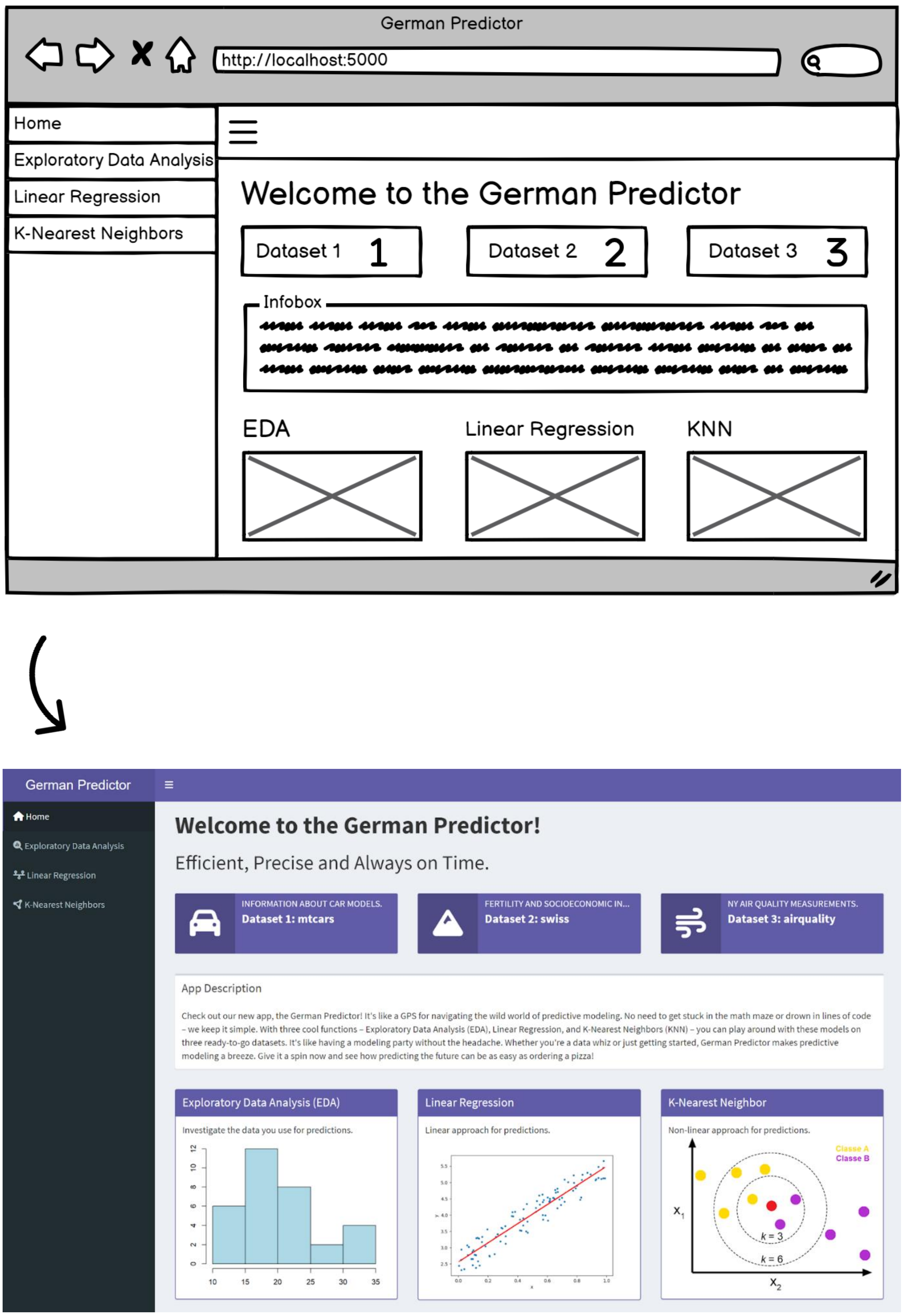
This page represents the first page the user sees when entering the web application. It creates a fluid page, allowing for a responsive layout that adjusts to different screen sizes. The UI is organized into three main rows, each serving a specific purpose:

First Row: The main and subtitle titles are displayed using the h1 and h2 functions, creating a welcoming introduction to the application. Three info boxes (infoBox) provide concise information about the available datasets. Each box includes a brief description, dataset name, and an icon representing the dataset type.

Second Row: An explanation box is presented using the div and box functions. This box provides a detailed description of the application's purpose, functionality, and the convenience it offers to users. The content is visually formatted to enhance readability.

Third Row: Three columns are defined using the fluidRow and column functions, each containing a box representing a different modeling technique. Each box includes a title, description, and an image relevant to the modeling technique or the EDA.

The code emphasizes a visually appealing and user-friendly design, providing informative content about the application's purpose, available datasets, and the three main modeling techniques. The use of images enhances understanding and engagement, making the application accessible to users with varying levels of expertise in predictive modeling.



Welcome to the German Predictor!

Efficient, Precise and Always on Time.

App Description

Check out our new app, the German Predictor! It's like a GPS for navigating the wild world of predictive modeling. No need to get stuck in the math maze or drown in lines of code – we keep it simple. With three cool functions – Exploratory Data Analysis (EDA), Linear Regression, and K-Nearest Neighbors (KNN) – you can play around with these models on three ready-to-go datasets. It's like having a modeling party without the headache. Whether you're a data whiz or just getting started, German Predictor makes predictive modeling a breeze. Give it a spin now and see how predicting the future can be as easy as ordering a pizza!

Investigate the data you use for predictions.

Linear approach for predictions.

Non-linear approach for predictions.

Figure 3: Screenshot Homepage Mockup vs. Product

/views/eda_page.R:

```
# Define a list of built-in dataframes in R
dataframes_list <- list(
  "mtcars" = mtcars,
  "swiss" = swiss,
  "airquality" = airquality
)

# Define the UI
ui <- fluidPage(
  # Page title with a magnifying glass icon
  h1(
    tags$span(icon("magnifying-glass-chart"), "Exploratory Data Analysis
(EDA)"),
    class = "titlePanel"
  ),

  # First row with an explanation box
  fluidRow(
    div(
      class = "explanation-box",
      box(
        title = "How to use this function?",
        width = NULL,
        "Exploratory Data Analysis (EDA) is a data analysis approach...
      )
    )
  ),

  # Dropdown to choose a dataframe
  selectInput("dataframe", "Choose a dataframe", choices =
names(dataframes_list)),

  # DataTable output for displaying datasets
  DTOutput("table"),

  # Output for displaying a correlation heatmap
  plotOutput("correlation_heatmap"),

  # Dropdown to choose a numerical variable for histogram
  selectInput("numeric_var", "Choose a numerical variable for histogram", ""),

  # Output for displaying a histogram
  plotOutput("histogram")
)
```

The code above defines the UI for the Exploratory Data Analysis page of the German Predictor. Similar to the previous explanation, the following section breaks down the important parts of the provided code snippet.

First Row: An explanation box is presented using the div and box functions, providing guidance on how to use the EDA function. The box includes information about the purpose of EDA, emphasizing visual exploration and summarization of data to gain insights before formal modeling.

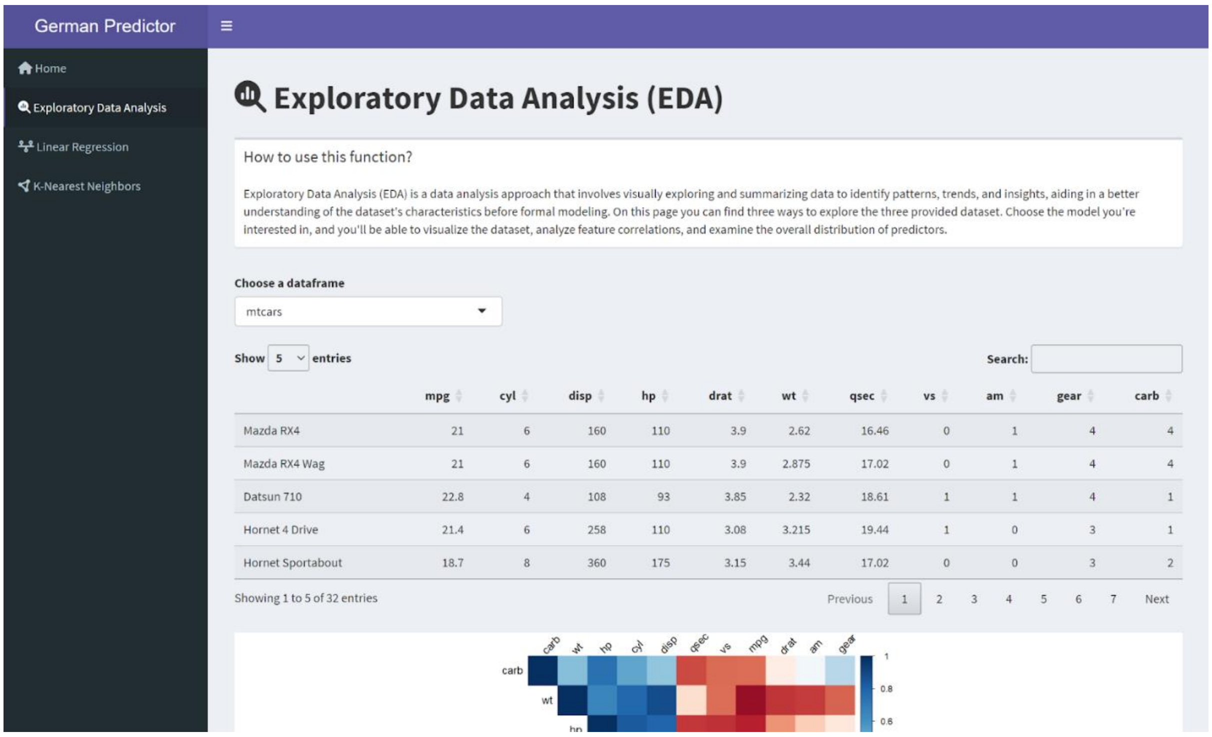
Dropdown to Choose a Dataframe: A selectInput component is defined, allowing the user to choose a dataframe from the list of built-in dataframes in R (mtcars, swiss, airquality). The choices for the dropdown are dynamically generated from the names of the dataframes.

DataTable Output: A DTOutput component is included to display the selected dataframe as an interactive DataTable. This facilitates a clear visualization of the dataset.

Correlation Heatmap Output: A plotOutput component is defined to display a correlation heatmap for the selected dataframe. This visualization helps users analyze feature correlations within the dataset.

Dropdown to Choose a Numerical Variable for Histogram: Another selectInput component is included, allowing users to choose a numerical variable for histogram analysis. The choices are dynamically generated based on the numeric variables present in the selected dataframe.

Histogram Output: Another plotOutput component is defined to display a histogram for the selected numerical variable. This visualization aids in understanding the distribution of the chosen predictor variable.



Exploratory Data Analysis (EDA)

How to use this function?

Exploratory Data Analysis (EDA) is a data analysis approach that involves visually exploring and summarizing data to identify patterns, trends, and insights, aiding in a better understanding of the dataset's characteristics before formal modeling. On this page you can find three ways to explore the three provided dataset. Choose the model you're interested in, and you'll be able to visualize the dataset, analyze feature correlations, and examine the overall distribution of predictors.

Choose a dataframe

mtcars

Show 5 entries

Search:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21	6	160	110	3.9	2.62	16.46	0	1	4	4
Mazda RX4 Wag	21	6	160	110	3.9	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.32	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.44	17.02	0	0	3	2

Showing 1 to 5 of 32 entries

Previous 1 2 3 4 5 6 7 Next

Figure 4: Screenshot of EDA Page

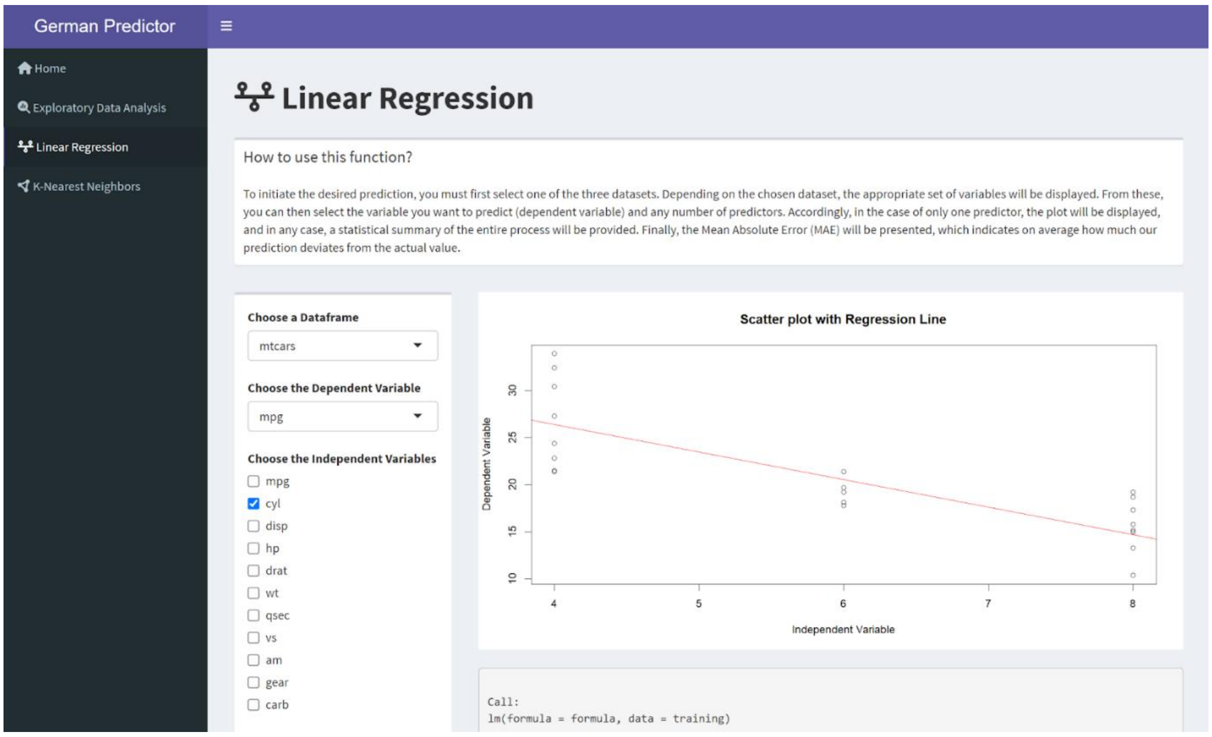
/views/linear_regression_page.R and /views/knn_page.R

```
# Define the UI for the Linear Regression page
ui <- fluidPage(
  h1(tags$span(icon("timeline"), "Linear Regression"), class = "titlePanel"),

  # First row with an explanation box
  fluidRow(
    div(
      class = "explanation-box",
      box(
        title = "How to use this function?",
        width = NULL,
        "To initiate the desired prediction, you must first select...
      ))),

  # Sidebar layout with input and output definitions
  sidebarLayout(
    box(
      # Dropdown to choose a dataframe
      selectInput("dataframe_2", "Choose a Dataframe", choices =
names(dataframes_list_2)),

      # Dynamic UI for selecting the dependent variable
      uiOutput("dependent_var_ui_2"),

      # Dynamic UI for selecting independent variables
      uiOutput("independent_vars_ui_2"),

      # Text output for informational messages
      textOutput("message"),
      width = 3,
      style = "background: transparent; padding: 15px;"
    ),

    # Main panel with output elements
    mainPanel(
      # Scatter plot for Linear Regression
      plotOutput("scatter_plot"),

      # Verbatim text output for statistical summary
      verbatimTextOutput("summary"),

      # Text output for Mean Absolute Error (MAE)
      htmlOutput("MAE"),

      # Table output
      htmlOutput("my_table"), width = 9
    )))
```

The UI for the Linear Regression page in our Shiny application is designed to facilitate intuitive and interactive linear regression analysis. The page features a title incorporating a timeline icon, emphasizing the focus on linear modeling. An explanation box guides users on utilizing the functionality, covering dataset selection, variable choices, plot display, statistical summaries, and the Mean Absolute Error (MAE) as an accuracy indicator.

The layout comprises a sidebar and a main panel. In the sidebar, users can choose a dataset from a dropdown menu and dynamically select the dependent and independent variables. Informational messages are displayed in a text output. The main panel presents a scatter plot illustrating the linear regression model, a statistical summary in verbatim text, and the MAE as a text output. Additionally, a table output provides supplementary information.

This design aims for clarity and adaptability, allowing users to seamlessly perform linear regression on their chosen dataset. Dynamic UI elements enhance user interaction, while the organized sidebar layout ensures an intuitive user experience.



Linear Regression

How to use this function?

To initiate the desired prediction, you must first select one of the three datasets. Depending on the chosen dataset, the appropriate set of variables will be displayed. From these, you can then select the variable you want to predict (dependent variable) and any number of predictors. Accordingly, in the case of only one predictor, the plot will be displayed, and in any case, a statistical summary of the entire process will be provided. Finally, the Mean Absolute Error (MAE) will be presented, which indicates on average how much our prediction deviates from the actual value.

Choose a Dataframe: mtcars

Choose the Dependent Variable: mpg

Choose the Independent Variables
- [ ] mpg
- [x] cyl
- [ ] disp
- [ ] hp
- [ ] drat
- [ ] wt
- [ ] qsec
- [ ] vs
- [ ] am
- [ ] gear
- [ ] carb

```
Call:
lm(formula = formula, data = training)
```

Figure 5: Screenshot Linear Regression Page

For KNN the procedure of building the UI was similar, but instead of displaying a scatter plot with a regression line, the subpage consists of a graph which is shown below and that visualizes the RMSE (residual mean squared error) with an increasing number of k’s.

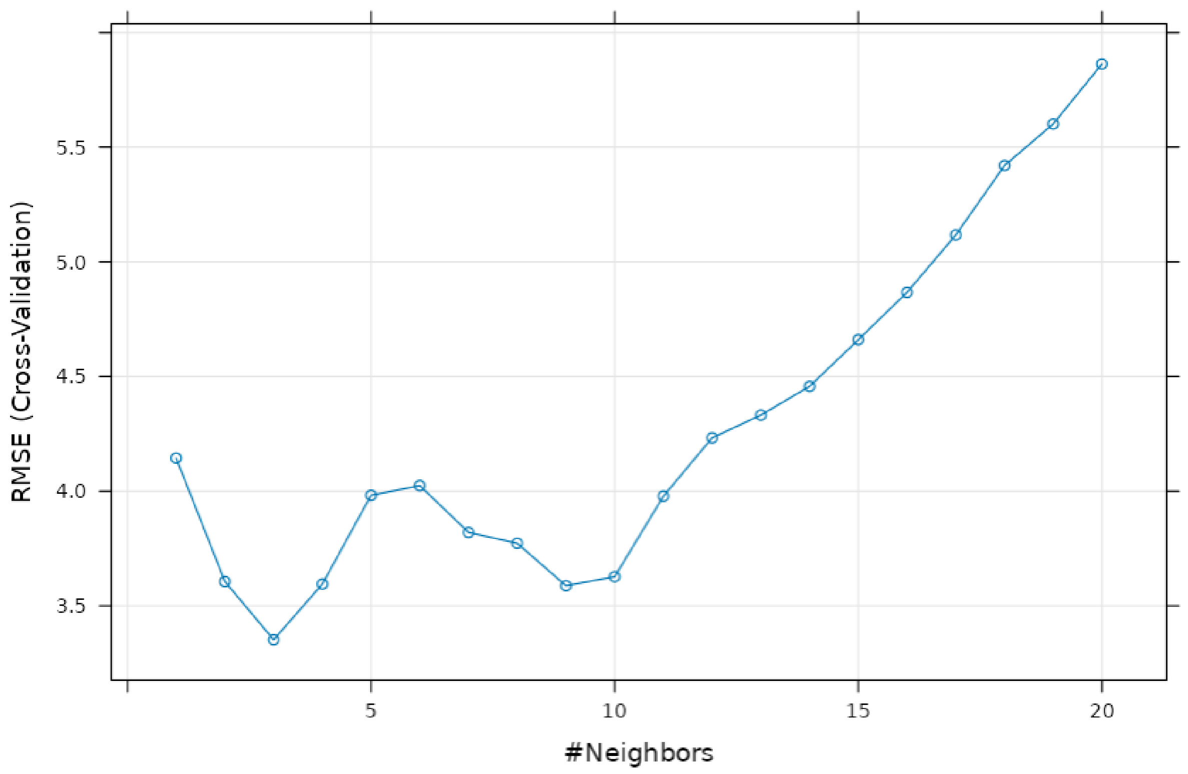


Figure 6: Screenshot of KNN Plot from KNN Page

5. Conclusion

In conclusion, our Shiny application “German Predictor” addresses a crucial need in data science education, providing an accessible platform for interactive exploration and understanding of predictive modeling concepts. Motivated by the increasing importance of data-driven decision-making, our application demonstrates predictive modeling, allowing users to engage with key techniques without the requirement of extensive coding knowledge.

The application offers three main functionalities: Exploratory Data Analysis (EDA), Linear Regression, and K-Nearest Neighbors (KNN). EDA serves as the foundation for data-driven endeavors, enabling users to comprehend dataset characteristics. Linear Regression introduces a fundamental predictive modeling technique, while KNN showcases a non-linear approach. These features collectively aim to empower learners with a comprehensive and interactive educational experience.

Looking ahead, the future development of our Shiny application holds great potential for expanding its educational capabilities and user functionality. Here are key areas for future work:

Diverse Algorithmic Integration: Including additional machine learning algorithms, such as Decision Trees, Random Forests, and Support Vector Machines, would enrich the educational experience. This expansion ensures users encounter a broader spectrum of modeling techniques, catering to diverse educational needs and skill levels.

Expert Mode: The introduction of an "Expert Mode" stands as a crucial enhancement. This feature would empower users to customize the abstraction level of the modeling process. Beginners can benefit from a guided approach, while advanced users can delve into a more intricate and customizable workflow, adjusting parameters and exploring advanced functionalities.

Dataset Flexibility: Enabling users to import their datasets expands the application's versatility. While currently offering three built-in datasets, allowing users to upload their data provides a more personalized and adaptable learning experience.

Results Export: Incorporating an export feature for results would enhance the application's practical utility. Users could export model outcomes, visualizations, and summaries for further analysis or inclusion in reports, contributing to a more seamless integration into their workflow.

Algorithm Comparison Page: A dedicated comparison page would enable users to assess the performance of different algorithms without switching between pages. This feature would streamline the learning process, allowing users to make informed decisions about algorithm selection based on direct comparisons.

In conclusion, the envisioned future developments for our Shiny application are set to elevate its role not only in education but also in real-world applications within the industry and beyond. The expanded algorithmic repertoire and customization options empower users to tackle real-world challenges, providing a bridge between theoretical learning and practical implementation.