Applied Machine Learning

Student Name: Abdulrahman Alhariri
Student ID: C2644873

Project Title: Predicting Incident Severity Using Random Forest

---

Table of Contents

1. Introduction
2. Data Preprocessing and Exploration
3. Model Implementation
4. Results
5. Conclusion
6. Code and Experiment Screenshots

---

1. Introduction

This project focuses on predicting the severity of gun violence incidents using machine learning techniques, specifically a Random Forest model. The dataset contains features such as state, number of guns involved, and incident severity.

---

2. Data Preprocessing

step 1: Load Libraries & Data

1. **Libraries Used:**
    - **Load necessary libraries for data manipulation and machine learning.**

```
# ---- Step 1: Load Libraries & Data
library(dplyr)
library(lubridate)
library(tidyr)
library(caret)
```

2. **Data Loading:**

- **Reads the dataset from the specified path.**

```
gun_data <- read.csv("/Gun_violence.csv", stringsAsFactors = FALSE)
```

Step 2: Preprocess Data

1. **Convert Date Column:**

   Converts the date column to a Date object for easier handling.

```
gun_data$date <- dmy(gun_data$date)
```

2. **Handle Missing Values:**

   Replace empty strings with NA.

```
gun_data[gun_data == ""] <- NA
```

3. **Drop Unnecessary Columns:**

   Removes irrelevant columns.

```
# Remove unnecessary columns from the data set
gun_data <- gun_data %>%
 select(
  -address, -incident_url, -source_url, -congressional_district,
  -incident_url_fields_missing, -notes, -participant_name, -incident_id,
  -participant_relationship, -state_house_district, -state_senate_district,
  -sources, -latitude, -longitude, -location_description, -date, -city_or_county,
  -incident_characteristics, -participant_age, -participant_age_group,
  -participant_status, -participant_gender, -participant_type
 )
```

4. **Summarize Missing Values:**

   Check NA values per column

```
colSums(is.na(gun_data))
```

5. **Drop Critical Missing Data Rows:**

   Removes rows with NA in key columns.

```
gun_data <- gun_data %>% drop_na(n_guns_involved, gun_stolen)
```

Step 3: Encode Categorical Variables

1. **Encode state as Factor:**

   Convert the state column to a factor.

```
gun_data$state <- as.factor(gun_data$state)
```

2. **Gun Type Mapping:**

   Define a mapping for gun types.

```
gun_type_map <- list(
  Unknown_type = c("Unknown", ""),
  Pistol = c("Handgun", "9mm", "45 Auto", "40 SW", "44 Mag", "38 Spl", "380 Auto", "32 Auto", "357 Mag", "25 Auto", "10mm"),
  Shotgun = c("Shotgun", "12 gauge", "410 gauge", "16 gauge", "20 gauge", "28 gauge"),
  Rifle = c("22 LR", "223 Rem [AR-15]", "7.62 [AK-47]", "308 win", "Rifle", "30-30 Win", "30-06 Spr", "300 Win"),
  Other = c("Other")
)
```

3. **Process gun_type Data:**

   Clean and split the gun_type column.

```
gun_data$gun_type_cleaned <- gsub("[0-9]+::", "", gun_data$gun_type)
gun_data$gun_type_cleaned <- gsub("\\|", "||", gun_data$gun_type_cleaned)
gun_data <- gun_data[!grepl("^[0-9]+:", gun_data$gun_type_cleaned), ]
gun_data$gun_type_split <- strsplit(as.character(gun_data$gun_type_cleaned), "\\|\\|")
```

4. **Categorize Gun Types:**

   Assign categories to gun types.

```
gun_data$gun_type_category <- lapply(gun_data$gun_type_split, function(types) {
  categories <- unlist(lapply(types, function(type) {
    sapply(names(gun_type_map), function(category) if(type %in% gun_type_map[[category]])
category else NULL)
  }))
  unique(categories[!is.null(categories)])
})
```

5. **Binary Matrix for Gun Types:**

   Create a binary matrix for gun type categories.

```
gun_data_categories <- do.call(rbind, lapply(gun_data$gun_type_category, function(categories) {
  as.integer(names(gun_type_map) %in% categories)
}))
colnames(gun_data_categories) <- names(gun_type_map)
gun_data_categories <- as.data.frame(gun_data_categories)
```

6. **Process gun_stolen Data:**

   Clean and process gun_stolen.

```
stolen_categories <- c("Unknown", "Stolen", "Not-stolen")
# Process 'gun_stolen' by cleaning up the data
gun_data$gun_stolen_cleaned <- gsub("[0-9]+::", "", gun_data$gun_stolen)
gun_data$gun_stolen_cleaned <- gsub("\\|", "||", gun_data$gun_stolen_cleaned)
gun_data <- gun_data[!grepl("^[0-9]+:", gun_data$gun_stolen_cleaned), ]

# Replace empty strings with "Unknown" and split
gun_data$gun_stolen_cleaned <- ifelse(gun_data$gun_stolen_cleaned == "", "Unknown",
gun_data$gun_stolen_cleaned)
gun_data$gun_stolen_split <- strsplit(as.character(gun_data$gun_stolen_cleaned), "\\|\\|")
```

7. **Binary Matrix for gun_stolen:**

   Create binary matrix for gun_stolen statuses.

```
binary_df_gun_stolen <- data.frame(sapply(stolen_categories, function(category) {
  sapply(gun_data$gun_stolen_split, function(x) if(category %in% x) 1 else 0)
}))
colnames(binary_df_gun_stolen) <- ifelse(names(binary_df_gun_stolen) == "Unknown",
"Unknown_status", names(binary_df_gun_stolen))
```

8. **Combine Encoded Data:**

   Merge binary matrices with the dataset and remove intermediate columns.

```
gun_data_final <- cbind(gun_data, binary_df_gun_stolen, gun_data_categories)
gun_data_final <- gun_data_final[, !(colnames(gun_data_final) %in% c("gun_type", "gun_type_cleaned", "gun_type_split", "gun_stolen", "gun_stolen_cleaned", "gun_stolen_split", "gun_type_category"))]
```

Step 4: Feature Engineering

1. **Create n_casualties:**

   Add a new column for total casualties.

```
gun_data_final <- mutate(gun_data_final, n_casualties = n_killed + n_injured)
gun_data_final <- mutate(gun_data_final %>% relocate(n_casualties, .after = n_injured))
gun_data_final <- select(gun_data_final, -n_killed, -n_injured)
```

2. **Create severity:**

   Categorize severity based on casualties.

```
gun_data_final$severity <- cut(
  gun_data_final$n_casualties,
  breaks = c(-1, 0, 2, 5, 10, Inf),
  labels = c(1, 2, 3, 4, 5),
  right = TRUE
)
gun_data_final$severity <- as.numeric(as.character(gun_data_final$severity))
gun_data_final <- select(gun_data_final, -n_casualties)
```

Step 5: Split Data

1. **Partition Data**:

   Split into training (80%) and testing (20%).

```
train_indices <- createDataPartition(gun_data_final$severity, p = 0.8, list = FALSE)
train_data <- gun_data_final[train_indices, ]
test_data <- gun_data_final[-train_indices, ]
```

---

3. Model Implementation

Step 1: Reduce Data Size for Faster Training

1. **Set Seed for Reproducibility:**:
   This ensures that the random sampling produces the same results every time the script is run.

```
set.seed(123)
```

2. **Reduce Training and Testing Data Size:**
   A smaller subset of the data is used to speed up the training process.

```
sample_size_train <- 5000  # Reduced training rows
sample_size_test <- 1000   # Reduced testing rows
```

3. **Sample Data:**
   Random samples of the specified sizes are taken from the training and testing datasets.

```
train_data <- train_data[sample(nrow(train_data), sample_size_train), ]
test_data <- test_data[sample(nrow(test_data), sample_size_test), ]
```

Step 2: Train a Random Forest Model

1. **Define Training Control**:

   Configures cross-validation to validate the model during training:

   - **Method**: 3-fold cross-validation (cv).
   - **Progress Display**: Real-time feedback (verboseIter = TRUE).

```
train_control <- trainControl(
  method = "cv",  # Cross-validation
  number = 3,     # Reduced number of folds to speed up training
  verboseIter = TRUE # Show progress during training
)
```

2. **Train the Model**:

   A Random Forest model is trained on the reduced dataset using the caret package:

   - **Target Variable**: severity.
   - **Predictors**: All other variables.
   - **Hyperparameter Tuning**: Limited to two levels (tuneLength = 2) to speed up training.

```
rf_model <- train(
  severity ~ .,
  data = train_data,
  method = "rf",
  trControl = train_control,
  tuneLength = 2 # Reduced tuneLength for faster training
)
```

Output:

```
+ Fold1: mtry= 2
- Fold1: mtry= 2
+ Fold1: mtry=59
- Fold1: mtry=59
+ Fold2: mtry= 2
- Fold2: mtry= 2
+ Fold2: mtry=59
- Fold2: mtry=59
+ Fold3: mtry= 2
- Fold3: mtry= 2
+ Fold3: mtry=59
- Fold3: mtry=59
Aggregating results
Selecting tuning parameters
Fitting mtry = 2 on full training set
> |
```

Step 3: Evaluate Model Performance

1. **Generate Predictions**:

   The model predicts severity for the test dataset.

```
test_predictions <- predict(rf_model, newdata = test_data))
```

2. **Align Factor Levels**:

   Converts both predicted and actual values into factors with matching levels to ensure compatibility.

```
test_predictions <- factor(test_predictions, levels = levels(test_data$severity))

test_data$severity <- factor(test_data$severity, levels = levels(test_predictions))
```

3. **Confusion Matrix**:

   A confusion matrix evaluates the model's accuracy, precision, recall, and F1 score.

```
conf_matrix <- confusionMatrix(test_predictions, test_data$severity)
```

Step 4: Final Output

1. **Combine predictions and actual values**:
   Creates a DataFrame to compare predicted and actual values.

```
test_results <- data.frame(
  Actual = test_data$severity,
  Predicted = test_predictions
)
```

2. Preview Results:
   Displays the first few rows of the comparison.

```
head(test_results)
```

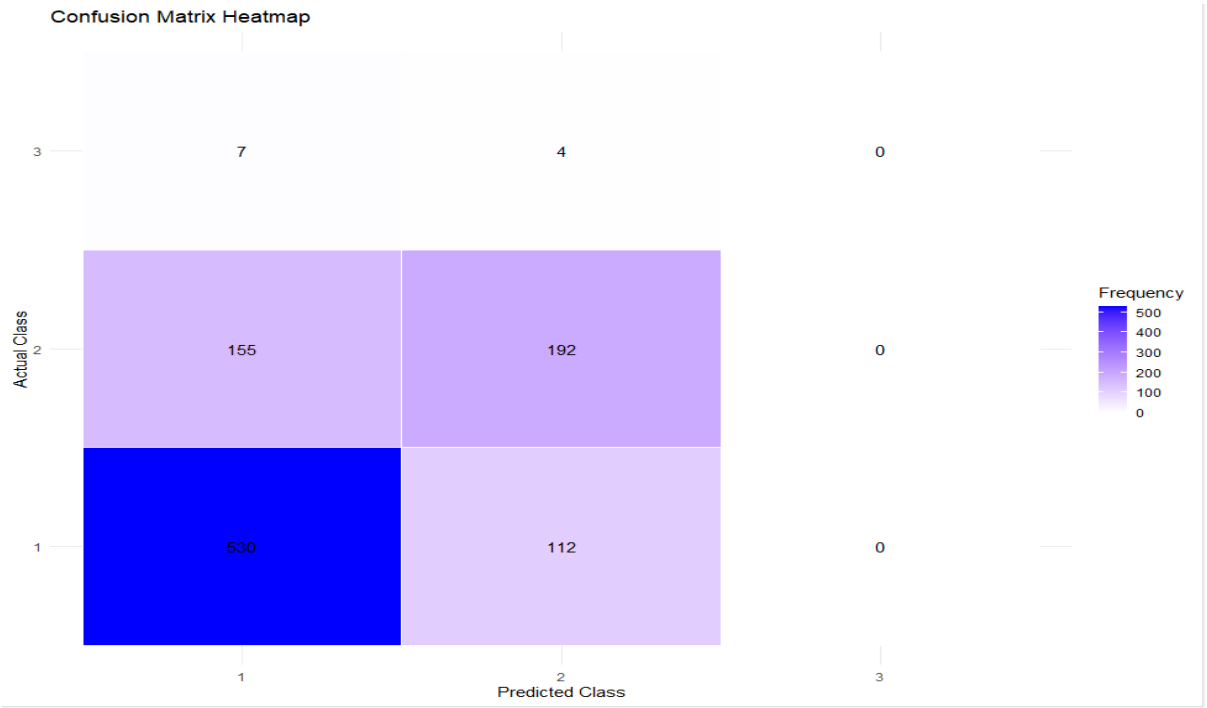
Step 5: Visualisations

1. Confusion Matrix Heatmap

- **Purpose: This heatmap provides a detailed view of the model's performance by visualizing how often the predicted severity matches the actual severity.**
- **Description: Each cell represents the frequency of predictions for a given actual class (y-axis) compared to predicted class (x-axis). Darker shades indicate higher frequencies.**
- **Insights: High values along the diagonal indicate strong predictive accuracy, where the model's predictions match the actual severity levels. Off-diagonal values represent misclassifications, which can highlight specific areas where the model struggles.**

```
conf_matrix_table <- as.table(conf_matrix$table)
conf_matrix_df <- as.data.frame(conf_matrix_table)
colnames(conf_matrix_df) <- c("Prediction", "Reference", "Frequency")

ggplot(conf_matrix_df, aes(x = Prediction, y = Reference)) +
  geom_tile(aes(fill = Frequency), color = "white") +
  scale_fill_gradient(low = "white", high = "blue") +
  geom_text(aes(label = Frequency), color = "black") +
  labs(
    title = "Confusion Matrix Heatmap",
    x = "Predicted Class",
    y = "Actual Class",
    fill = "Frequency"
  ) +
  theme_minimal()
```

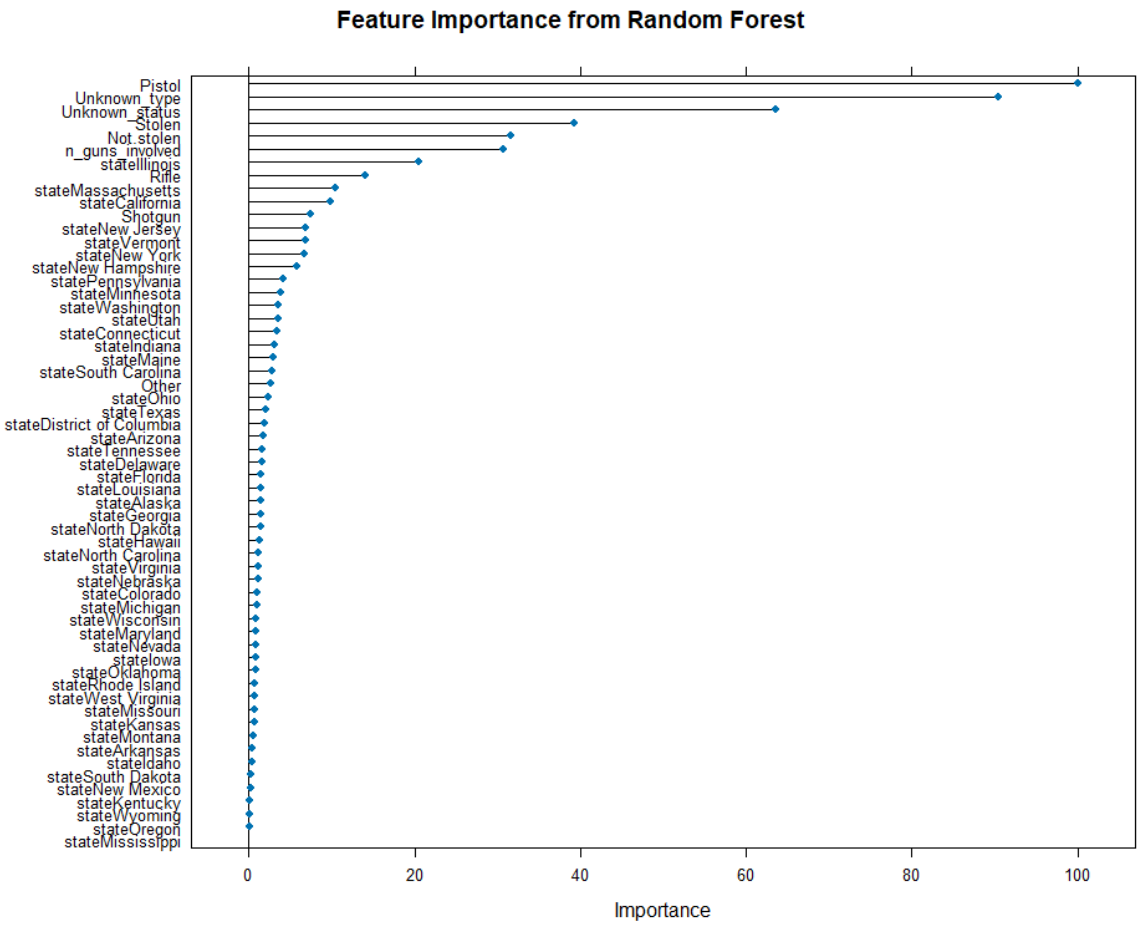


2. Feature Importance Plot

- **Purpose:** This plot ranks the importance of features in predicting incident severity, as determined by the Random Forest model.
- **Description:** Each bar represents a feature, with its length indicating the relative importance. Features contributing significantly to the predictions will have longer bars.
- **Insights:** The most important features provide actionable insights into what influences incident severity the most. For example, a feature like the number of guns involved might be highly influential, suggesting that interventions could focus on reducing access to guns.

```
varImp_rf <- varImp(rf_model)

plot(varImp_rf, main = "Feature Importance from Random Forest")
```



3. Predicted vs Actual Distribution

- **Purpose:** This bar chart compares the distribution of actual severity classes with the predicted classes to assess the model's classification tendencies.
- **Description:** Bars are grouped by actual classes, with different colors representing the predicted classes. This visualization helps identify any systemic biases in the model, such as over-predicting or under-predicting specific severity levels.
- **Insights:** A balanced distribution between actual and predicted classes indicates strong performance, whereas significant discrepancies might suggest areas for improvement, such as addressing class imbalance or refining feature engineering.

```
ggplot(test_results, aes(x = Actual, fill = Predicted)) +
 geom_bar(position = "dodge") +
 labs(
  title = "Distribution of Actual vs Predicted Classes",
  x = "Actual Class",
  fill = "Predicted Class"
 ) +
 theme_minimal()
```