```r
# importing the necessary libraries and suppressing any possible warnings
suppressWarnings(library(mice))
suppressWarnings(library(MASS))
suppressWarnings(library(caret))
suppressWarnings(library(matlib))
suppressWarnings(library(Matrix))
suppressWarnings(library(ggplot2))
suppressWarnings(library(dplyr))
suppressWarnings(library(DescTools))
suppressWarnings(library(yardstick))
suppressWarnings(library(trainsplit))
suppressWarnings(library(glmnet))

main_data <- read.csv("C://Users/Hareen/Desktop/5303/cleaned_5250.csv")

main_data <- main_data %>%
  mutate(mass = case_when(
    mass_wrt == "Jupiter" ~ 1.899e+27 * mass_multiplier,  # Jupiter mass
    mass_wrt == "Earth" ~ 5.9722e+24 * mass_multiplier,   # Earth mass
    TRUE ~ NA_real_  # Handle any other cases
  )) %>%
  filter(!is.na(mass))  # Remove rows with NA in the radius column

main_data <- main_data %>%
  mutate(radius = case_when(
    radius_wrt == "Jupiter" ~ 43441 * radius_multiplier,  # Jupiter radius
    radius_wrt == "Earth" ~ 3963.1 * radius_multiplier,   # Earth radius
    TRUE ~ NA_real_  # Handle any other cases
  )) %>%
  filter(!is.na(radius))  # Remove rows with NA in the radius column

summary(main_data)

main_data <- na.omit(main_data)
summary(main_data)

# Set a seed for reproducibility
set.seed(123)
# Select random 1000 rows from the dataset
data <- main_data[sample(nrow(main_data), 1000), ]

data_standarized <- data %>%
  mutate(across(where(is.numeric), ~ scale(.)))  # Standardize only numeric columns

attach(data_standarized)

# Density curve of distance
ggplot(data_standarized, aes(x = distance)) +
  geom_density(color = "blue", fill = "lightblue", alpha = 0.5) +
  labs(title = "Density Curve of Distance", x = "Distance", y = "Density") +
  theme_minimal()

# Density curve of stellar_magnitude
ggplot(data_standarized, aes(x = stellar_magnitude)) +
  geom_density(color = "blue", fill = "lightblue", alpha = 0.5) +
  labs(title = "Density Curve of Stellar Magnitude", x = "Stellar Magnitude", y =
"Density") +
  theme_minimal()

# Density curve with discovery year
ggplot(data_standarized, aes(x = discovery_year)) +
  geom_density(color = "blue", fill = "lightblue", alpha = 0.5) +
  labs(title = "Density Curve of Discovery Year", x = "Discovery Year", y = "Density") +
  theme_minimal()
```

```r
# Density curve with radius
ggplot(data_standarized, aes(x = radius)) +
  geom_density(color = "blue", fill = "lightblue", alpha = 0.5) +
  labs(title = "Density Curve of Radius", x = "Radius", y = "Density") +
  theme_minimal()

# Density curve with mass
ggplot(data_standarized, aes(x = mass)) +
  geom_density(color = "blue", fill = "lightblue", alpha = 0.5) +
  labs(title = "Density Curve of Mass", x = "Mass", y = "Density") +
  scale_x_continuous(limits = c(0, 12)) +
  theme_minimal()

# Density curve with orbital radius
ggplot(data_standarized, aes(x = orbital_radius)) +
  geom_density(color = "blue", fill = "lightblue", alpha = 0.5) +
  labs(title = "Density Curve of Orbital Radius", x = "Orbital Radius", y = "Density") +
  scale_x_continuous(limits = c(0, 28)) +
  theme_minimal()

# Density curve with orbital period
ggplot(data_standarized, aes(x = orbital_period)) +
  geom_density(color = "blue", fill = "lightblue", alpha = 0.5) +
  labs(title = "Density Curve of Orbital Period", x = "Orbital Period", y = "Density") +
  scale_x_continuous(limits = c(0, 29)) +
  theme_minimal()

# Density curve with eccentricity
ggplot(data_standarized, aes(x = eccentricity)) +
  geom_density(color = "blue", fill = "lightblue", alpha = 0.5) +
  labs(title = "Density Curve of Eccentricity", x = "Eccentricity", y = "Density") +
  theme_minimal()

dist_cont <- table(planet_type, distance)
Assocs(dist_cont) # 0.8476

stellar_cont <- table(planet_type, stellar_magnitude)
Assocs(stellar_cont) # 0.8535

disc_cont <- table(planet_type, discovery_year)
Assocs(disc_cont) # 0.5109

mass_mult_cont <- table(planet_type, mass_multiplier)
Assocs(mass_mult_cont) # 0.8520

mass_wrt_cont <- table(planet_type, mass_wrt)
Assocs(mass_wrt_cont) # 0.6996

mass_cont <- table(planet_type, mass)
Assocs(mass_cont) # 0.8617

radius_mult_cont <- table(planet_type, radius_multiplier)
Assocs(radius_mult_cont) # 0.8378

radius_wrt_cont <- table(planet_type, radius_wrt)
Assocs(radius_wrt_cont) # 0.6643

radius_cont <- table(planet_type, radius)
Assocs(radius_cont) # 0.8613

orb_radius_cont <- table(planet_type, orbital_radius)
Assocs(orb_radius_cont) # 0.8446

orb_period_cont <- table(planet_type, orbital_period)
```

```
Assocs(orb_period_cont) # 0.7646

ecc_cont <- table(planet_type, eccentricity)
Assocs(ecc_cont) # 0.5426

# Splitting the data into training and testing sets
set.seed(123)
train_index <- createDataPartition(planet_type, p=0.7, list=FALSE)
train_data <- data_standarized[train_index,]
test_data <- data_standarized[-train_index,]

# Fit the LDA model on the training data
lda_model <- lda(planet_type ~ as.vector(mass) + as.vector(radius), data = train_data)

# Print the model summary
print(lda_model)

# Make predictions on the test data
predictions <- predict(lda_model, newdata = test_data)

# Create a data frame for evaluation
results <- data.frame(
  actual = test_data$planet_type,
  predicted = predictions$class
)

# Convert actual and predicted to factors
results$actual <- as.factor(results$actual)
results$predicted <- as.factor(results$predicted)

# Convert to a tibble for easier manipulation
results <- as_tibble(results)

# Calculate confusion matrix
confusion_matrix <- conf_mat(results, truth = actual, estimate = predicted)

# Calculate accuracy
accuracy <- accuracy_vec(truth = results$actual, estimate = results$predicted)

# Calculate precision, recall, and F1-score
precision <- precision_vec(truth = results$actual, estimate = results$predicted)
recall <- recall_vec(truth = results$actual, estimate = results$predicted)
f1 <- f_meas_vec(truth = results$actual, estimate = results$predicted)

# Print the metrics
cat("\n")
print(paste("Precision:", precision))
print(paste("Recall:", recall))
print(paste("F1 Score:", f1))
print(paste("Accuracy:", accuracy))

# Visualize confusion matrix with a red gradient heatmap
confusion_matrix_plot <- autoplot(confusion_matrix, type = "heatmap") +
  scale_fill_gradient(
    low = "white",  # Start of the gradient
    high = "red"    # End of the gradient
  ) +
  labs(
    title = "Confusion Matrix Heatmap",
    x = "Predicted Class",
    y = "Actual Class",
    fill = "Count"
  ) +
  theme_minimal()
```

```
print(confusion_matrix_plot)

# Convert Class to a factor for LDA
data_standarized$planet_type <- as.factor(data_standarized$planet_type)

# Fit the LDA model
lda_model <- lda(planet_type ~ as.vector(mass) + as.vector(radius), data = train_data)

# Create a grid for prediction
x_min <- min(data_standarized$mass) - 1
x_max <- max(data_standarized$mass) + 1
y_min <- min(data_standarized$radius) - 1
y_max <- max(data_standarized$radius) + 1
grid <- expand.grid(mass = seq(x_min, x_max, length = 200),
                    radius = seq(y_min, y_max, length = 200))

# Predict the class for each point in the grid
lda_pred <- predict(lda_model, grid)$class
grid$planet_type <- lda_pred

# Plotting the LDA decision boundaries for all classes
lda_plot <- ggplot(data_standarized, aes(x = mass, y = radius, color = planet_type)) +
  geom_point(size = 3) +
  geom_tile(data = grid, aes(fill = planet_type), alpha = 0.3) +  # Fill the grid with
predicted classes
  labs(title = "LDA Decision Boundary", x = "Mass", y = "Radius") +
  theme_minimal() +
  scale_fill_manual(values = c("Terrestrial" = "blue", "Gas Giant" = "red", "Ice Giant" =
"green")) +
  theme(legend.title = element_blank())

# Print the LDA plot
print(lda_plot)

# Fit the LDA model on the training data
qda_model <- qda(planet_type ~ as.vector(mass) + as.vector(radius), data = train_data)

# Print the model summary
print(qda_model)

# Make predictions on the test data
predictions <- predict(qda_model, newdata = test_data)

# Create a data frame for evaluation
results <- data.frame(
  actual = test_data$planet_type,
  predicted = predictions$class
)

# Convert actual and predicted to factors
results$actual <- as.factor(results$actual)
results$predicted <- as.factor(results$predicted)

# Convert to a tibble for easier manipulation
results <- as_tibble(results)

# Calculate confusion matrix
confusion_matrix <- conf_mat(results, truth = actual, estimate = predicted)

# Print confusion matrix
print(confusion_matrix)

# Calculate accuracy
accuracy <- accuracy_vec(truth = results$actual, estimate = results$predicted)
```

```r
# Calculate precision, recall, and F1-score
precision <- precision_vec(truth = results$actual, estimate = results$predicted)
recall <- recall_vec(truth = results$actual, estimate = results$predicted)
f1 <- f_meas_vec(truth = results$actual, estimate = results$predicted)

# Print the metrics
print(paste("Precision:", precision))
print(paste("Recall:", recall))
print(paste("F1 Score:", f1))
print(paste("Accuracy:", accuracy))

# Visualize confusion matrix with a red gradient heatmap
confusion_matrix_plot <- autoplot(confusion_matrix, type = "heatmap") +
  scale_fill_gradient(
    low = "white",  # Start of the gradient
    high = "red"    # End of the gradient
  ) +
  labs(
    title = "Confusion Matrix Heatmap",
    x = "Predicted Class",
    y = "Actual Class",
    fill = "Count"
  ) +
  theme_minimal()

print(confusion_matrix_plot)

# Convert Class to a factor for LDA
data_standarized$planet_type <- as.factor(data_standarized$planet_type)

# Fit the LDA model
qda_model <- qda(planet_type ~ as.vector(mass) + as.vector(radius), data = train_data)

# Create a grid for prediction
x_min <- min(data_standarized$mass) - 1
x_max <- max(data_standarized$mass) + 1
y_min <- min(data_standarized$radius) - 1
y_max <- max(data_standarized$radius) + 1
grid <- expand.grid(mass = seq(x_min, x_max, length = 200),
                    radius = seq(y_min, y_max, length = 200))

# Predict the class for each point in the grid
qda_pred <- predict(qda_model, grid)$class
grid$planet_type <- qda_pred

# Plotting the QDA decision boundaries for all classes
qda_plot <- ggplot(data_standarized, aes(x = mass, y = radius, color = planet_type)) +
  geom_point(size = 3) +
  geom_tile(data = grid, aes(fill = planet_type), alpha = 0.3) +  # Fill the grid with
predicted classes
  labs(title = "QDA Decision Boundary", x = "Mass", y = "Radius") +
  theme_minimal() +
  scale_fill_manual(values = c("Terrestrial" = "blue", "Gas Giant" = "red", "Ice Giant" =
"green")) +
  theme(legend.title = element_blank())

# Print the LDA plot
print(qda_plot)
```