Overview of the Wine Quality Dataset:

The Wine Quality dataset contains various physicochemical properties and sensory quality ratings of white wines. It is often used in machine learning and data analysis to predict the quality of wines based on their chemical composition. The dataset consists of 11 physicochemical attributes and one target variable representing the wine quality rating.

Key Features:

Number of Instances: The dataset contains 4,898 instances of white wine samples.

Number of Attributes: There are 11 input attributes (predictors) representing different chemical properties of the wines.

Target Variable: The dataset includes one target variable, "quality," which represents the sensory quality rating of wines on a scale from 0 to 10.

Attribute Information: The attributes include:

Fixed Acidity: The nonvolatile acids in the wine.
Volatile Acidity: The amount of acetic acid in the wine, which contributes to vinegar taste.
Citric Acid: The presence of citric acid, which adds freshness to the wine.
Residual Sugar: The amount of sugar remaining after fermentation.
Chlorides: The amount of salt in the wine.
Free Sulfur Dioxide: The amount of free SO2, which prevents microbial growth.
Total Sulfur Dioxide: The total SO2 present, including the bound and free forms.
Density: The density of the wine, relative to water.
pH: The acidity level of the wine.
Sulphates: The amount of sulfur dioxide in the wine.
Alcohol: The alcohol content in the wine.

Code:

```
# Load required libraries
install.packages("rpart")
library(rpart)
library(rpart.plot)

# Load the Wine Quality dataset
wine_data <- read.csv("D:/Bsc life/10th semester/Data
Mining/Final Project/winequality-white.csv", sep = ";")
```

```
# Define the target variable and features
target_var <- "quality"
features <- names(wine_data)[1:11]

# Define k for k-fold cross-validation
k <- 5

# Split the dataset into k folds
set.seed(123)  # For reproducibility
fold_indices <- split(1:nrow(wine_data), cut(1:nrow(wine_data),
breaks = k, labels = FALSE))

# Initialize variables to store results
accuracy_info_gain <- vector("numeric", length = k)
accuracy_gini <- vector("numeric", length = k)
accuracy_gain_ratio <- vector("numeric", length = k)

confusion_matrices_info_gain <- list()
confusion_matrices_gini <- list()
confusion_matrices_gain_ratio <- list()

# Perform k-fold cross-validation for each criterion
for (i in 1:k) {
 # Extract the current fold's indices
  test_indices <- fold_indices[[i]]
  train_indices <- unlist(fold_indices[-i])

  # Create training and testing datasets
  train_data <- wine_data[train_indices, ]
  test_data <- wine_data[test_indices, ]

  # Fit the decision tree model with Information Gain
  decision_tree_info_gain <- rpart(formula(paste(target_var, "~",
paste(features, collapse = "+"))),
                                   data = train_data,
                                   method = "class",
                                   parms = list(split = "information"))
```

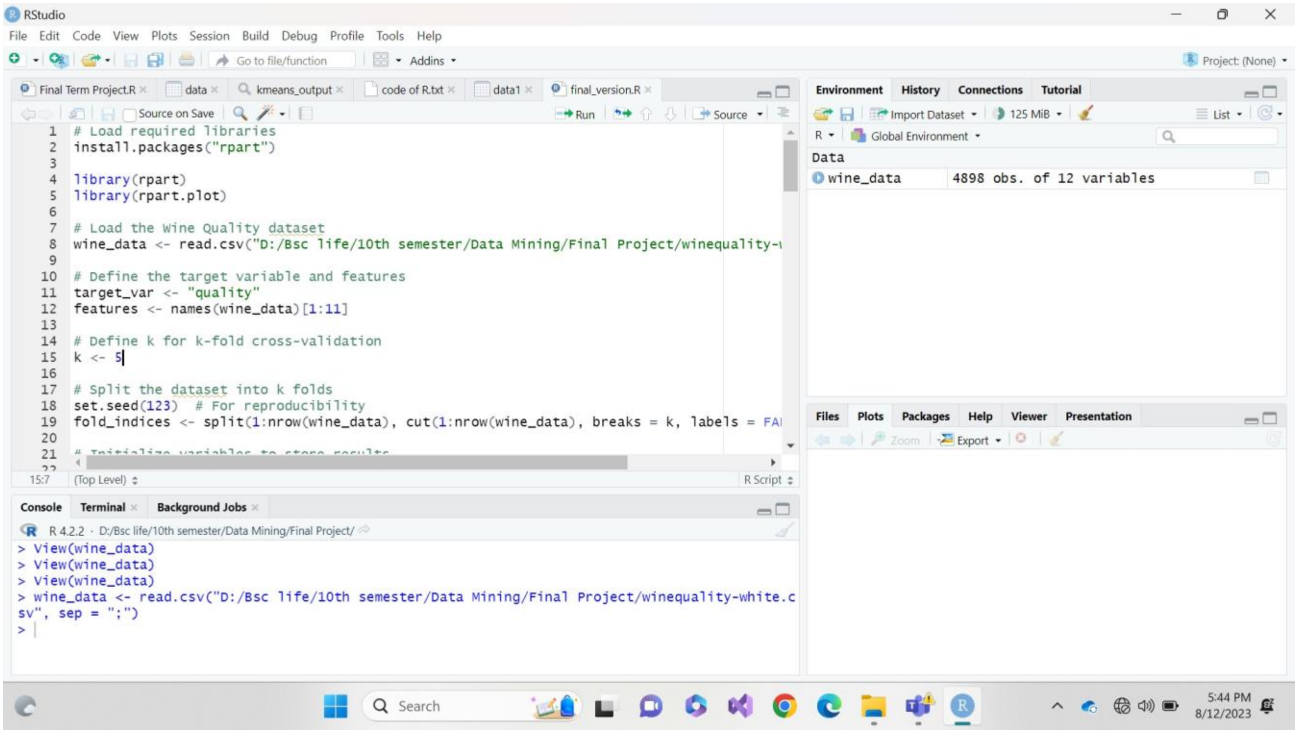
```
 # Fit the decision tree model with Gini Index
  decision_tree_gini <- rpart(formula(paste(target_var, "~",
paste(features, collapse = "+"))),
                              data = train_data,
                              method = "class",
                              parms = list(split = "gini"))

  # Fit the decision tree model with Gain Ratio
  decision_tree_gain_ratio <- rpart(formula(paste(target_var, "~",
paste(features, collapse = "+"))),
                                    data = train_data,
                                    method = "class",
                                    parms = list(split = "gainratio"))

  # Make predictions on the test data for each criterion
  predictions_info_gain <- predict(decision_tree_info_gain,
test_data, type = "class")
  predictions_gini <- predict(decision_tree_gini, test_data, type =
"class")
  predictions_gain_ratio <- predict(decision_tree_gain_ratio,
test_data, type = "class")

  # Calculate accuracy for each criterion
  accuracy_info_gain[i] <- mean(predictions_info_gain ==
test_data$quality)
  accuracy_gini[i] <- mean(predictions_gini == test_data$quality)
  accuracy_gain_ratio[i] <- mean(predictions_gain_ratio ==
test_data$quality)

  # Create confusion matrix for each criterion
  confusion_matrices_info_gain[[i]] <- table(Actual =
test_data$quality, Predicted = predictions_info_gain)
  confusion_matrices_gini[[i]] <- table(Actual = test_data$quality,
Predicted = predictions_gini)
  confusion_matrices_gain_ratio[[i]] <- table(Actual =
test_data$quality, Predicted = predictions_gain_ratio)
}
```

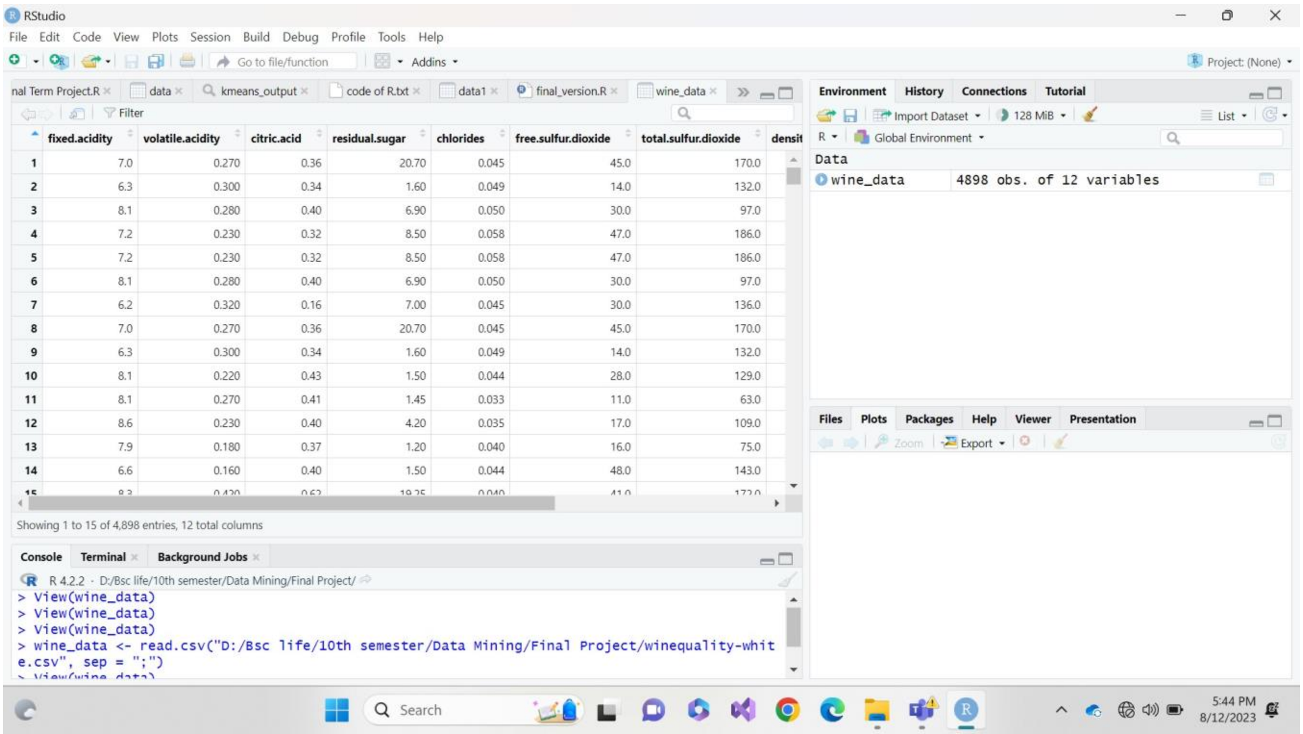
```
# Calculate the average accuracy for each criterion
average_accuracy_info_gain <- mean(accuracy_info_gain)
average_accuracy_gini <- mean(accuracy_gini)
average_accuracy_gain_ratio <- mean(accuracy_gain_ratio)

# Print the average accuracy for each criterion
cat("Average Accuracy with Information Gain:",
average_accuracy_info_gain, "\n")
cat("Average Accuracy with Gini Index:", average_accuracy_gini,
"\n")
cat("Average Accuracy with Gain Ratio:",
average_accuracy_gain_ratio, "\n")

# Print confusion matrices for each criterion
for (i in 1:k) {
  cat("Confusion Matrix for Information Gain (Fold", i, "):\n")
  print(confusion_matrices_info_gain[[i]])

  cat("Confusion Matrix for Gini Index (Fold", i, "):\n")
  print(confusion_matrices_gini[[i]])

  cat("Confusion Matrix for Gain Ratio (Fold", i, "):\n")
  print(confusion_matrices_gain_ratio[[i]])
}

# Visualize decision trees using rpart.plot
#options(repr.plot.width = 1000, repr.plot.height = 500)
# Visualize decision trees using rpart.plot with adjusted plot
dimensions
options(repr.plot.width = 800, repr.plot.height = 400)
print("Decision Tree with Information Gain")
rpart.plot(decision_tree_info_gain)

print("Decision Tree with Gini Index:")
rpart.plot(decision_tree_gini)

print("Decision Tree with Gain Ratio:")
rpart.plot(decision_tree_gain_ratio)
```

```
print("Decision Tree with Information Gain")
rpart.plot(decision_tree_info_gain)

print("Decision Tree with Gini Index:")
rpart.plot(decision_tree_gini)

print("Decision Tree with Gain Ratio:")
rpart.plot(decision_tree_gain_ratio)
```

Outputs:

1. **Load the Wine Quality dataset**
   wine_data <- read.csv ("D:/Bsc life/10th semester/Data Mining/Final Project/winequality-white.csv", sep = ";")

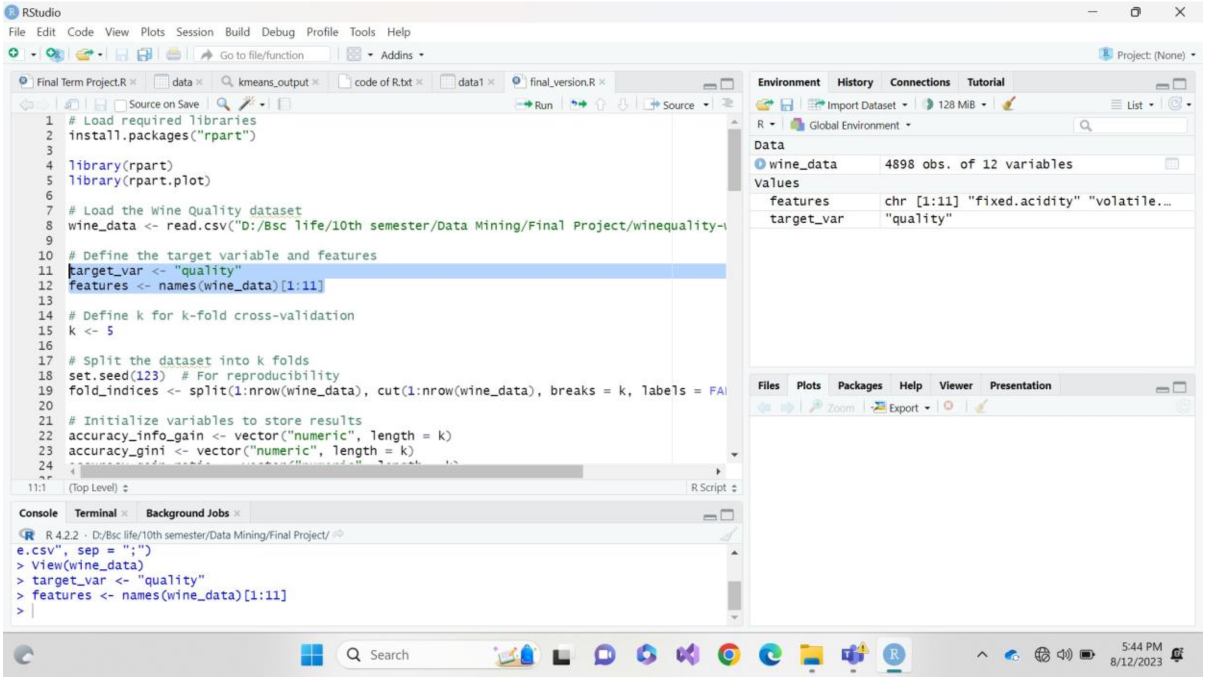




2. Define the target variable and features

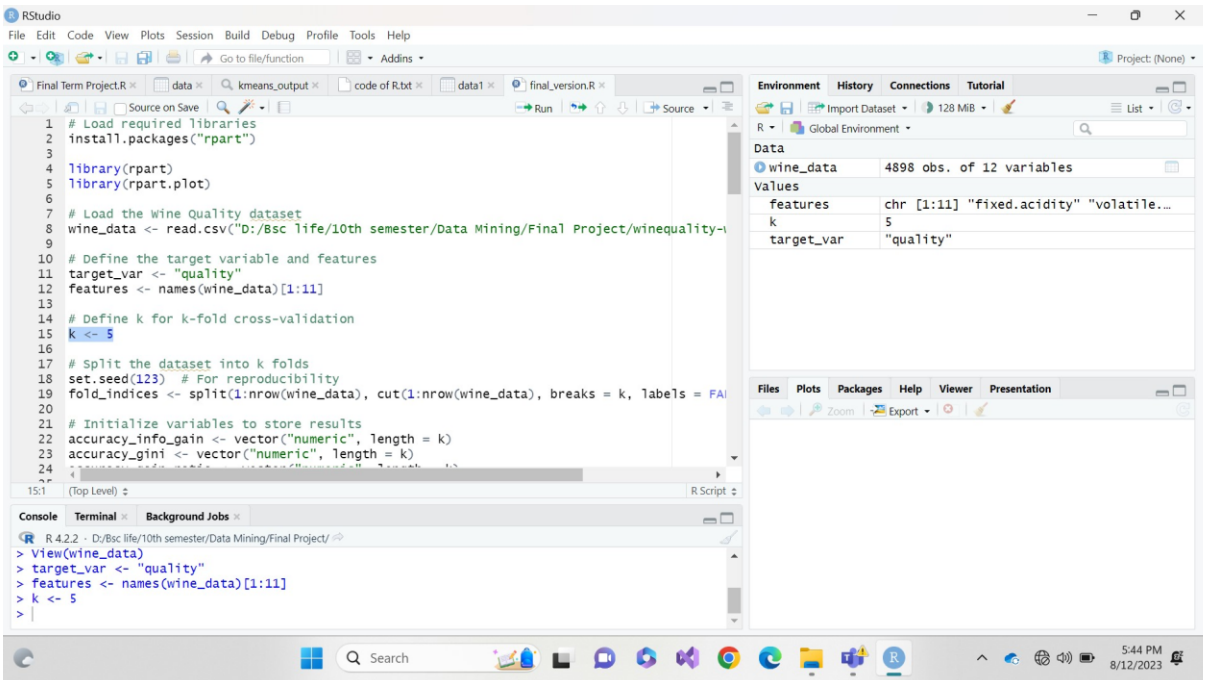
target_var <- "quality"

features <- names(wine_data)[1:11]



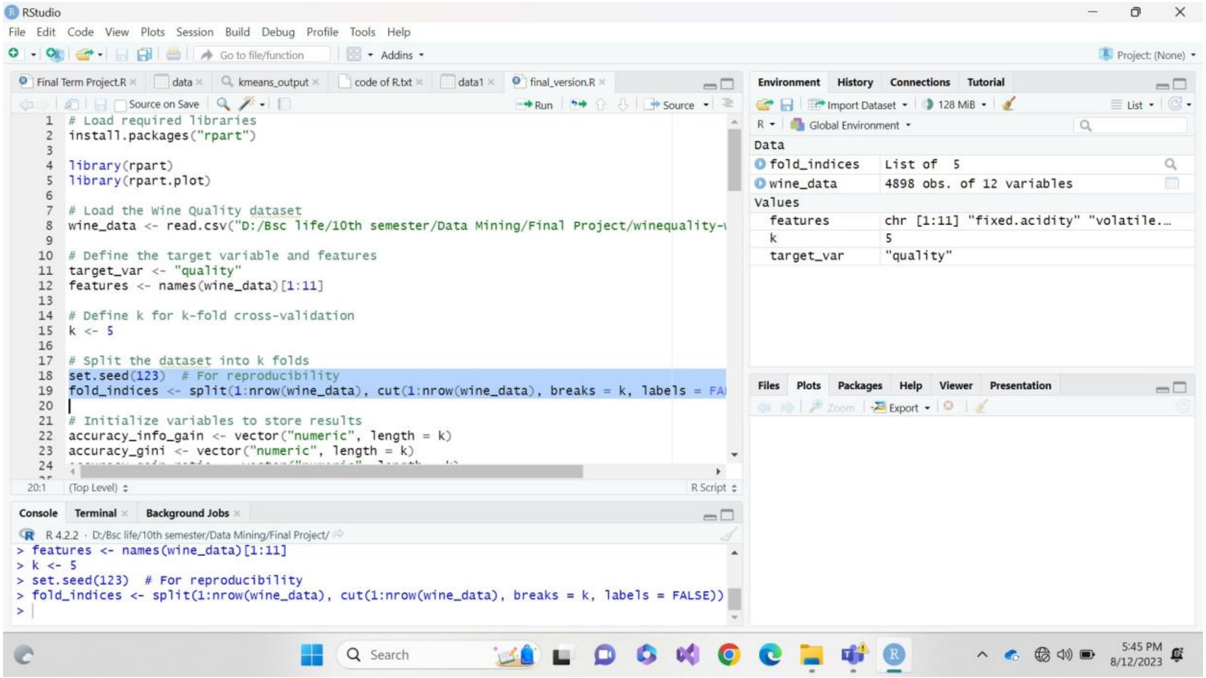
3. Define k for k-fold cross-validation

k <- 5



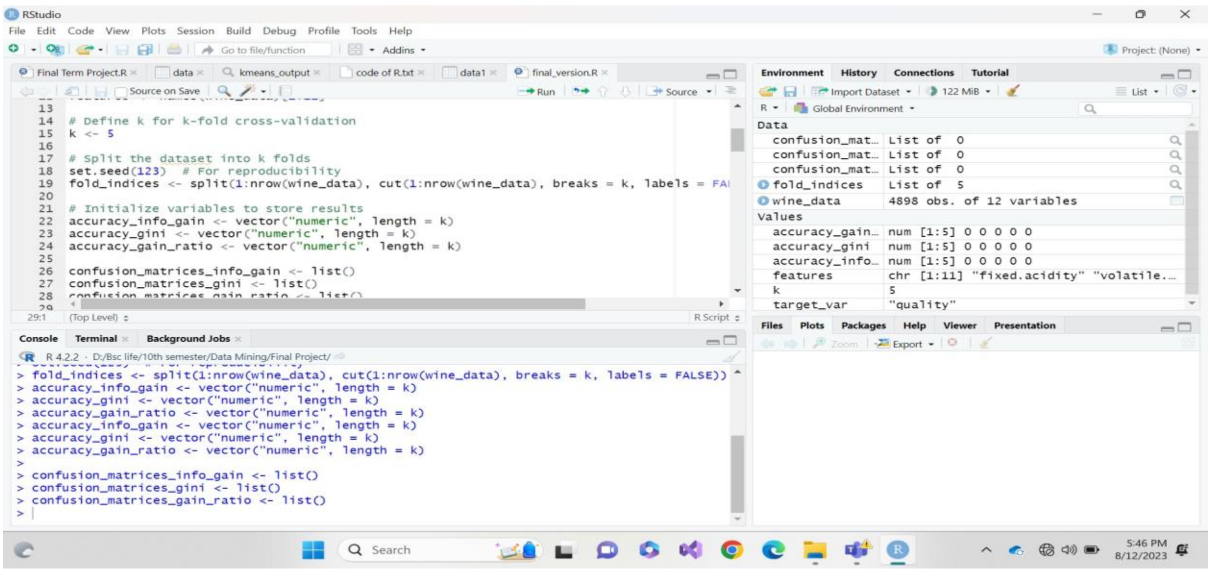
4. Split the dataset into k folds

```
set.seed(123)  # For reproducibility
fold_indices <- split(1:nrow(wine_data), cut(1:nrow(wine_data),
breaks = k, labels = FALSE))
```



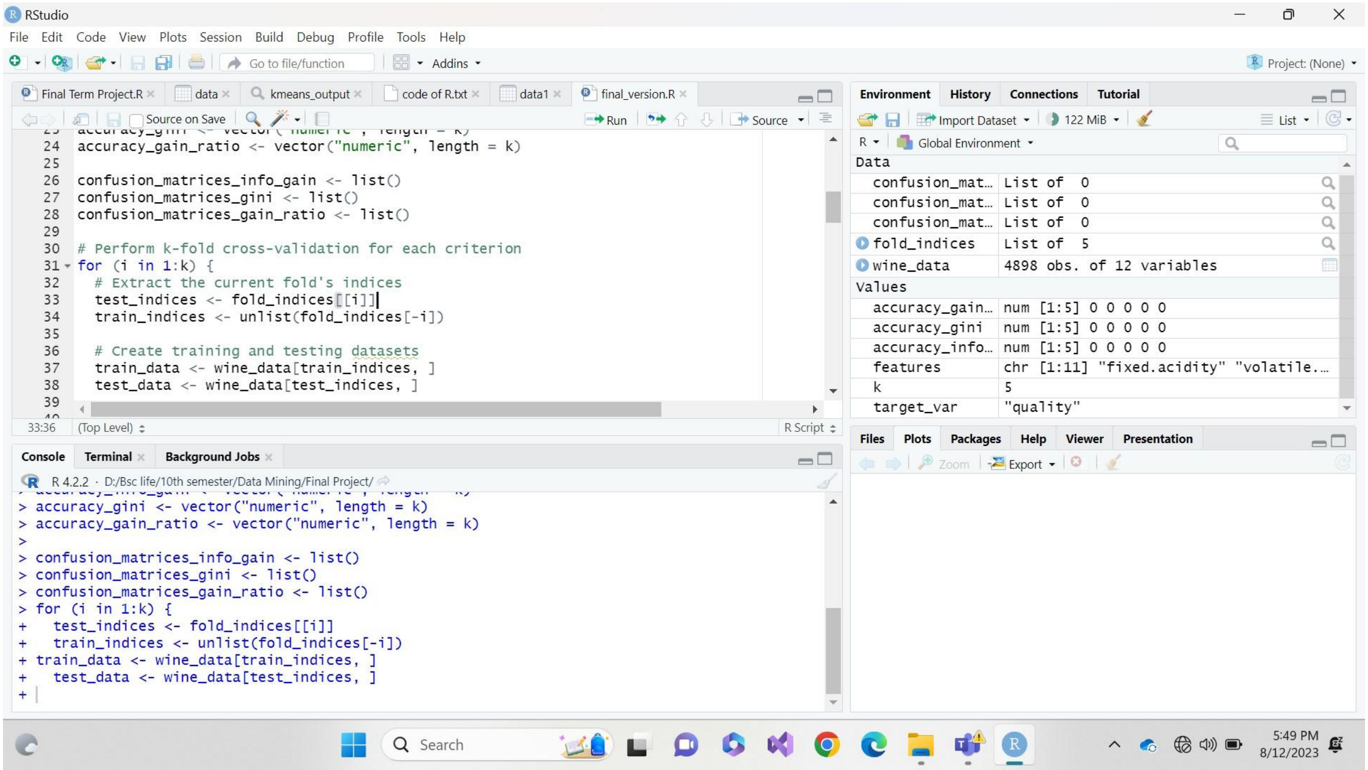
5. Initialize variables to store results

```
accuracy_info_gain <- vector("numeric", length = k)
accuracy_gini <- vector("numeric", length = k)
accuracy_gain_ratio <- vector("numeric", length = k)
confusion_matrices_info_gain <- list()
confusion_matrices_gini <- list()
confusion_matrices_gain_ratio <- list()
```



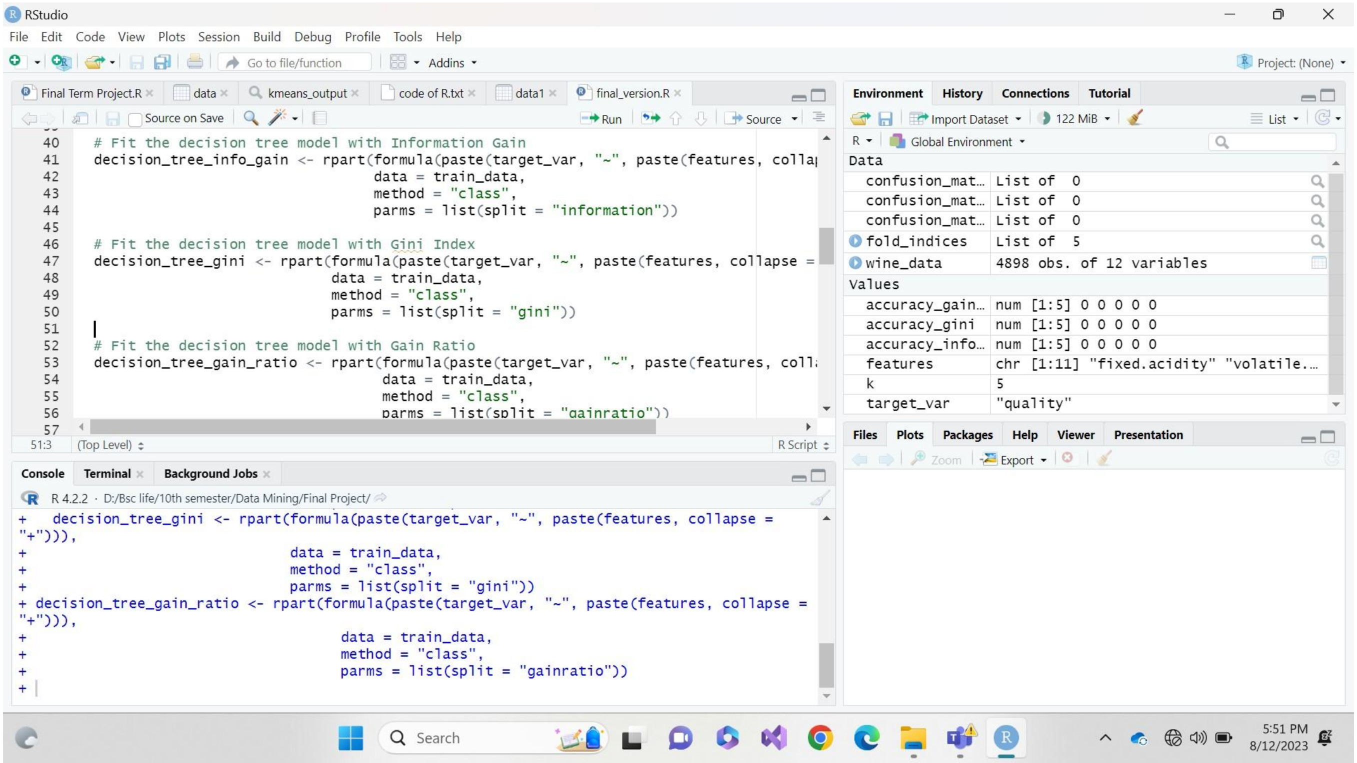
6. Perform k-fold cross-validation for each criterion

```
for (i in 1:k) {
# Extract the current fold's indices
  test_indices <- fold_indices[[i]]
  train_indices <- unlist(fold_indices[-i]
  # Create training and testing datasets
  train_data <- wine_data[train_indices, ]
  test_data <- wine_data[test_indices, ]
```



7. Fit the decision tree model with Information Gain, Gini Index, Gain Ratio

```
  decision_tree_info_gain <- rpart(formula(paste(target_var, "~",
paste(features, collapse = "+"))),
                                   data = train_data,
                                   method = "class",
                                   parms = list(split = "information"))
  decision_tree_gini <- rpart(formula(paste(target_var, "~",
paste(features, collapse = "+"))),
                              data = train_data,
                              method = "class",
                              parms = list(split = "gini"))
   decision_tree_gain_ratio <- rpart(formula(paste(target_var, "~",
paste(features, collapse = "+"))),
                                    data = train_data,
                                    method = "class",
                                    parms = list(split = "gainratio"))
```

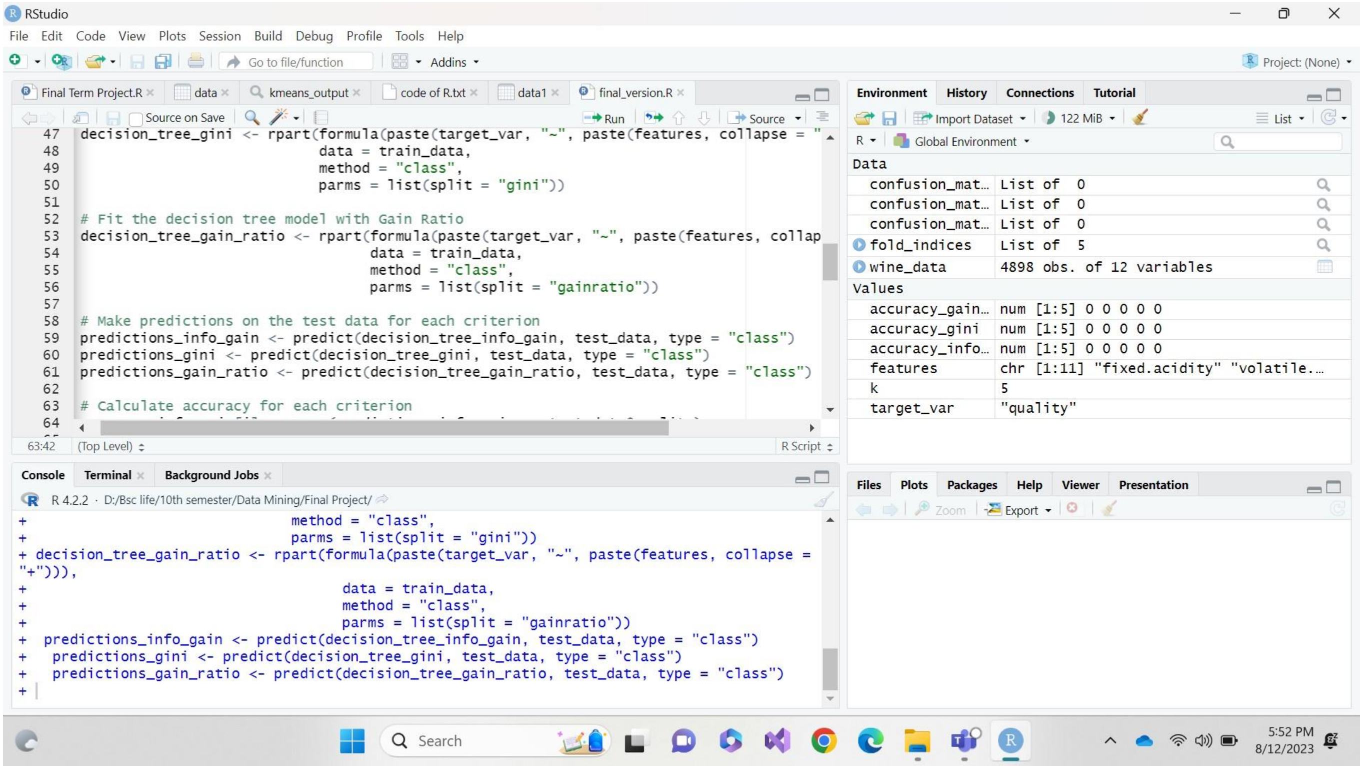


8. Make predictions on the test data for each criterion

predictions_info_gain <- predict(decision_tree_info_gain, test_data, type = "class")

predictions_gini <- predict(decision_tree_gini, test_data, type = "class")

predictions_gain_ratio <- predict(decision_tree_gain_ratio, test_data, type = "class")

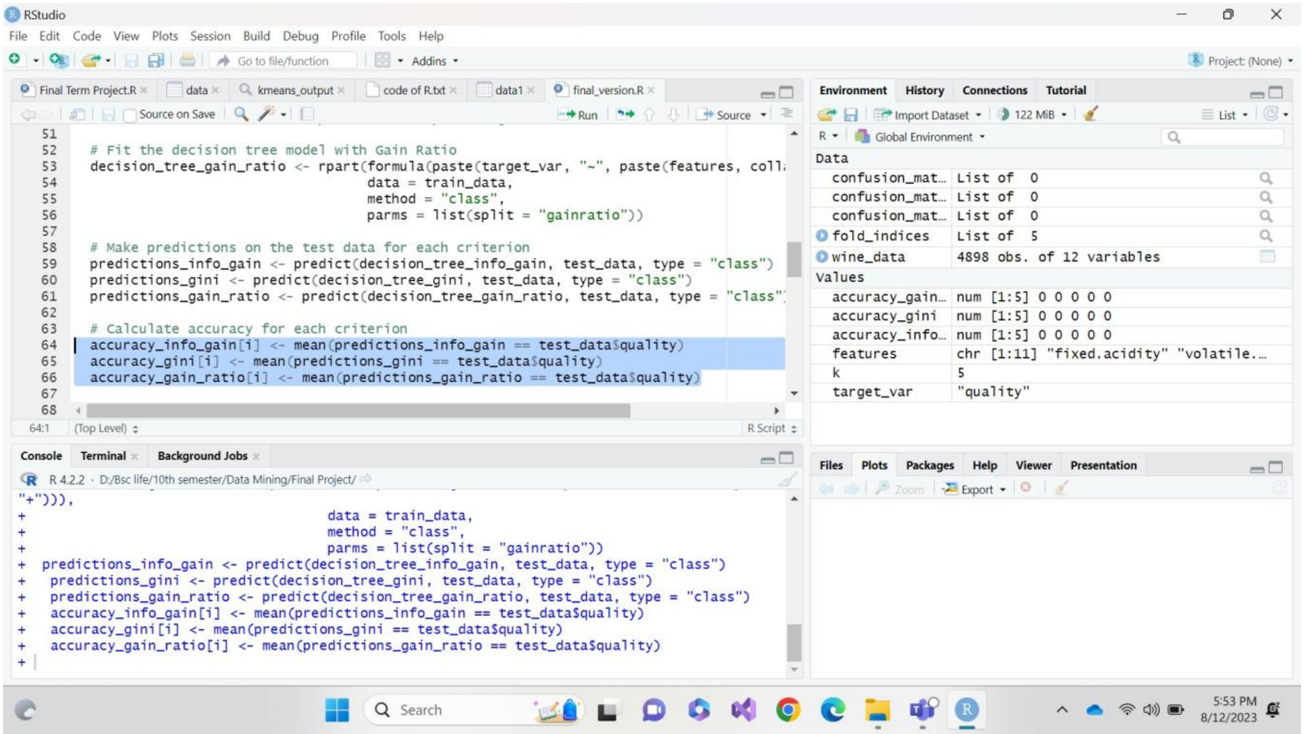


9. Calculate accuracy for each criterion

accuracy_info_gain[i] <- mean(predictions_info_gain == test_data$quality)

accuracy_gini[i] <- mean(predictions_gini == test_data$quality)

accuracy_gain_ratio[i] <- mean(predictions_gain_ratio == test_data$quality)



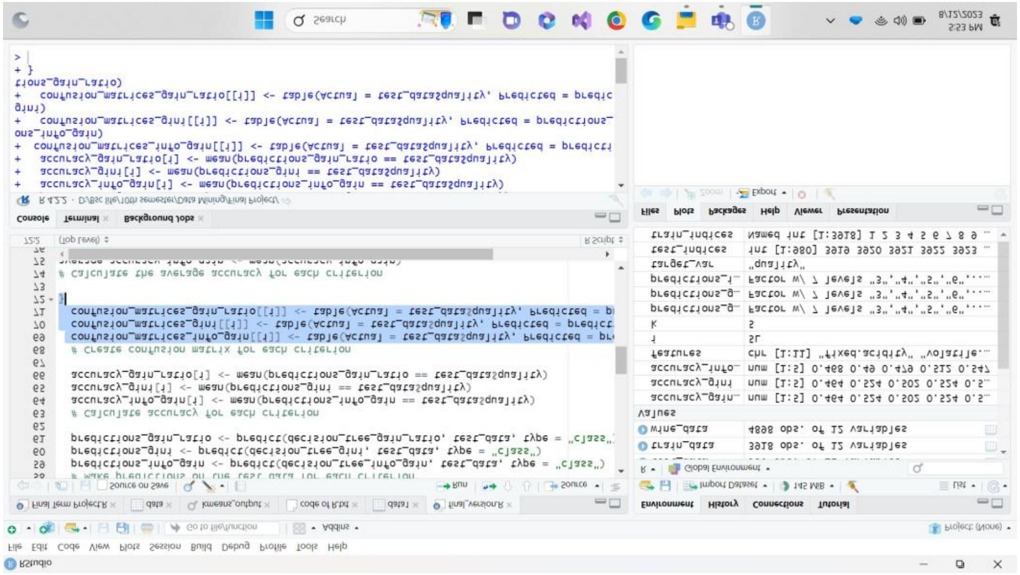
10. Create confusion matrix for each criterion

confusion_matrices_info_gain[[i]] <- table(Actual = test_data$quality, Predicted = predictions_info_gain)

confusion_matrices_gini[[i]] <- table(Actual = test_data$quality, Predicted = predictions_gini)

confusion_matrices_gain_ratio[[i]] <- table(Actual = test_data$quality, Predicted = predictions_gain_ratio)

}

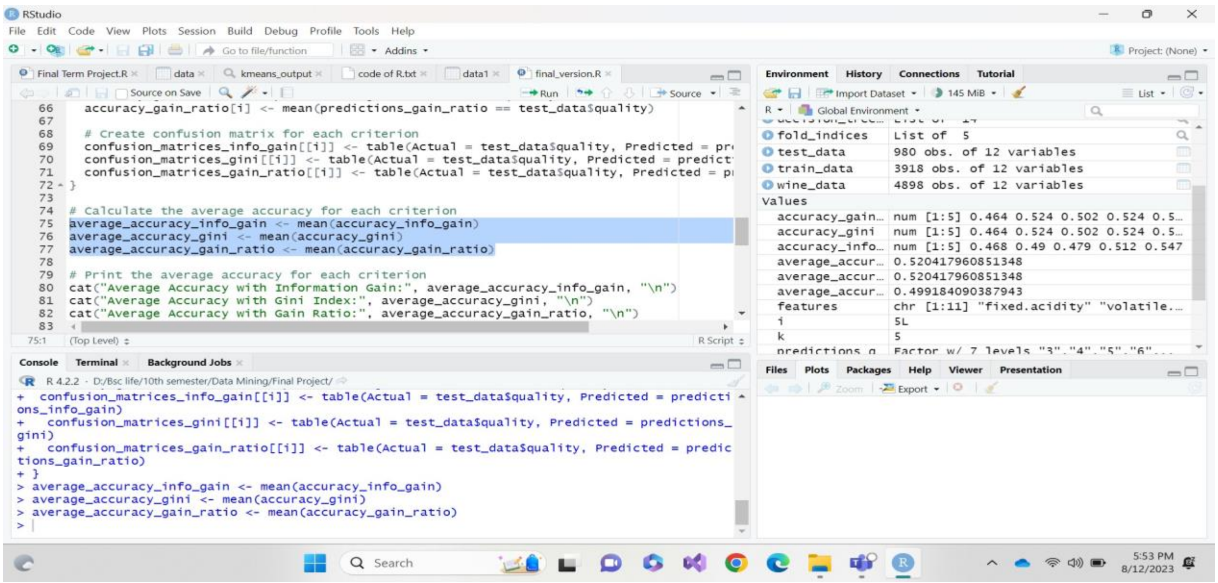


11. Calculate the average accuracy for each criterion

average_accuracy_info_gain <- mean(accuracy_info_gain)

average_accuracy_gini <- mean(accuracy_gini)

average_accuracy_gain_ratio <- mean(accuracy_gain_ratio)

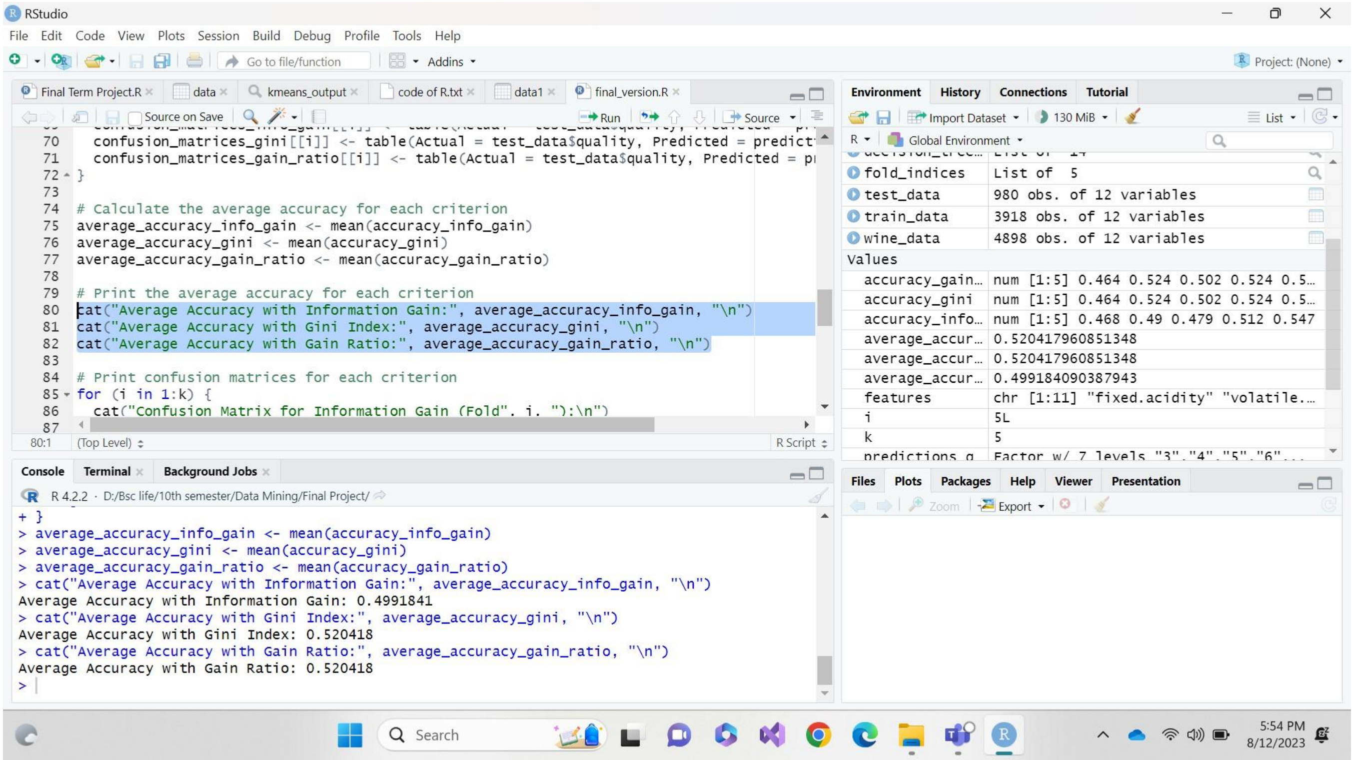


12. Print the average accuracy for each criterion

cat("Average Accuracy with Information Gain:", average_accuracy_info_gain, "\n")

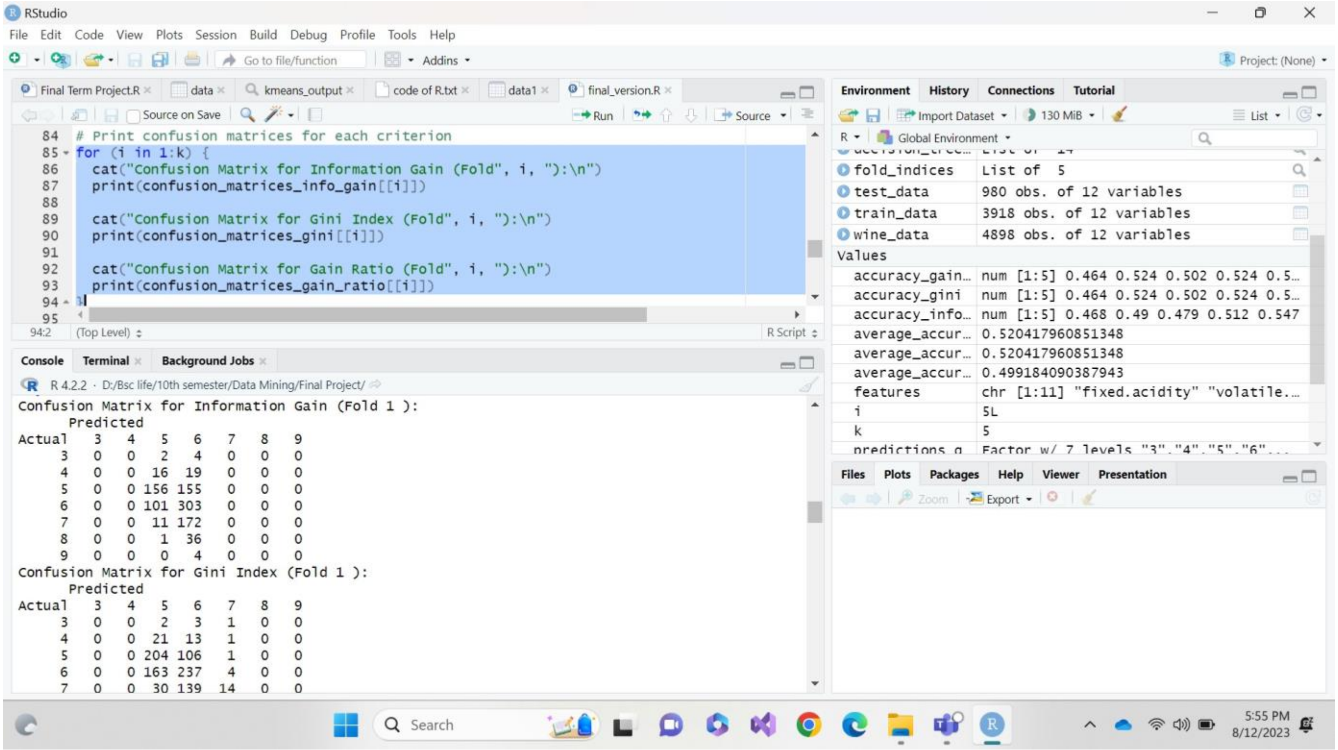
cat("Average Accuracy with Gini Index:", average_accuracy_gini, "\n")

cat("Average Accuracy with Gain Ratio:", average_accuracy_gain_ratio, "\n")



13. Print confusion matrices for each criterion

```
for (i in 1:k) {
  cat("Confusion Matrix for Information Gain (Fold", i, "):\n")
  print(confusion_matrices_info_gain[[i]])

  cat("Confusion Matrix for Gini Index (Fold", i, "):\n")
  print(confusion_matrices_gini[[i]])

  cat("Confusion Matrix for Gain Ratio (Fold", i, "):\n")
  print(confusion_matrices_gain_ratio[[i]])
}
```



14. Visualize decision trees using rpart.plot

#options(repr.plot.width = 1000, repr.plot.height = 500)

Visualize decision trees using rpart.plot with adjusted plot dimensions

options(repr.plot.width = 800, repr.plot.height = 400)

print("Decision Tree with Information Gain")

rpart.plot(decision_tree_info_gain)

print("Decision Tree with Gini Index:")

rpart.plot(decision_tree_gini)

print("Decision Tree with Gain Ratio:")

rpart.plot(decision_tree_gain_ratio)

print("Decision Tree with Information Gain")

rpart.plot(decision_tree_info_gain)

print("Decision Tree with Gini Index:")

rpart.plot(decision_tree_gini)

print("Decision Tree with Gain Ratio:")

rpart.plot(decision_tree_gain_ratio)