```r
# Alex Vo
# Assignment 3 - MATH 524
setwd('/Users/alexvo/Desktop/Education/SDSU/MATH-524 Linear Algebra/HW3')
getwd()
library(e1071)
library(dplyr)
library(ggplot2)
library(randomForest)
library(caret)
library(nnet)

##### 3.1
points <- data.frame(
  x = c(1, 2, 2, 3, 4),
  y = c(1, 2, 3, 4, 4)
)

set.seed(123)
kmeans_result <- kmeans(points, centers = 2)

# Calculate total within-cluster sum of squares (total)
total <- kmeans_result$tot.withinss

plot(points$x, points$y,
     col = kmeans_result$cluster,  # Color points by cluster
     pch = 16,                     # Solid circles for points
     cex = 2,                      # Size of points
     xlim = c(0, 5),              # X-axis limits
     ylim = c(0, 5),              # Y-axis limits
     xlab = "X",
     ylab = "Y",
     main = "K-means Clustering (K=2)")

# Add cluster centers
points(kmeans_result$centers,
       col = 1:2,                  # Colors matching the clusters
       pch = 8,                    # Star symbol for centers
       cex = 2)                    # Size of centers

# Add legend
legend("topleft",
       legend = c("Cluster 1", "Cluster 2", "Centroids"),
       col = c(1, 2, 1),
       pch = c(16, 16, 8))

cat("\nCluster Assignments:\n")
print(kmeans_result$cluster)
cat("\nTotal Within-Cluster Sum of Squares (total):", total, "\n")



##### 3.3
data <- read.csv("MiamiIntlAirport2001_2020.csv")
data$DATE <- as.Date(data$DATE) # Convert DATE column to Date type for filtering

# Filter TMIN and WDF2 for 2015
data_2015 <- data %>%
filter(format(DATE, "%Y") == "2015") %>%
select(TMIN, WDF2) %>%
na.omit()  # Remove rows with missing values

head(data_2015)

# Perform K-means clustering with K = 2
```

```
set.seed(123)
kmeans_result <- kmeans(data_2015, centers = 2)

# Calculate total within-cluster sum of squares (total)
total <- kmeans_result$tot.withinss

# Convert cluster centers to a data frame and set column names
centers <- as.data.frame(kmeans_result$centers)
colnames(centers) <- c("TMIN", "WDF2")

ggplot(data_2015, aes(x = TMIN, y = WDF2, color = Cluster)) +
  geom_point(size = 3) +
  geom_point(data = centers,
             aes(x = TMIN, y = WDF2), color = "red", size = 5, shape = 8) +
  labs(title = "K-means Clustering of TMIN and WDF2 (K=2)",
       x = "TMIN (Daily Minimum Temperature)",
       y = "WDF2 (Fastest 2-Minute Wind Direction)") +
  theme_minimal() + theme(legend.position = "top")

cat("Total Within-Cluster Sum of Squares (total):", total, "\n")



##### 3.8
svm_points <- data.frame(
  x = c(1, 2, 2, 3, 4),
  y = c(1, 2, 3, 4, 4)
)

# Assign classes based on y >= 3
svm_points$class <- factor(ifelse(svm_points$y >= 3, 1, -1))

# Train the SVM model with a linear kernel
svm_model <- svm(class ~ ., data = svm_points, kernel = "linear", cost = 1)

# Create a grid for decision boundary visualization
grid <- expand.grid(x = seq(0, 5, by = 0.1),
                    y = seq(0, 5, by = 0.1))
grid$pred <- predict(svm_model, grid)

# Extract support vectors
support_vectors <- as.data.frame(svm_model$SV)
colnames(support_vectors) <- c("x", "y")

# Plot
ggplot() +
  geom_tile(data = grid, aes(x = x, y = y, fill = pred), alpha = 0.2) +
  geom_point(data = svm_points, aes(x = x, y = y, color = class), size = 3) +
  geom_point(data = support_vectors, aes(x = x, y = y),
             color = "black", shape = 8, size = 4, stroke = 1.5) +
  scale_fill_manual(values = c("-1" = "blue", "1" = "red"),
                    name = "Prediction") +
  scale_color_manual(values = c("-1" = "blue", "1" = "red"),
                     name = "Class") +
  labs(title = "SVM Analysis with Decision Boundary and Support Vectors",
       x = "X Coordinate",
       y = "Y Coordinate") +
  theme_minimal() + theme(legend.position = "top")

# Print support vectors
cat("Support Vectors:\n")
print(support_vectors)
```

```
##### 3.9
# Create new points for prediction
new_points <- data.frame(
  x = c(1.5, 3),
  y = c(1, 3)
)

predictions <- predict(svm_model, new_points)
new_points$class <- predictions

# Create a grid for decision boundary visualization
grid <- expand.grid(x = seq(0, 5, by = 0.1),
                    y = seq(0, 5, by = 0.1))
grid$pred <- predict(svm_model, grid)

# Plot
ggplot() +
  geom_tile(data = grid, aes(x = x, y = y, fill = pred), alpha = 0.2) +
  geom_point(data = svm_points, aes(x = x, y = y, color = class), size = 3) +
  geom_point(data = new_points, aes(x = x, y = y, color = class),
             size = 5, shape = 17, stroke = 1.5) +  # Triangle shape with outline
  geom_text(data = new_points, aes(x = x, y = y, label = paste0("Q", 1:nrow(new_points))),
            vjust = -1.5, size = 5, fontface = "bold") +
  scale_fill_manual(values = c("-1" = "blue", "1" = "red"),
                    name = "Prediction") +
  scale_color_manual(values = c("-1" = "blue", "1" = "red"),
                     name = "Class") +
  labs(title = "SVM Classification with New Points",
       x = "X Coordinate",
       y = "Y Coordinate",
       subtitle = "Original points (circles), New points (triangles with labels)") +
  theme_minimal() +
  theme(legend.position = "top",
        plot.subtitle = element_text(size = 12, face = "italic"))

cat("\nClassifications for new points:\n")
for (i in 1:nrow(new_points)) { cat(sprintf("Q%d(%0.1f, %0.1f): %s\n", i, new_points$x[i],
new_points$y[i], as.character(new_points$class[i]))) }




##### 3.13
data <- read.csv("iris.csv")
# Create training dataset
training_data <- rbind(
  iris[1:40, ],          # First 40 setosa
  iris[51:90, ],         # First 40 versicolor
  iris[101:140, ]        # First 40 virginica
)

# Create testing dataset
testing_data <- rbind(
  iris[41:50, ],         # Remaining 10 setosa
  iris[91:100, ],        # Remaining 10 versicolor
  iris[141:150, ]        # Remaining 10 virginica
)

# Train Random Forest model
set.seed(123)
rf_model <- randomForest(
  Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
  data = training_data,
  ntree = 500,
  importance = TRUE
```

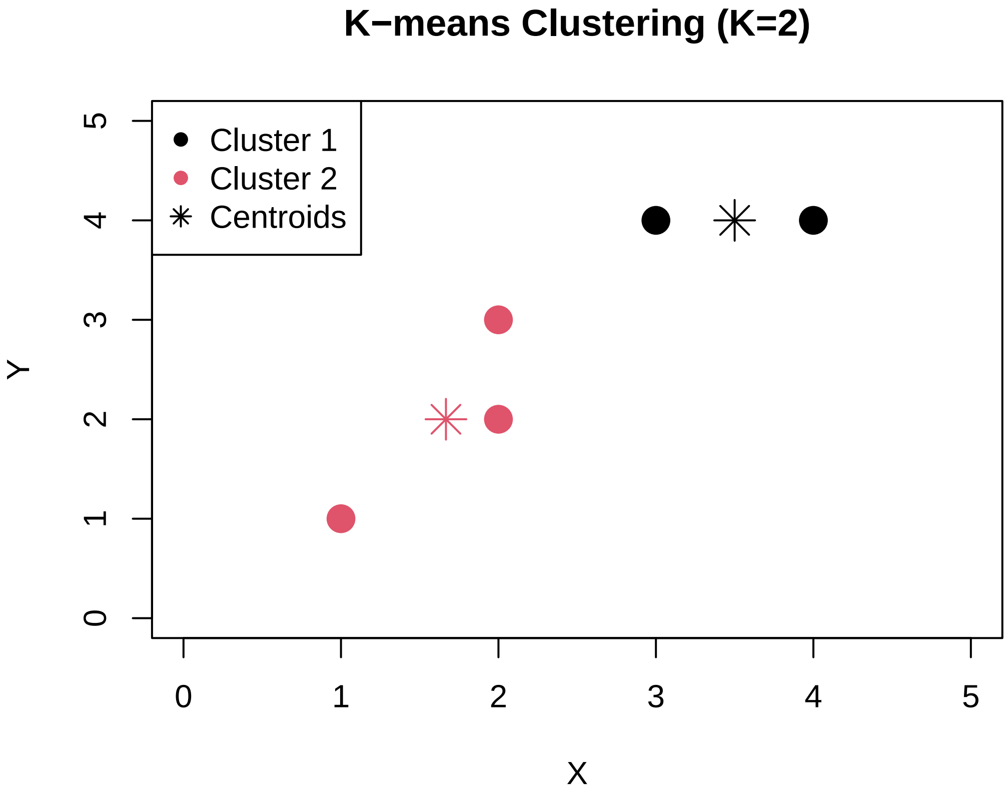
```
)

# Make predictions on test data
predictions <- predict(rf_model, testing_data)

# Create confusion matrix
conf_matrix <- table(predictions, testing_data$Species)

# Calculate accuracy
accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)

# Print results
cat("\nRandom Forest Model Results:\n")
cat("\nConfusion Matrix:\n")
print(conf_matrix)
cat("\nAccuracy:", round(accuracy * 100, 2), "%\n")

# Variable importance
cat("\nVariable Importance:\n")
print(importance(rf_model))

# Plot variable importance
varImpPlot(rf_model, main = "Variable Importance Plot")

# Create detailed results
results_df <- cbind(testing_data, Predicted = predictions)
cat("\nDetailed Predictions:\n")
print(results_df)




##### 3.14
# Set parameters for sampling
set.seed(123)
n_train <- 10  # 20% per species (10 samples)
n_test <- 5    # 10% per species (5 samples)

# Helper function to sample indices
sample_indices <- function(start, end, n) sort(sample(start:end, n))

# Generate training and testing datasets
train_indices <- unlist(lapply(seq(1, 150, by = 50), function(start) sample_indices(start,
start + 49, n_train)))
test_indices <- unlist(lapply(seq(1, 150, by = 50), function(start) setdiff(start:(start +
49), train_indices[train_indices %in% start:(start + 49)])[1:n_test]))

training_data <- iris[train_indices, ]
testing_data <- iris[test_indices, ]

# Train Random Forest model
rf_model <- randomForest(Species ~ ., data = training_data, ntree = 500, importance =
TRUE)

# Make predictions and compute confusion matrix
predictions <- predict(rf_model, testing_data)
conf_matrix <- table(Predicted = predictions, Actual = testing_data$Species)

# Calculate metrics
accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
error_rate <- 1 - accuracy
class_metrics <- data.frame(
  Species = levels(iris$Species),
  Precision = diag(conf_matrix) / colSums(conf_matrix),
  Recall = diag(conf_matrix) / rowSums(conf_matrix)
```

```
)
class_metrics$F1 <- with(class_metrics, 2 * (Precision * Recall) / (Precision + Recall))

# Print results
cat("\nRandom Forest Model Results:\n")
cat(sprintf("Training Data Size: %d samples (20%% per species)\n", nrow(training_data)))
cat(sprintf("Testing Data Size: %d samples (10%% per species)\n", nrow(testing_data)))
cat("\nConfusion Matrix:\n")
print(conf_matrix)
cat(sprintf("\nOverall Accuracy: %.2f%%\n", accuracy * 100))
cat(sprintf("Error Rate: %.2f%%\n", error_rate * 100))
cat("\nPer-Class Metrics:\n")
print(class_metrics)

# Print variable importance
cat("\nVariable Importance:\n")
print(importance(rf_model))

# Detailed results
results_df <- cbind(testing_data, Predicted = predictions, Correct = testing_data$Species
== predictions)
cat("\nDetailed Prediction Results:\n")
print(results_df)




##### 3.19
# Scale the features for better neural network performance
# Create scaling parameters from training data
scale_params <- preProcess(training_data[, 1:4], method = c("center", "scale"))
scaled_training <- predict(scale_params, training_data[, 1:4])
scaled_testing <- predict(scale_params, testing_data[, 1:4])

# Convert species to dummy variables for training
species_dummy <- class.ind(training_data$Species)

# Train Neural Network model
set.seed(123)
nn_model <- nnet(
  x = scaled_training,
  y = species_dummy,
  size = 5,       # Number of hidden nodes
  maxit = 1000,   # Maximum iterations
  decay = 0.01,   # Weight decay for regularization
  trace = FALSE   # Suppress training output
)

# Make predictions on test data
predictions_prob <- predict(nn_model, scaled_testing)
predictions <- factor(max.col(predictions_prob), labels = levels(iris$Species))

# Create confusion matrix and calculate accuracy
conf_matrix <- table(Predicted = predictions, Actual = testing_data$Species)
accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)

# Print model structure and results
cat("\nNeural Network Model Results:\n")
cat("\nModel Structure:\n")
cat("  - Input Layer: 4 nodes (4 features)\n")
cat("  - Hidden Layer: 5 nodes\n")
cat("  - Output Layer: 3 nodes (3 species)\n")
cat("\nConfusion Matrix:\n")
print(conf_matrix)
cat("\nAccuracy:", round(accuracy * 100, 2), "%\n")
```

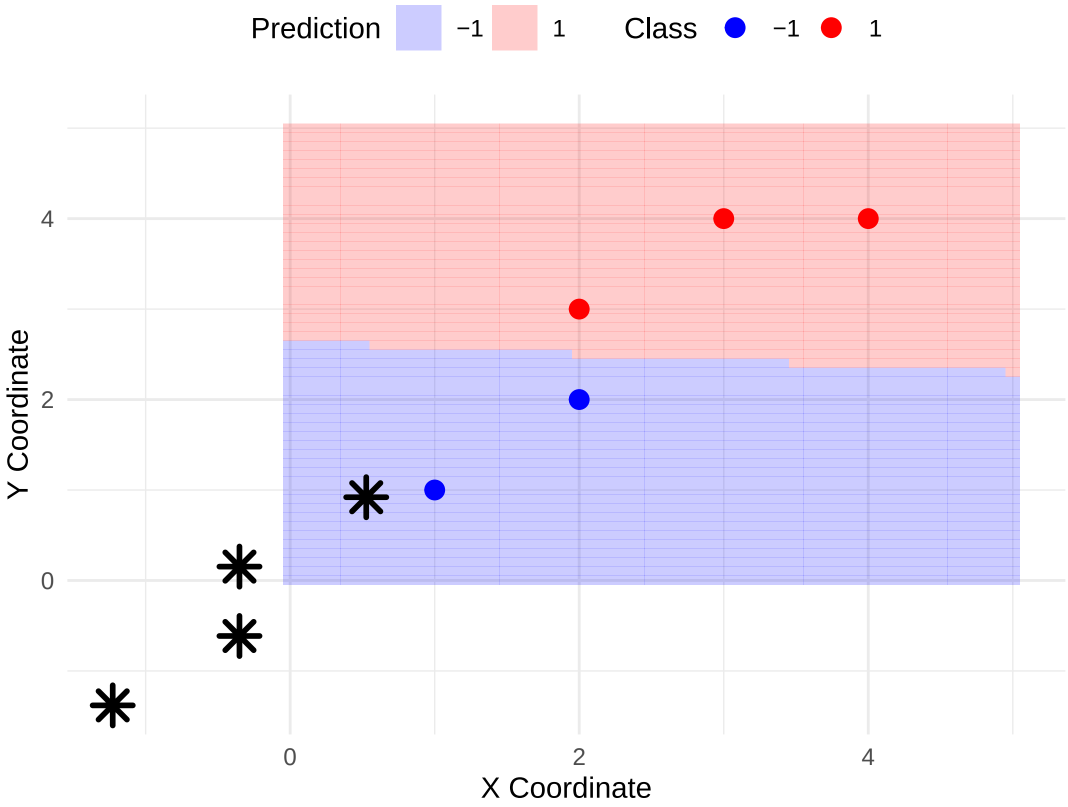
```
# Per-class metrics
precision <- diag(conf_matrix) / colSums(conf_matrix)
recall <- diag(conf_matrix) / rowSums(conf_matrix)
f1_score <- 2 * (precision * recall) / (precision + recall)

cat("\nPer-Class Performance Metrics:\n")
metrics_df <- data.frame(
  Species = levels(iris$Species),
  Precision = round(precision, 3),
  Recall = round(recall, 3),
  F1_Score = round(f1_score, 3)
)
print(metrics_df)

results_df <- cbind(testing_data, Predicted = predictions, Correct = testing_data$Species
== predictions)
cat("\nDetailed Predictions:\n")
print(results_df)




##### 3.20
# Split data into training and testing sets
train_indices <- unlist(lapply(seq(1, 150, by = 50), function(start) sample_indices(start,
start + 49, n_train)))
test_indices <- unlist(lapply(seq(1, 150, by = 50), function(start) setdiff(start:(start +
49), train_indices[train_indices %in% start:(start + 49)])[1:n_test]))

training_data <- iris[train_indices, ]
testing_data <- iris[test_indices, ]

# Scale features
scale_params <- preProcess(training_data[, 1:4], method = c("center", "scale"))
scaled_training <- predict(scale_params, training_data[, 1:4])
scaled_testing <- predict(scale_params, testing_data[, 1:4])

# Convert species to dummy variables
species_dummy <- class.ind(training_data$Species)

# Train Neural Network model
nn_model <- nnet(
  x = scaled_training,
  y = species_dummy,
  size = 4,       # Hidden nodes
  maxit = 1000,   # Maximum iterations
  decay = 0.1,    # Regularization
  trace = FALSE   # Suppress training output
)

# Make predictions
predictions_prob <- predict(nn_model, scaled_testing)
predictions <- factor(max.col(predictions_prob), labels = levels(iris$Species))

# Evaluate model
conf_matrix <- table(Predicted = predictions, Actual = testing_data$Species)
accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
error_rate <- 1 - accuracy

# Calculate metrics
precision <- diag(conf_matrix) / colSums(conf_matrix)
recall <- diag(conf_matrix) / rowSums(conf_matrix)
f1_score <- 2 * (precision * recall) / (precision + recall)
```

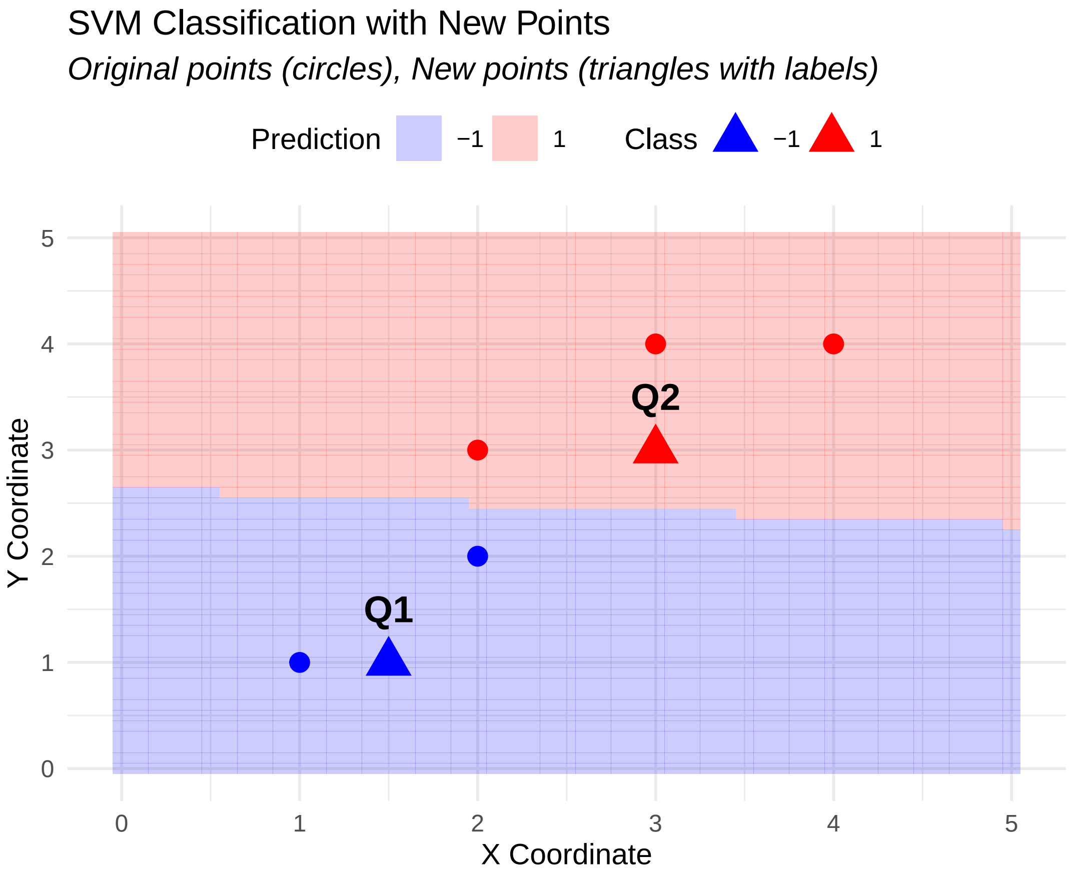
```
# Print results
cat("\nNeural Network Model Results:\n")
cat("\nTraining Data Size:", nrow(training_data), "samples (20% per species)")
cat("\nTesting Data Size:", nrow(testing_data), "samples (10% per species)")
cat("\n\nConfusion Matrix:\n")
print(conf_matrix)
cat("\nOverall Accuracy:", round(accuracy * 100, 2), "%")
cat("\nError Rate:", round(error_rate * 100, 2), "%\n")

cat("\nPer-Class Performance Metrics:\n")
metrics_df <- data.frame(
  Species = levels(iris$Species),
  Precision = round(precision, 3),
  Recall = round(recall, 3),
  F1_Score = round(f1_score, 3)
)
print(metrics_df)

# Detailed results
results_df <- cbind(testing_data, Predicted = predictions, Correct = testing_data$Species
== predictions)
cat("\nDetailed Prediction Results:\n")
print(results_df)




##### 4.11
##### (a) 12th Order Polynomial Fitting #####
# Read and process the data
data <- read.csv("NOAAGlobalT.csv", check.names = FALSE)
temps <- as.numeric(data[1, ])  # Extract temperatures
years <- 1880:2018             # Years from 1880 to 2018

# Create data frame
data <- data.frame(
  year = years,
  temp = temps
)

# Fit a 12th-order polynomial using linear regression
fit <- lm(temp ~ poly(year, 12, raw = TRUE), data = data)

# Display coefficients
cat("Polynomial Coefficients:\n")
coefficients <- coef(fit)
print(coefficients)

##### (b) Plot Data and Fitted Function #####
# Generate points for the fitted curve
years_seq <- seq(min(data$year), max(data$year), length.out = 500)
predicted_temps <- predict(fit, newdata = data.frame(year = years_seq))

# Plot observed data and fitted curve
library(ggplot2)
p1 <- ggplot() +
  geom_point(data = data, aes(x = year, y = temp), color = "blue", size = 2) +
  geom_line(aes(x = years_seq, y = predicted_temps), color = "red", size = 1) +
  labs(title = "12th Order Polynomial Fit", x = "Year", y = "Temperature Anomaly (°C)") +
  theme_minimal()
print(p1)

##### (c) Residuals Scatter Plot #####
# Residuals
residuals <- residuals(fit)
```

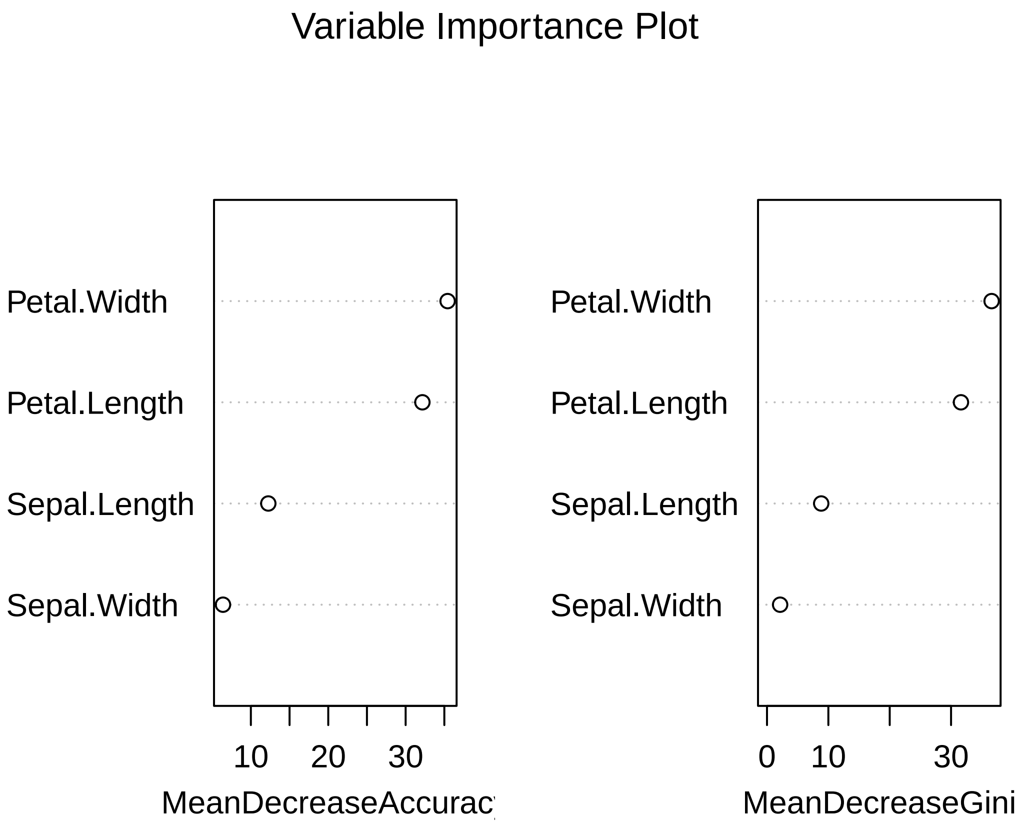
```r
# Plot residuals
p2 <- ggplot() +
  geom_point(aes(x = data$year, y = residuals), color = "purple", size = 2) +
  geom_hline(yintercept = 0, linetype = "dashed", color = "black") +
  labs(title = "Residuals of Polynomial Fit", x = "Year", y = "Residuals (°C)") +
  theme_minimal()
print(p2)

# Summary statistics
cat("R-squared:", summary(fit)$r.squared, "\n")
cat("RMSE:", sqrt(mean(residuals^2)), "\n")
```


K−means Clustering (K=2)
Cluster 1
Cluster 2
Centroids
X
Y
0
1
2
3
4
5
0
1
2
3
4
5

# SVM Analysis with Decision Boundary and Support Vectors




SVM Classification with New Points
Original points (circles), New points (triangles with labels)
Prediction
−1
1
Class
−1
1
Q1
Q2
Y Coordinate
X Coordinate
0
1
2
3
4
5


Variable Importance Plot
Petal.Width
Petal.Length
Sepal.Length
Sepal.Width
10
20
30
MeanDecreaseAccurac
Petal.Width
Petal.Length
Sepal.Length
Sepal.Width
0
10
30
MeanDecreaseGini