````
---
title: "Build and deploy a stroke prediction model using R"
date: "`r Sys.Date()`"
output: html_document
author: "Priyanshu Jha!"
---

# About Data Analysis Report

This RMarkdown file contains the report of the data analysis done for the project on building and deploying a stroke prediction model in R. It contains analysis such as data exploration, summary statistics and building the prediction models. The final report was completed on `r date()`.

**Data Description:**

According to the World Health Organization (WHO) stroke is the 2nd leading cause of death globally, responsible for approximately 11% of total deaths.

This data set is used to predict whether a patient is likely to get stroke based on the input parameters like gender, age, various diseases, and smoking status. Each row in the data provides relevant information about the patient.

# Task One: Import data and data preprocessing

## Load data and install packages

```{r}
installed.packages()
install.packages("tidymodels")
install.packages("tidyverse")
install.packages("workflows")
install.packages("tune")
install.packages("readr")
install.packages("skimr")

# load the relevant tidymodels libraries
library(tidymodels)
library(tidyverse)
library(workflows)
library(tune)
library(readr)


data <- read_csv("healthcare-dataset-stroke-data.csv")
head(data)

```

## Describe and explore the data

```{r}

# Load necessary libraries
library(skimr)      # For detailed data summaries
library(ggplot2)    # For data visualizations
library(dplyr)      # For data manipulation
library(tidyr)      # For data cleaning

# View the first few rows
head(data)

# Get the structure of the data
str(data)

# Summary statistics
summary(data)

# Skimr package for detailed overview
````
````

```
skim(data)

# Replace "Unknown" or NA with actual missing values
data_cleaned <- data %>%
  mutate(across(where(is.character), ~na_if(., "Unknown"))) %>%
  mutate(across(where(is.character), as.factor))  # Convert character to factors if
needed

# Check for missing values
sum(is.na(data_cleaned))  # Total missing values

# Count missing values per column
colSums(is.na(data_cleaned))

# Install and load the naniar package
install.packages("naniar")
library(naniar)

# Visualize missing values
gg_miss_var(data_cleaned, show_pct = TRUE)  # Missing values per variable
vis_miss(data_cleaned)  # Visualize missing patterns


# Convert character columns to factors
data_cleaned <- data_cleaned %>%
  mutate(across(where(is.character), as.factor))


install.packages("mice")
library(mice)


# Impute missing data with mice
imputed_data <- mice(data_cleaned, m = 5, method = 'pmm', seed = 123)

# Extract the completed dataset
data_cleaned <- complete(imputed_data)

# Check for remaining missing values
sum(is.na(data_cleaned))  # Should return 0

# Inspect the data
str(data_cleaned)
head(data_cleaned)

# Convert factors back to characters
data_cleaned <- data_cleaned %>%
  mutate(across(where(is.factor), as.character))

# Select only numeric columns
numeric_data <- data_cleaned %>% select(where(is.numeric))

# Calculate the correlation matrix
cor_matrix <- cor(numeric_data, use = "complete.obs")
print(cor_matrix)

# Visualize the correlation matrix (optional)
install.packages('corrplot')
library(corrplot)
corrplot::corrplot(cor_matrix, method = "color", type = "upper")

# Ensure stroke is a factor for proper grouping and filling
data_cleaned <- data_cleaned %>%
  mutate(stroke = as.factor(stroke))  # Convert stroke to a factor

# Create the boxplot
ggplot(data_cleaned, aes(x = stroke, y = age, fill = stroke)) +
  geom_boxplot() +
```

```
  labs(
    title = "Age by Stroke Outcome",
    x = "Stroke (0 = No, 1 = Yes)",
    y = "Age"
  ) +
  theme_minimal() +
  scale_fill_manual(values = c("0" = "blue", "1" = "red"))  # Optional: Customize
colors


```

Task Two: Build prediction models

```{r}
# Load necessary libraries
install.packages("themis")
library(tidymodels)  # For modeling workflow
library(tidyverse)   # For data wrangling and visualization
library(themis)      # For handling class imbalance


# Ensure stroke is a factor
data_cleaned <- data_cleaned %>%
  mutate(stroke = as.factor(stroke))

# Split data into predictors and response
data_split <- initial_split(data_cleaned, prop = 0.8, strata = stroke)
train_data <- training(data_split)
test_data <- testing(data_split)


# Check distribution of target variable
train_data %>%
  count(stroke) %>%
  mutate(prop = n / sum(n))

library(tidyverse)

# Check class distribution before balancing
table(train_data$stroke)

# Set seed for reproducibility
set.seed(123)

# Separate minority and majority classes
minority_class <- train_data %>% filter(stroke == 1)
majority_class <- train_data %>% filter(stroke == 0)

# Random oversample the minority class
oversampled_minority <- minority_class %>%
  slice_sample(n = nrow(majority_class), replace = TRUE)

# Combine the oversampled minority class with the majority class
balanced_train_data <- bind_rows(majority_class, oversampled_minority)

# Check class distribution after balancing
table(balanced_train_data$stroke)

# Preprocessing recipe (including the balanced data)
preprocessor <- recipe(stroke ~ ., data = balanced_train_data) %>%
  step_novel(all_nominal()) %>%  # Handle unseen factor levels
  step_dummy(all_nominal(), -all_outcomes()) %>%  # One-hot encode categorical
variables
  step_zv(all_predictors()) %>%  # Remove zero variance predictors
  step_normalize(all_numeric())  # Normalize numeric variables

# Prep the recipe with the balanced data
prepped_data <- prep(preprocessor, training = balanced_train_data)
```

```
# Apply the preprocessed data to the training set
train_processed <- bake(prepped_data, new_data = balanced_train_data)

# Check the processed data
head(train_processed)


# Install and load the necessary packages
install.packages("tidymodels")
install.packages("xgboost")
install.packages("nnet")
install.packages("ranger")
install.packages("kernlab")
install.packages("kknn")

library(tidymodels)
library(xgboost)
library(nnet)  # For Neural Networks
library(ranger) # For Random Forest
library(kernlab) # For Support Vector Machines
library(kknn) # For k-Nearest Neighbors

#KNN
knn_model <- nearest_neighbor() %>%
  set_mode("classification") %>%
  set_engine("kknn")


#SVM
knn_fit <- knn_model %>%
  fit(stroke ~ ., data = train_processed)

svm_model <- svm_linear() %>%
  set_mode("classification") %>%
  set_engine("kernlab")

svm_fit <- svm_model %>%
  fit(stroke ~ ., data = train_processed)

#Random Forest
rf_model <- rand_forest(trees = 500) %>%
  set_mode("classification") %>%
  set_engine("ranger")

rf_fit <- rf_model %>%
  fit(stroke ~ ., data = train_processed)



#Classification and Regression Trees (CART)
cart_model <- decision_tree() %>%
  set_mode("classification") %>%
  set_engine("rpart")

cart_fit <- cart_model %>%
  fit(stroke ~ ., data = train_processed)




```

Task Three: Evaluate and select prediction models

```{r}
# Install yardstick if not already installed
```

```
install.packages("yardstick")

# Load the package
library(yardstick)

# Install tibble if not already installed
install.packages("tibble")

# Load the tibble package
library(tibble)

# Ensure all prediction columns are the same length
# First, find the minimum length among all prediction columns
min_length <- min(
  length(knn_predictions$.pred_class),
  length(svm_predictions$.pred_class),
  length(rf_predictions$.pred_class),
  length(cart_predictions$.pred_class)
)

# Subset all prediction columns to this minimum length
knn_pred_class <- knn_predictions$.pred_class[1:min_length]
svm_pred_class <- svm_predictions$.pred_class[1:min_length]
rf_pred_class <- rf_predictions$.pred_class[1:min_length]
cart_pred_class <- cart_predictions$.pred_class[1:min_length]

# Create the tibble with matched length columns
model_predictions <- tibble(
  knn = knn_pred_class,
  svm = svm_pred_class,
  rf = rf_pred_class,
  cart = cart_pred_class
)

# Verify the structure
str(model_predictions)


#evaluation

library(dplyr)

# Function to align predictions and actual values
align_predictions <- function(predictions, test_data) {
  # Ensure predictions and actual values have consistent length
  min_length <- min(nrow(predictions), nrow(test_data))

  # Select only the first min_length rows
  predictions <- predictions[1:min_length, ]
  test_data <- test_data[1:min_length, ]

  # Combine predictions with actual stroke values
  result <- predictions %>%
    mutate(stroke = test_data$stroke)

  return(result)
}

# Predictions for each model with alignment
knn_predictions <- predict(knn_fit, new_data = test_processed) %>%
  align_predictions(test_processed)

svm_predictions <- predict(svm_fit, new_data = test_processed) %>%
  align_predictions(test_processed)

rf_predictions <- predict(rf_fit, new_data = test_processed) %>%
  align_predictions(test_processed)

cart_predictions <- predict(cart_fit, new_data = test_processed) %>%
```

```
  align_predictions(test_processed)

library(yardstick)

# Ensure consistent factor levels
knn_predictions <- knn_predictions %>%
  mutate(
    .pred_class = factor(.pred_class, levels = c("0", "1")),
    stroke = factor(stroke, levels = c("0", "1"))
  )

svm_predictions <- svm_predictions %>%
  mutate(
    .pred_class = factor(.pred_class, levels = c("0", "1")),
    stroke = factor(stroke, levels = c("0", "1"))
  )

rf_predictions <- rf_predictions %>%
  mutate(
    .pred_class = factor(.pred_class, levels = c("0", "1")),
    stroke = factor(stroke, levels = c("0", "1"))
  )

cart_predictions <- cart_predictions %>%
  mutate(
    .pred_class = factor(.pred_class, levels = c("0", "1")),
    stroke = factor(stroke, levels = c("0", "1"))
  )

# Now create confusion matrices
knn_cm <- conf_mat(knn_predictions, truth = stroke, estimate = .pred_class)
svm_cm <- conf_mat(svm_predictions, truth = stroke, estimate = .pred_class)
rf_cm <- conf_mat(rf_predictions, truth = stroke, estimate = .pred_class)
cart_cm <- conf_mat(cart_predictions, truth = stroke, estimate = .pred_class)

# Performance Metrics
knn_metrics <- knn_predictions %>%
  metrics(truth = stroke, estimate = .pred_class)

svm_metrics <- svm_predictions %>%
  metrics(truth = stroke, estimate = .pred_class)

rf_metrics <- rf_predictions %>%
  metrics(truth = stroke, estimate = .pred_class)

cart_metrics <- cart_predictions %>%
  metrics(truth = stroke, estimate = .pred_class)

# Combine Metrics
model_performance <- bind_rows(
  knn_metrics %>% mutate(model = "KNN"),
  svm_metrics %>% mutate(model = "SVM"),
  rf_metrics %>% mutate(model = "Random Forest"),
  cart_metrics %>% mutate(model = "CART")
)

# Visualization of Performance
library(ggplot2)

# Accuracy Comparison
ggplot(model_performance %>% filter(.metric == "accuracy"),
       aes(x = model, y = .estimate, fill = model)) +
  geom_bar(stat = "identity") +
  labs(title = "Model Accuracy Comparison",
       y = "Accuracy", x = "Model") +
  theme_minimal()


# Print Detailed Performance
```

```
print(model_performance)


```

Task Four: Deploy the prediction model

```{r}
# CLI Prediction Function
predict_stroke <- function(input_data) {
  # Validate input data
  required_cols <- c("age", "hypertension", "heart_disease", "ever_married",
                     "work_type", "Residence_type", "avg_glucose_level",
                     "bmi", "smoking_status", "gender")

  # Check if all required columns are present
  missing_cols <- setdiff(required_cols, names(input_data))
  if (length(missing_cols) > 0) {
    stop(paste("Missing columns:", paste(missing_cols, collapse=", ")))
  }

  # Preprocessing function
  preprocess_input <- function(data) {
    # One-hot encoding
    data$gender_Male <- as.integer(data$gender == "Male")
    data$gender_Female <- as.integer(data$gender == "Female")

    data$ever_married_Yes <- as.integer(data$ever_married == "Yes")
    data$ever_married_No <- as.integer(data$ever_married == "No")

    # Work type encoding
    work_types <- c("Private", "Self-employed", "Govt_job", "Children", "Never_worked")
    for(wt in work_types) {
      col_name <- paste0("work_type_", gsub("-", "_", wt))
      data[[col_name]] <- as.integer(data$work_type == wt)
    }

    # Residence type encoding
    data$Residence_type_Urban <- as.integer(data$Residence_type == "Urban")
    data$Residence_type_Rural <- as.integer(data$Residence_type == "Rural")

    # Smoking status encoding
    smoking_statuses <- c("formerly smoked", "never smoked", "smokes", "Unknown")
    for(ss in smoking_statuses) {
      col_name <- paste0("smoking_status_", gsub(" ", "_", ss))
      data[[col_name]] <- as.integer(data$smoking_status == ss)
    }

    # Select only processed columns
    processed_cols <- names(train_processed)[!names(train_processed) %in% "stroke"]
    return(data[, processed_cols, drop = FALSE])
  }

  # Preprocess input
  tryCatch({
    processed_input <- preprocess_input(input_data)

    # Predict
    prediction <- predict(rf_fit, new_data = processed_input)

    # Return prediction
    return(list(
      prediction_class = prediction$.pred_class,
      prediction_prob = prediction$.pred_1
    ))
  }, error = function(e) {
    stop(paste("Preprocessing error:", e$message))
  })
}
```

```
# Example CLI Interface
cli_stroke_predictor <- function() {
  cat("Stroke Prediction CLI\n")
  cat("Please enter patient details:\n")

  # Interactive input
  age <- as.numeric(readline("Age: "))
  gender <- readline("Gender (Male/Female): ")
  hypertension <- as.numeric(readline("Hypertension (0/1): "))
  heart_disease <- as.numeric(readline("Heart Disease (0/1): "))
  ever_married <- readline("Ever Married (Yes/No): ")
  work_type <- readline("Work Type (Private/Self-
employed/Govt_job/Children/Never_worked): ")
  residence_type <- readline("Residence Type (Urban/Rural): ")
  avg_glucose_level <- as.numeric(readline("Average Glucose Level: "))
  bmi <- as.numeric(readline("BMI: "))
  smoking_status <- readline("Smoking Status (formerly smoked/never
smoked/smokes/Unknown): ")

  # Create input data frame
  input_data <- data.frame(
    age = age,
    hypertension = hypertension,
    heart_disease = heart_disease,
    ever_married = ever_married,
    work_type = work_type,
    Residence_type = residence_type,
    avg_glucose_level = avg_glucose_level,
    bmi = bmi,
    smoking_status = smoking_status,
    gender = gender
  )

  # Predict
  tryCatch({
    result <- predict_stroke(input_data)

    cat("\nStroke Prediction Results:\n")
    cat("Prediction Class:", ifelse(result$prediction_class == 1, "High Risk", "Low
Risk"), "\n")
    cat("Probability of Stroke:", round(result$prediction_prob * 100, 2), "%\n")
  }, error = function(e) {
    cat("Error:", e$message, "\n")
  })
}

# Run the CLI
cli_stroke_predictor()

```

Task Five: Findings and Conclusions