Case Study

Load in the important libraries.

```
library(car)
```

```
## Loading required package: carData
```

```
library(iml)
library(xgboost)
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```
library(mice)
```

```
##
## Attaching package: 'mice'

## The following object is masked from 'package:stats':
##
##     filter

## The following objects are masked from 'package:base':
##
##     cbind, rbind
```

```
library(Rtsne)
library(umap)
library(here)
```

```
## here() starts at /Users/ianthomasdover/Reinsurance Case Study
```

```r
library(purrr)
```

```
##
## Attaching package: 'purrr'

## The following object is masked from 'package:caret':
##
##     lift

## The following object is masked from 'package:car':
##
##     some
```

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v stringr   1.5.1
## v forcats   1.0.0     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v readr     2.1.5

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks mice::filter(), stats::filter()
## x dplyr::lag()    masks stats::lag()
## x purrr::lift()   masks caret::lift()
## x dplyr::recode() masks car::recode()
## x dplyr::slice()  masks xgboost::slice()
## x purrr::some()   masks car::some()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
library(dplyr)
library(tidyr)
library(ggplot2)
library(lubridate)
```

Problem 1 – Data handling, analysis and plotting

The first problem of the case study builds on the data in the files p01-02_portfolio.csv and p01-02_rates.csv. One file contains membership information for a Group Life portfolio and one has information on the rates which should be charged.

```r
# Load the CSV file
portfolio_data <- read_delim(
  "Case Study/data/p01-02_portfolio.csv",
  delim = ";",
  show_col_types = FALSE
)

# View the data
head(portfolio_data)
```

```
## # A tibble: 6 x 5
##   SchemeName Date.of.Birth Gender DeathSI Industry
##   <chr>      <chr>         <chr>  <chr>   <chr>
## 1 Scheme2    29.05.1949    F      <NA>    Government & Public Administration
## 2 Scheme2    07.09.1950    F      <NA>    Government & Public Administration
## 3 Scheme2    27.09.1956    F      <NA>    Government & Public Administration
## 4 Scheme2    18.02.1942    F      <NA>    Government & Public Administration
## 5 Scheme2    31.07.1951    F      <NA>    Government & Public Administration
## 6 Scheme2    10.07.1960    F      <NA>    Government & Public Administration
```

```
# Load the CSV file
rates_data <- read_delim("Case Study/data/p01-02_rates.csv",
                         delim = ";",
                         show_col_types = FALSE)

# View the data
head(rates_data)
```

```
## # A tibble: 6 x 3
##     Age Gender  Rate
##   <dbl> <chr>  <dbl>
## 1    18 M       0.32
## 2    19 M       0.32
## 3    20 M       0.32
## 4    21 M       0.31
## 5    22 M       0.31
## 6    23 M       0.31
```

rates_data

```
# Count occurrences of each combination of Gender and Age
duplicates <- rates_data %>%
  group_by(Gender, Age) %>%
  summarise(count = n()) %>%
  filter(count > 1)
```

```
## `summarise()` has grouped output by 'Gender'. You can override using the
## `.groups` argument.
```

```
# Check if any duplicates exist
if (nrow(duplicates) == 0) {
  message("Sanity Check Passed: 'Gender' and 'Age' form a unique key.")
} else {
  message("Sanity Check Failed: There are duplicate combinations of 'Gender' and 'Age'.")
  print(duplicates)
}
```

```
## Sanity Check Passed: 'Gender' and 'Age' form a unique key.
```

Question a.

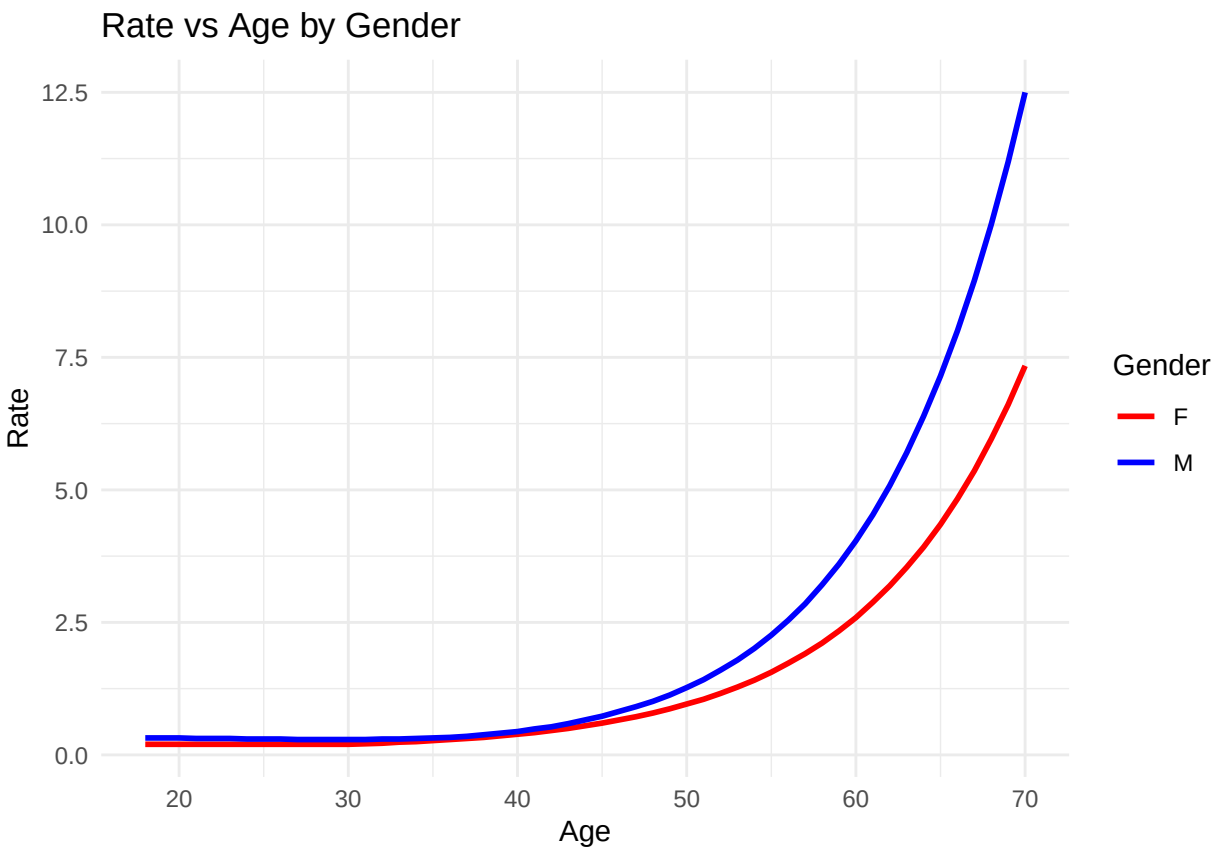
Read the data from the two files into R's memory. The rates are applicable to each individual in the portfolio, depending on that individual's age and gender. Combine the two datasets into a single table by looking up the rate for each line of the portfolio.

```r
# Step 1: Convert the Date.of.Birth column to Date format
# dmy is used for "day-month-year" format
portfolio_data$Date.of.Birth <- dmy(portfolio_data$Date.of.Birth)

# Step 2: Calculate the time difference in years
portfolio_data$age <- ceiling(interval(portfolio_data$Date.of.Birth, today()) / years(1))

# Step 3: View the updated data with age column
head(portfolio_data)
```

```
## # A tibble: 6 x 6
##   SchemeName Date.of.Birth Gender DeathSI Industry                          age
##   <chr>      <date>        <chr>  <chr>   <chr>                           <dbl>
## 1 Scheme2    1949-05-29    F      <NA>    Government & Public Administrat~    76
## 2 Scheme2    1950-09-07    F      <NA>    Government & Public Administrat~    75
## 3 Scheme2    1956-09-27    F      <NA>    Government & Public Administrat~    68
## 4 Scheme2    1942-02-18    F      <NA>    Government & Public Administrat~    83
## 5 Scheme2    1951-07-31    F      <NA>    Government & Public Administrat~    74
## 6 Scheme2    1960-07-10    F      <NA>    Government & Public Administrat~    65
```

Extrapolate Rates Data beyond 70 years

```r
# Create the line plot with grouping by Gender
ggplot(rates_data, aes(
  x = Age,
  y = Rate,
  color = Gender,
  group = Gender
)) +
  geom_line(linewidth = 1) +  # Use linewidth instead of size
  scale_color_manual(values = c("F" = "red", "M" = "blue")) +  # Set colors for genders
  labs(title = "Rate vs Age by Gender", x = "Age", y = "Rate") +
  theme_minimal()  # Use a clean theme for the plot
```



Fit an Exponential Model for Each Gender

```r
# Fit an exponential model for each gender
exp_model_female <- nls(Rate ~ exp(a + b * Age), data = subset(rates_data, Gender == "F"), start = list
exp_model_male <- nls(Rate ~ exp(a + b * Age), data = subset(rates_data, Gender == "M"), start = list(a

# Summarize the models
summary(exp_model_female)
```

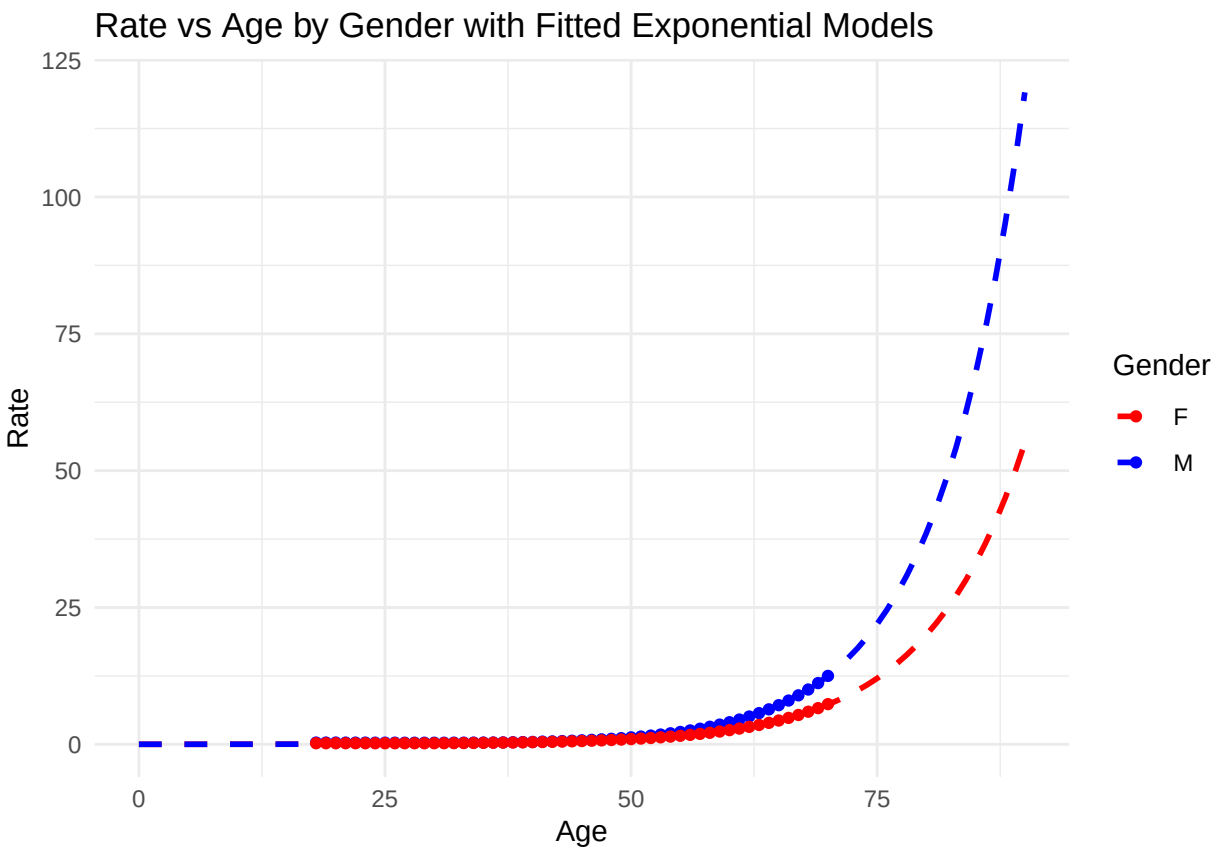
```
##
## Formula: Rate ~ exp(a + b * Age)
##
## Parameters:
##     Estimate Std. Error t value Pr(>|t|)
## a -5.1051905  0.0584289  -87.37   <2e-16 ***
## b  0.1012605  0.0008889  113.92   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.07425 on 51 degrees of freedom
##
## Number of iterations to convergence: 7
## Achieved convergence tolerance: 1.604e-06
```

```r
summary(exp_model_male)
```

```
##
## Formula: Rate ~ exp(a + b * Age)
##
## Parameters:
##   Estimate Std. Error t value Pr(>|t|)
## a -5.35891    0.06689  -80.12   <2e-16 ***
## b  0.11266    0.00101  111.49   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1247 on 51 degrees of freedom
##
## Number of iterations to convergence: 6
## Achieved convergence tolerance: 4.649e-06
```

Plot the Data and the Fitted Exponential Curves

```r
# Generate new data for ages 0 to 90 for both genders
new_data <- data.frame(Age = rep(0:90, 2), Gender = rep(c("F", "M"), each = 91))

# Predict rates for the new data using the fitted models
new_data$Rate_pred <- ifelse(
  new_data$Gender == "F",
  predict(exp_model_female, newdata = subset(new_data, Gender == "F")),
  predict(exp_model_male, newdata = subset(new_data, Gender == "M"))
)

# Plot the original data along with the fitted exponential curves
ggplot() +
  geom_line(
    data = new_data,
    aes(x = Age, y = Rate_pred, color = Gender),
    linewidth = 1,
    linetype = "dashed"
  ) +  # Fitted lines for each gender
  geom_point(data = rates_data, aes(x = Age, y = Rate, color = Gender)) +  # Original data points
  scale_color_manual(values = c("F" = "red", "M" = "blue")) +  # Set colors for genders
  labs(title = "Rate vs Age by Gender with Fitted Exponential Models", x = "Age", y = "Rate") +
  theme_minimal()
```



Add Extrapolated Values to Rates Data

```
# Get the unique values from the Gender column
unique_genders <- unique(rates_data$Gender)

# Print the unique values
unique_genders
```

```
## [1] "M" "F"
```

```
# Get the unique values from the Gender column
unique_genders <- unique(portfolio_data$Gender)

# Print the unique values
unique_genders
```

```
## [1] "F"      "M"      "Male"   "Female"
```

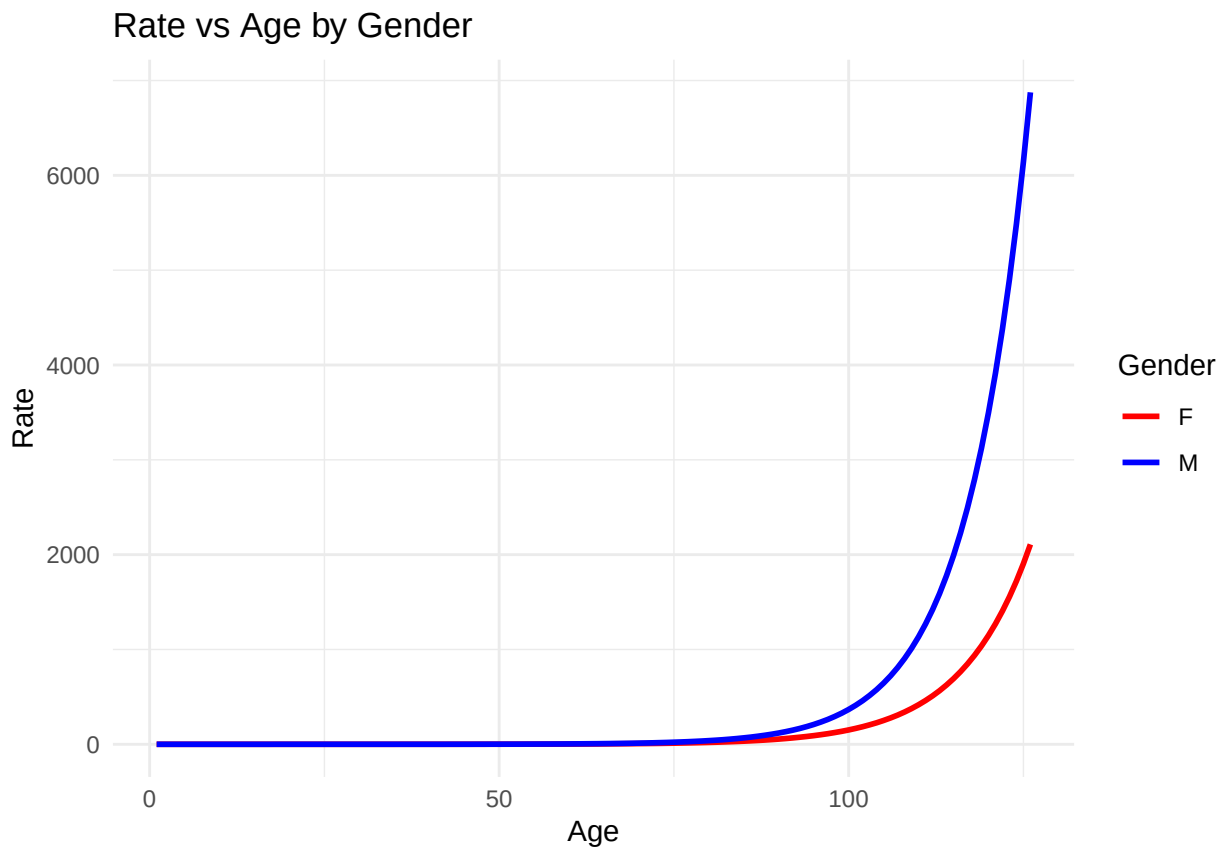
```
# Convert "Male" to "M" and "Female" to "F" using ifelse
portfolio_data$Gender <- ifelse(
  portfolio_data$Gender == "Male",
  "M",
  ifelse(portfolio_data$Gender == "Female", "F", portfolio_data$Gender)
```

```
)

# Get the unique values from the Gender column
unique_genders <- unique(portfolio_data$Gender)

# Print the unique values
unique_genders
```

```
## [1] "F" "M"
```

```
# Step 1: Create a complete sequence of ages from 1 to 110 for both genders
complete_ages <- expand.grid(Age = 1:126, Gender = c("F", "M"))

# Step 2: Identify missing "Age" and "Gender" combinations in the existing data
# Perform an anti-join to find missing combinations (from dplyr)
missing_ages <- anti_join(complete_ages, rates_data, by = c("Age", "Gender"))

# Step 3: Predict the rates for the missing combinations using the fitted models
missing_ages$Rate <- ifelse(
  missing_ages$Gender == "F",
  predict(exp_model_female, newdata = subset(missing_ages, Gender == "F")),
  predict(exp_model_male, newdata = subset(missing_ages, Gender == "M"))
)

# Step 4: Append the new data (with missing combinations filled) to the existing data
rates_data_extended <- rbind(rates_data, missing_ages)
```

```
# Create the line plot with grouping by Gender
ggplot(rates_data_extended,
       aes(
         x = Age,
         y = Rate,
         color = Gender,
         group = Gender
       )) +
  geom_line(linewidth = 1) +  # Use linewidth instead of size
  scale_color_manual(values = c("F" = "red", "M" = "blue")) +  # Set colors for genders
  labs(title = "Rate vs Age by Gender", x = "Age", y = "Rate") +
  theme_minimal()  # Use a clean theme for the plot
```




```
# Inner join the two datasets
combined_data <- inner_join(portfolio_data,
                            rates_data_extended,
                            by = c("age" = "Age", "Gender" = "Gender"),
)

# Check if the row count of the joined data matches the original portfolio data
if (nrow(combined_data) == nrow(portfolio_data)) {
  message("Sanity Check Passed: The row count of combined_data matches portfolio_data.")
} else {
  message("Sanity Check Failed: The row count of combined_data does not match portfolio_data.")
  message("Rows in portfolio_data: ", nrow(portfolio_data))
  message("Rows in combined_data: ", nrow(combined_data))
}
```

```
## Sanity Check Passed: The row count of combined_data matches portfolio_data.
```

```
# Get the unique values in the SchemeName column
unique_schemes <- unique(combined_data$SchemeName)
unique_schemes
```

```
## [1] "Scheme2" "Scheme1" "Scheme3" "Scheme5" "Scheme4"
```

```r
# Find rows in portfolio_data that do not have a match in rates_data
missing_matches <- anti_join(portfolio_data,
                             rates_data_extended,
                             by = c("age" = "Age", "Gender" = "Gender"))

# Check how many rows are missing
nrow(missing_matches)
```

Investigate missing matches

```
## [1] 0
```

Question b.

Group the Industry field into common-sense based groupings and determine the mean, standard deviation and quantiles of DeathSI for each of your industry groups.

```r
industry_counts <- combined_data %>%
  count(Industry) %>%
  arrange(desc(n))

# View the result
print(industry_counts)
```

```
## # A tibble: 33 x 2
##    Industry                                n
##    <chr>                               <int>
##  1 <NA>                               116769
##  2 Government & Public Administration  31304
##  3 Other                               14455
##  4 Sporting Club                        2243
##  5 Ex-Services Club                     1501
##  6 BSS-Business Services                1228
##  7 MAN-Manufacturing                    1147
##  8 EDN-Education                        1004
##  9 COM-Communication Serv.               878
## 10 FIN-Finance & Insurance               851
## # i 23 more rows
```

```r
combined_data <- combined_data %>%
  mutate(
    Industry_Group = case_when(
      Industry %in% c("Government & Public Administration", "Ex-Services Club") ~ "Government and Public
      Industry %in% c(
        "Sporting Club",
        "Golf Club",
        "Bowls Club",
        "Registered Club",
        "Surf Life Saving Club",
        "Workers Club",
```

```
        "Australian Rules Football Club",
        "Leagues Club",
        "Associated with Club Industry"
      ) ~ "Clubs and Associations",
      Industry %in% c(
        "BSS-Business Services",
        "FIN-Finance & Insurance",
        "Professional Services",
        "LAW-Solicitors/Barrister",
        "ENG-Engineers",
        "MGE-Medical Services Gen"
      ) ~ "Professional and Business Services",
      Industry %in% c(
        "MAN-Manufacturing",
        "CON-Construction",
        "ELE-Electricians",
        "VEH-Vehicle Industry",
        "WEO-Wholesale Trades"
      ) ~ "Manufacturing, Construction, and Trades",
      Industry %in% c(
        "EDN-Education",
        "HEA-Health Industry",
        "MGE-Medical Services Gen"
      ) ~ "Education and Health",
      Industry %in% c(
        "RTL-Retail Trade",
        "ACR-Accom. Cafes & Rests",
        "FOO-Food",
        "Hospitality"
      ) ~ "Retail, Hospitality, and Food",
      Industry %in% c("AGR-Farming/Agriculture", "EGW
-Electric/Gas/Water") ~ "Agriculture and Utilities",
Industry == "Other" ~ "Other",
TRUE ~ "Uncategorized"  # Catch any uncategorized industries
    )
  )


# View the newly grouped data
print(combined_data)
```

```
## # A tibble: 177,922 x 8
##    SchemeName Date.of.Birth Gender DeathSI Industry     age  Rate Industry_Group
##    <chr>      <date>        <chr>  <chr>   <chr>      <dbl> <dbl> <chr>
##  1 Scheme2    1949-05-29    F      <NA>    Governmen~    76 13.3  Government an~
##  2 Scheme2    1950-09-07    F      <NA>    Governmen~    75 12.1  Government an~
##  3 Scheme2    1956-09-27    F      <NA>    Governmen~    68  5.96 Government an~
##  4 Scheme2    1942-02-18    F      <NA>    Governmen~    83 27.1  Government an~
##  5 Scheme2    1951-07-31    F      <NA>    Governmen~    74 10.9  Government an~
##  6 Scheme2    1960-07-10    F      <NA>    Governmen~    65  4.35 Government an~
##  7 Scheme2    1954-12-24    F      <NA>    Governmen~    70  7.34 Government an~
##  8 Scheme2    1942-03-13    F      <NA>    Governmen~    83 27.1  Government an~
##  9 Scheme2    1958-02-28    F      <NA>    Governmen~    67  5.36 Government an~
```

```
## 10 Scheme2    1968-09-12    F      <NA>    Governmen~    57  1.91 Government an~
## # i 177,912 more rows
```

```
industry_counts <- combined_data %>%
  count(Industry_Group) %>%
  arrange(desc(n))

# View the result
print(industry_counts)
```

```
## # A tibble: 9 x 2
##   Industry_Group                              n
##   <chr>                                   <int>
## 1 Uncategorized                          117899
## 2 Government and Public Services          32805
## 3 Other                                   14455
## 4 Clubs and Associations                   4944
## 5 Manufacturing, Construction, and Trades  2529
## 6 Professional and Business Services       2494
## 7 Education and Health                     1396
## 8 Retail, Hospitality, and Food            1008
## 9 Agriculture and Utilities                 392
```

1. ```
   # Check the type of DeathSI
   typeof(combined_data$DeathSI)
   ```

   ```
   ## [1] "character"
   ```

   ```
   # Count the number of NA values in DeathSI when it was character type
   na_count <- sum(is.na(combined_data$DeathSI))
   na_count
   ```

   ```
   ## [1] 21531
   ```

   ```
   # Count the number of "NA" string values in DeathSI when it was character type
   na_string_count <- sum(combined_data$DeathSI == "NA", na.rm = TRUE)
   na_string_count
   ```

   ```
   ## [1] 0
   ```

   ```
   # Remove apostrophes and convert the DeathSI column from character to numeric
   combined_data$DeathSI <- as.numeric(gsub("'", "", combined_data$DeathSI))

   # Check the type of DeathSI
   typeof(combined_data$DeathSI)
   ```

   ```
   ## [1] "double"
   ```
   ```

```r
# Count the number of NA values in DeathSI when it is the double type
na_count <- sum(is.na(combined_data$DeathSI))
na_count
```

```
## [1] 21531
```

```r
# Calculate mean, standard deviation, and quantiles for each industry group
summary_stats <- combined_data %>%
  group_by(Industry_Group) %>%
  summarize(
    mean_value = mean(DeathSI, na.rm = TRUE),
    sd_value = sd(DeathSI, na.rm = TRUE),
    q25 = quantile(DeathSI, 0.25, na.rm = TRUE),
    median_value = median(DeathSI, na.rm = TRUE),
    q75 = quantile(DeathSI, 0.75, na.rm = TRUE)
  )

# View the result
print(summary_stats)
```

```
## # A tibble: 9 x 6
##   Industry_Group                  mean_value sd_value    q25 median_value    q75
##   <chr>                                <dbl>    <dbl>  <dbl>        <dbl>  <dbl>
## 1 Agriculture and Utilities          153992.  213974. 2.49e4       106412. 1.79e5
## 2 Clubs and Associations             275620.  183697. 1.98e5       219890. 2.96e5
## 3 Education and Health               323269.  256159. 1.40e5       279255  4.69e5
## 4 Government and Public Services     215104.  104726. 1.5 e5       220000  3   e5
## 5 Manufacturing, Construction, a~    289155.  237042. 1.17e5       250380  3.96e5
## 6 Other                              259435.  115096. 2.05e5       244264  2.85e5
## 7 Professional and Business Serv~    438260.  332429. 2.40e5       368416  5.44e5
## 8 Retail, Hospitality, and Food      313829.  206804. 2.05e5       245601  3.78e5
## 9 Uncategorized                      217656.  220107. 7.15e4       162404  2.83e5
```

Question c.

The following code performs a Monte Carlo simulation on the data you have loaded and combined in Question a.:

```r
set.seed(1234)
nsim <- 1000
res <- lapply(1:nsim, function(i,...) {
  x <- ifelse(
    runif(dim(combined_data)[1]) < combined_data$Rate / 1000,
    combined_data$DeathSI,
    0
  );
  list(cost = sum(x), count = length(x[x > 0]))
})
```

Apply this simulation to each scheme in the dataset you were provided, running 1000 simulations per scheme. Produce a plot of the simulated outcomes ("cost"). Your plot should show:

- a separate histogram per scheme;
- all 5 histograms below each other so that they can be easily compared;
- vertical lines in each graph indicating the median, mean and 99.5th percentile of each distribution.

Remove rows where DeathSI is NA for Monte Carlo simulation.

```
# Get the number of rows in the original dataset
original_row_count <- nrow(combined_data)

# Subset combined_data where DeathSI is not NA
combined_data_death_si_non_na <- subset(combined_data, !is.na(DeathSI))

# Get the number of rows in the filtered dataset
filtered_row_count <- nrow(combined_data_death_si_non_na)

# Print out the row counts to validate reduction
cat("Original row count:", original_row_count, "\n")
```

```
## Original row count: 177922
```

```
cat("Filtered row count (DeathSI not NA):", filtered_row_count, "\n")
```

```
## Filtered row count (DeathSI not NA): 156391
```

```
# Check if the row count was reduced
if (filtered_row_count < original_row_count) {
  cat("Row reduction validated: Rows were reduced after filtering.\n")
} else {
  cat("No row reduction: No NA values in DeathSI.\n")
}
```

```
## Row reduction validated: Rows were reduced after filtering.
```

```
monte_carlo_simulation <- function(data, nsim = 1000, seed = 1234) {
  # Set the seed for reproducibility
  set.seed(seed)

  # Perform the simulation
  res <- lapply(1:nsim, function(i, ...) {
    x <- ifelse(runif(dim(data)[1]) < data$Rate / 1000, data$DeathSI, 0)
    # Return the cost and count as a list
    list(cost = sum(x), count = length(x[x > 0]))
  })

  # Return the result of the simulation
  return(res)
}

# Get the unique values in the SchemeName column
```

```r
unique_schemes <- unique(combined_data$SchemeName)

# Print the unique values
unique_schemes
```

```
## [1] "Scheme2" "Scheme1" "Scheme3" "Scheme5" "Scheme4"
```

```r
# Filter rows where SchemeName is "Scheme_1", "Scheme_2", "Scheme_3", "Scheme_4", "Scheme_5"
combined_data_scheme_1 <- subset(combined_data_death_si_non_na, SchemeName == "Scheme1")
combined_data_scheme_2 <- subset(combined_data_death_si_non_na, SchemeName == "Scheme2")
combined_data_scheme_3 <- subset(combined_data_death_si_non_na, SchemeName == "Scheme3")
combined_data_scheme_4 <- subset(combined_data_death_si_non_na, SchemeName == "Scheme4")
combined_data_scheme_5 <- subset(combined_data_death_si_non_na, SchemeName == "Scheme5")

# Sanity check that each subset has more than 0 rows
check_scheme_1 <- nrow(combined_data_scheme_1) > 0
check_scheme_2 <- nrow(combined_data_scheme_2) > 0
check_scheme_3 <- nrow(combined_data_scheme_3) > 0
check_scheme_4 <- nrow(combined_data_scheme_4) > 0
check_scheme_5 <- nrow(combined_data_scheme_5) > 0

# Print the results
cat("Scheme 1 has more than 0 rows:", check_scheme_1, "\n")
```

```
## Scheme 1 has more than 0 rows: TRUE
```

```r
cat("Scheme 2 has more than 0 rows:", check_scheme_2, "\n")
```

```
## Scheme 2 has more than 0 rows: TRUE
```

```r
cat("Scheme 3 has more than 0 rows:", check_scheme_3, "\n")
```

```
## Scheme 3 has more than 0 rows: TRUE
```

```r
cat("Scheme 4 has more than 0 rows:", check_scheme_4, "\n")
```

```
## Scheme 4 has more than 0 rows: TRUE
```

```r
cat("Scheme 5 has more than 0 rows:", check_scheme_5, "\n")
```

```
## Scheme 5 has more than 0 rows: TRUE
```

```r
# Perform a monte carlo simulation for each scheme
monte_carlo_scheme_1_result <- monte_carlo_simulation(combined_data_scheme_1)
monte_carlo_scheme_2_result <- monte_carlo_simulation(combined_data_scheme_2)
monte_carlo_scheme_3_result <- monte_carlo_simulation(combined_data_scheme_3)
monte_carlo_scheme_4_result <- monte_carlo_simulation(combined_data_scheme_4)
monte_carlo_scheme_5_result <- monte_carlo_simulation(combined_data_scheme_5)
```

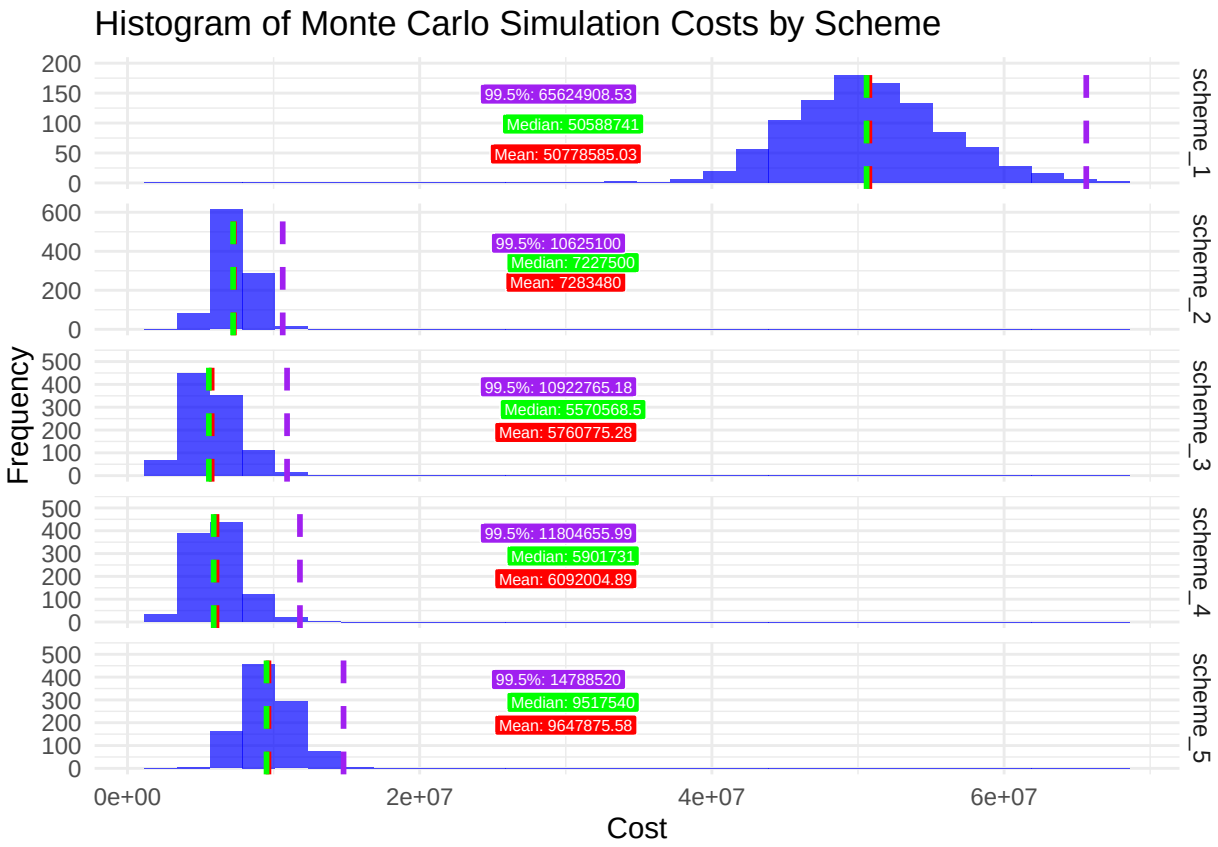
```r
# Assuming you have 5 Monte Carlo simulation results, stored in a list
monte_carlo_results <- list(
  scheme_1 = monte_carlo_scheme_1_result,
  scheme_2 = monte_carlo_scheme_2_result,
  scheme_3 = monte_carlo_scheme_3_result,
  scheme_4 = monte_carlo_scheme_4_result,
  scheme_5 = monte_carlo_scheme_5_result
)

# Create an empty data frame to hold all the costs and corresponding scheme names
all_costs_data <- data.frame()

# Loop over each result and calculate costs, then combine into one data frame
for (scheme_name in names(monte_carlo_results)) {
  costs <- sapply(monte_carlo_results[[scheme_name]], function(x)
    x$cost)
  temp_data <- data.frame(Cost = costs, Scheme = scheme_name)
  all_costs_data <- rbind(all_costs_data, temp_data)
}

all_stats <- all_costs_data %>%
  group_by(Scheme) %>%
  summarise(
    mean_cost = mean(Cost),
    median_cost = median(Cost),
    percentile_99_5_cost = quantile(Cost, 0.995)
  )

# Create different x and y positions for each label depending on the scheme
all_stats <- all_stats %>%
  mutate(
    # Dynamic x positions for each scheme
    x_position_mean = ifelse(
      Scheme == "scheme_1",
      3e+07,
      ifelse(
        Scheme == "scheme_2",
        3e+07,
        ifelse(
          Scheme == "scheme_3",
          3e+07,
          ifelse(Scheme == "scheme_4", 3e+07, 3e+07)
        )
      )
    ),
    x_position_median = x_position_mean + 0.5e+06,
    # Offset for median
    x_position_995 = x_position_mean - 0.5e+06,
    # Offset for 99.5th percentile

    # Dynamic y positions for each scheme
    y_position_mean = ifelse(
      Scheme == "scheme_1",
```

```
      100,
      ifelse(
        Scheme == "scheme_2",
        400,
        ifelse(Scheme == "scheme_3", 325, ifelse(Scheme == "scheme_4", 325, 325))
      )
    ),
    y_position_median = ifelse(
      Scheme == "scheme_1",
      150,
      ifelse(
        Scheme == "scheme_2",
        500,
        ifelse(Scheme == "scheme_3", 425, ifelse(Scheme == "scheme_4", 425, 425))
      )
    ),
    y_position_995 = ifelse(
      Scheme == "scheme_1",
      200,
      ifelse(
        Scheme == "scheme_2",
        600,
        ifelse(Scheme == "scheme_3", 525, ifelse(Scheme == "scheme_4", 525, 525))
      )
    ),
  )

# Create the faceted plot with adjusted x and y positions for each level in the stack
ggplot(all_costs_data, aes(x = Cost)) +
  geom_histogram(bins = 30,
                 fill = "blue",
                 alpha = 0.7) +  # Set the number of bins
  # Add vertical lines for mean, median, and 99.5th percentile
  geom_vline(
    data = all_stats,
    aes(xintercept = mean_cost),
    color = "red",
    linetype = "dashed",
    size = 1
  ) +
  geom_vline(
    data = all_stats,
    aes(xintercept = median_cost),
    color = "green",
    linetype = "dashed",
    size = 1
  ) +
  geom_vline(
    data = all_stats,
    aes(xintercept = percentile_99_5_cost),
    color = "purple",
    linetype = "dashed",
    size = 1
```

```
  ) +

  # Add labels with dynamically adjusted x and y positions for each scheme
  geom_label(
    data = all_stats,
    aes(
      x = x_position_mean,
      y = y_position_mean,
      label = paste("Mean:", round(mean_cost, 2))
    ),
    color = "white",
    fill = "red",
    size = 2,
    angle = 0,
    vjust = 2,
    hjust = 0.5
  ) +
  geom_label(
    data = all_stats,
    aes(
      x = x_position_median,
      y = y_position_median,
      label = paste("Median:", round(median_cost, 2))
    ),
    color = "white",
    fill = "green",
    size = 2,
    angle = 0,
    vjust = 2,
    hjust = 0.5
  ) +
  geom_label(
    data = all_stats,
    aes(
      x = x_position_995,
      y = y_position_995,
      label = paste("99.5%:", round(percentile_99_5_cost, 2))
    ),
    color = "white",
    fill = "purple",
    size = 2,
    angle = 0,
    vjust = 2,
    hjust = 0.5
  ) +

  # Facet the plot vertically for each scheme
  facet_grid(Scheme ~ ., scales = "free_y") +
  labs(title = "Histogram of Monte Carlo Simulation Costs by Scheme", x = "Cost", y = "Frequency") +
  theme_minimal()
```

```
## Warning: Using ‘size‘ aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use ‘linewidth‘ instead.
```

```
## This warning is displayed once every 8 hours.
## Call 'lifecycle::last_lifecycle_warnings()' to see where this warning was
## generated.
```



Problem 2 – Statistical learning

The following code loads the data used in Problem 2. The data is stored in two .csv files containing a data set and a scoring set, respectively. These data are NOT related to Problem 1.

```r
library(tidyverse)
library(here)

dta     <- read_csv(
  here("data/p02re_data.csv"),
  col_types = cols()
)
scoring <- read_csv(
  here("data/p02re_scoring.csv"),
  col_types = cols()
)
```

The goal of this problem is to predict the value of the variable outcome in the dataset defined above. The measure of goodness of fit is the area under the receiver operating characteristic curve (AUC) and will be measured on the scoring set.

```r
dta <- read_csv(
  here("Case Study/data/p02re_data.csv"),
  col_types = cols()
)

scoring <- read_csv(
  here("Case Study/data/p02re_scoring.csv"),
  col_types = cols()
)

head(dta)
```

```
## # A tibble: 6 x 11
##   outcome group     q     r       s       t      u      v       w       x
##     <dbl> <chr> <dbl> <dbl>   <dbl>   <dbl>  <dbl>  <dbl>   <dbl>   <dbl>
## 1       0 b     2.70     NA  1.69   -0.0179  1.09   0.938  0.296   0.620
## 2       1 b     0.647    NA -0.951  -0.223  -0.384  0.328  0.160  -0.307
## 3       0 d     1.60     NA -0.219   0.480  -0.630 -0.242 -0.435  -0.0416
## 4       0 a     2.90     NA  0.224   0.0942 -0.163  0.975  0.826   0.834
## 5       0 f     0.572    NA  0.0153 -1.23    0.322 -0.668 -0.0807 -0.367
## 6       0 g     0.860    NA -0.916  -1.05    1.60   0.144 -0.933  -0.461
## # i 1 more variable: z <dbl>
```

```r
head(scoring)
```

```
## # A tibble: 6 x 10
##   group     q      r        s      t       u       v       w       x       z
##   <chr> <dbl>  <dbl>    <dbl>  <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 f     0.602 NA      0.242   -0.414  0.736   0.466   0.826   0.764   0.720
## 2 g     1.83  NA     -0.00545  1.25  -0.768   0.0911 -0.0695  0.979  -0.857
## 3 c     0.887  0.647 -0.928    0.463  0.189   0.0560  0.721  -0.104   0.171
## 4 a     2.31  NA     -0.214    0.231  0.143  -0.123   0.163  -0.0892  0.745
## 5 e     0.957  0.199  0.755    2.02  -0.0480 -0.550   0.177   0.640   0.0659
## 6 b     1.76  NA     -0.132   -0.617 -0.454   0.818   0.579  -1.28    0.284
```

```r
unique_values <- sort(unique(dta$group))
print(unique_values)
```

```
## [1] "a" "b" "d" "e" "f" "g"
```

```r
unique_values <- sort(unique(scoring$group))
print(unique_values)
```

```
## [1] "a" "b" "c" "e" "f" "g"
```

```r
# For the first dataset (dta)
dta_percent <- dta %>%
  group_by(group) %>%
  summarise(count = n()) %>%
  mutate(percentage = count / sum(count) * 100, dataset = "dta")
```

```r
# For the second dataset (scoring)
scoring_percent <- scoring %>%
  group_by(group) %>%
  summarise(count = n()) %>%
  mutate(percentage = count / sum(count) * 100, dataset = "scoring")

# Combine both datasets
combined_data <- bind_rows(dta_percent, scoring_percent)

ggplot(combined_data, aes(x = group, y = percentage, fill = dataset)) +
  geom_bar(stat = "identity", position = "dodge") +  # side-by-side bars
  labs(title = "Distribution of Group in dta and scoring datasets", x = "Group", y = "Percentage") +
  theme_minimal()
```

Observation: The "scoring" set contains a different Group, C, from the "dta" set containing the group, D. It may be worth checking to see if the Groups from both datasets derive from the same distribution or if the naming is erroneous. Perform the KS-test across distributions:

```r
# Define the groups you want to check
groups <- c("a", "b", "e", "f", "g")

# Function to perform the KS test for a single group
perform_ks_test <- function(grp) {
  filtered_dta <- dta %>% filter(group == grp)
```

```r
  filtered_scoring <- scoring %>% filter(group == grp)

  # Apply KS test only for numeric columns
  ks_results <- map_dbl(names(filtered_dta), function(col) {
    if (is.numeric(filtered_dta[[col]]) &&
        is.numeric(filtered_scoring[[col]])) {
      ks.test(filtered_dta[[col]], filtered_scoring[[col]])$p.value
    } else {
      NA_real_  # Return NA for non-numeric columns
    }
  })

  # Display results
  cat("\nGroup:", grp, "\n")
  print(ks_results)

  # Check if any p-values are below 0.05
  if (any(ks_results < 0.05, na.rm = TRUE)) {
    cat("\nGroup",
        grp,
        "shows a significant difference in distribution.\n")
  } else {
    cat(
      "\nNot enough information to suggest that Group",
      grp,
      "comes from a different distribution.\n"
    )
  }
}

# Use purrr::walk to apply the function for each group
walk(groups, perform_ks_test)
```

```
##
## Group: a
##  [1]        NA        NA 0.5748517 0.4370151 0.2377937 0.8895388 0.6907044
##  [8] 0.2900435 0.4513656 0.7040462 0.1843067
##
## Not enough information to suggest that Group a comes from a different distribution.
##
## Group: b
##  [1]        NA        NA 0.2916185 0.9893133 0.8159882 0.8958087 0.4129914
##  [8] 0.4355943 0.9835987 0.6428534 0.3412154
##
## Not enough information to suggest that Group b comes from a different distribution.
##
## Group: e
##  [1]        NA        NA 0.9094373 0.5233350 0.7870813 0.7052981 0.6968841
##  [8] 0.1116149 0.9862832 0.9724978 0.4498295
##
## Not enough information to suggest that Group e comes from a different distribution.
##
## Group: f
```

```
##  [1]         NA         NA 0.28690761 0.01326737 0.34615141 0.23143280
##  [7] 0.48704818 0.20637852 0.47489214 0.95703205 0.96215333
##
## Group f shows a significant difference in distribution.
##
## Group: g
##  [1]         NA         NA 0.75627507 0.79542353 0.29263176 0.53831301
##  [7] 0.57355907 0.03741027 0.38249216 0.17915481 0.21962804
##
## Group g shows a significant difference in distribution.
```

```r
filtered_dta <- dta %>% filter(group == "d")
filtered_scoring <- scoring %>% filter(group == "c")

# Apply KS test only for numeric columns
ks_results <- sapply(names(filtered_dta), function(col) {
  if (is.numeric(filtered_dta[[col]]) &&
      is.numeric(filtered_scoring[[col]])) {
    ks.test(filtered_dta[[col]], filtered_scoring[[col]])$p.value
  } else {
    NA  # If column is not numeric, return NA
  }
})

# View results
ks_results
```

```
##   outcome     group         q         r         s         t         u         v
##        NA        NA 0.1964348 0.8523103 0.4655289 0.3948154 0.8001175 0.6070964
##         w         x         z
## 0.6994330 0.7777700 0.6200299
```

Check Missing Values and Perform Imputation Technique

```r
# Function to get and display missing value counts for a dataset
display_na_counts <- function(dataset, name) {
  na_counts <- sapply(dataset, function(x)
    sum(is.na(x)))
  cat(paste("Missing values in '", name, "':\n", sep = ""))
  print(na_counts)
  cat("\n")
}

# Display missing values before imputation
display_na_counts(dta, "dta")
```

```
## Missing values in 'dta':
## outcome   group       q       r       s       t       u       v       w       x
##       0       0       0    7002       0       0       0       0       0       0
##       z
##       0
```

```r
display_na_counts(scoring, "scoring")
```

```
## Missing values in 'scoring':
## group     q     r     s     t     u     v     w     x     z
##     0     0  1398     0     0     0     0     0     0     0
```

```r
# Define imputation methods, excluding 'outcome' column from dta
method_dta <- make.method(dta)
method_dta["outcome"] <- ""

# Perform imputation on both datasets
#   •   Predictive Mean Matching (pmm):
#   •   Imputes missing values by finding observed values in the dataset that are similar (in terms of p
#   •   This method is often used for numeric data and is robust since it preserves the original distri

dta_imputed <- mice(dta, method = method_dta, m = 5)
```

```
##
##  iter imp variable
##   1   1  r
##   1   2  r
##   1   3  r
##   1   4  r
##   1   5  r
##   2   1  r
##   2   2  r
##   2   3  r
##   2   4  r
##   2   5  r
##   3   1  r
##   3   2  r
##   3   3  r
##   3   4  r
##   3   5  r
##   4   1  r
##   4   2  r
##   4   3  r
##   4   4  r
##   4   5  r
##   5   1  r
##   5   2  r
##   5   3  r
##   5   4  r
##   5   5  r
```

```
## Warning: Number of logged events: 1
```

```r
scoring_imputed <- mice(scoring, method = 'pmm', m = 5)
```

```
##
##  iter imp variable
```

```
##   1   1  r
##   1   2  r
##   1   3  r
##   1   4  r
##   1   5  r
##   2   1  r
##   2   2  r
##   2   3  r
##   2   4  r
##   2   5  r
##   3   1  r
##   3   2  r
##   3   3  r
##   3   4  r
##   3   5  r
##   4   1  r
##   4   2  r
##   4   3  r
##   4   4  r
##   4   5  r
##   5   1  r
##   5   2  r
##   5   3  r
##   5   4  r
##   5   5  r
```

```
## Warning: Number of logged events: 1
```

```r
# Suppress any output while extracting the complete data after imputation
invisible(dta_complete <- complete(dta_imputed))
invisible(scoring_complete <- complete(scoring_imputed))

# Display missing values after imputation
display_na_counts(dta_complete, "dta (complete)")
```

```
## Missing values in 'dta (complete)':
## outcome   group       q       r       s       t       u       v       w       x
##       0       0       0       0       0       0       0       0       0       0
##       z
##       0
```

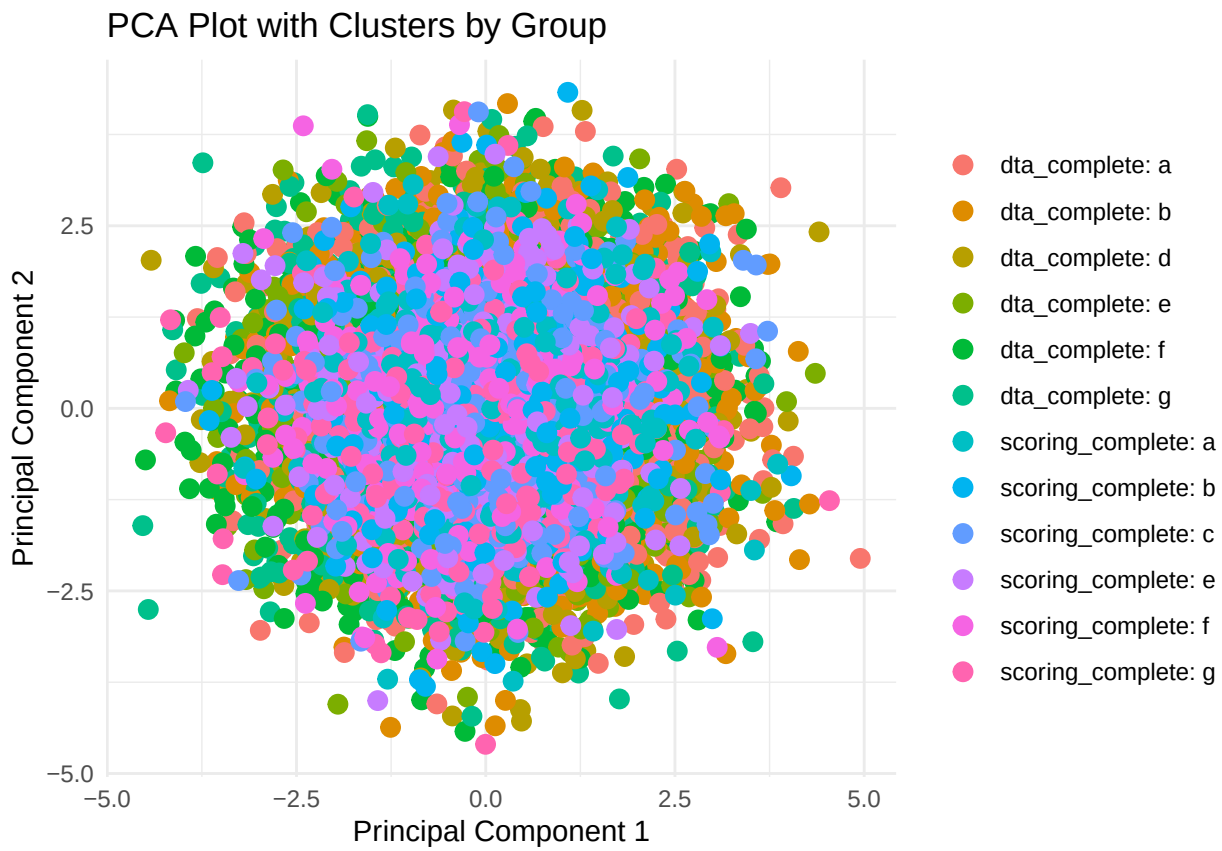
```r
display_na_counts(scoring_complete, "scoring (complete)")
```

```
## Missing values in 'scoring (complete)':
## group     q     r     s     t     u     v     w     x     z
##     0     0     0     0     0     0     0     0     0     0
```

Perform PCA to Visualize Groups

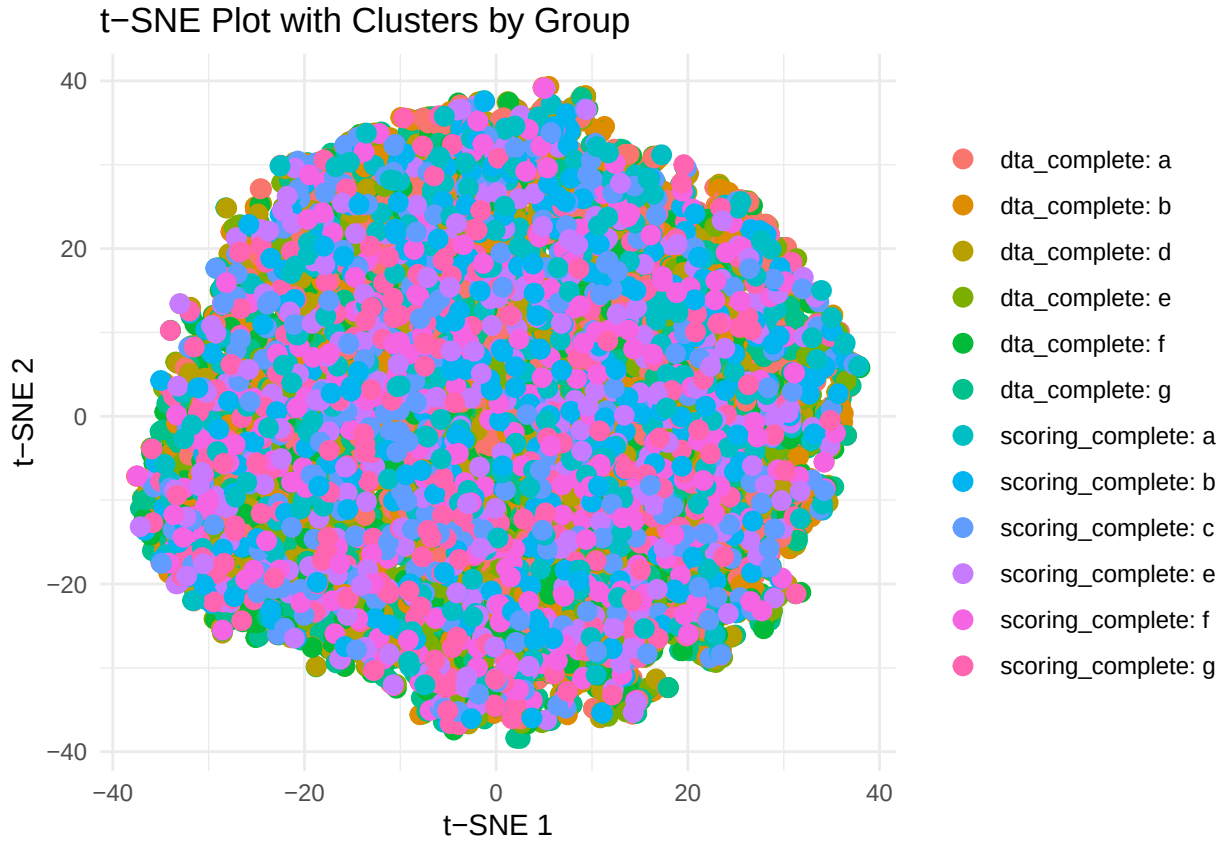
```r
dta_complete_group_renamed <- dta_complete %>%
  mutate(group = paste("dta_complete:", group))
```

```r
scoring_complete_group_renamed <- scoring_complete %>%
  mutate(group = paste("scoring_complete:", group))

# Create a renamed copy of the data without the "outcome" field
dta_renamed <- dta_complete_group_renamed %>% select(-outcome)

# Combine dta_renamed and scoring vertically (row-wise)
combined_part_2_data <- bind_rows(dta_renamed, scoring_complete_group_renamed)

# Retain the 'group' column separately
group_column <- combined_part_2_data$group

# Select only the numeric columns for PCA (excluding 'group')
numeric_data <- combined_part_2_data %>% select(where(is.numeric))

# Perform PCA and reduce to 2 principal components
pca_result <- prcomp(numeric_data, center = TRUE, scale. = TRUE)

# Extract the first 2 principal components
pca_data <- as.data.frame(pca_result$x[, 1:2])

# Rename the principal components for clarity
colnames(pca_data) <- c("PC1", "PC2")

# Add the 'group' column back to the PCA data
pca_data_with_group <- cbind(group = group_column, pca_data)

# Create a 2D scatter plot with different colors for each group
ggplot(pca_data_with_group, aes(x = PC1, y = PC2, color = group)) +
  geom_point(size = 3) +
  labs(title = "PCA Plot with Clusters by Group",
       x = "Principal Component 1",
       y = "Principal Component 2") +
  theme_minimal() +
  theme(legend.title = element_blank())  # Remove legend title
```



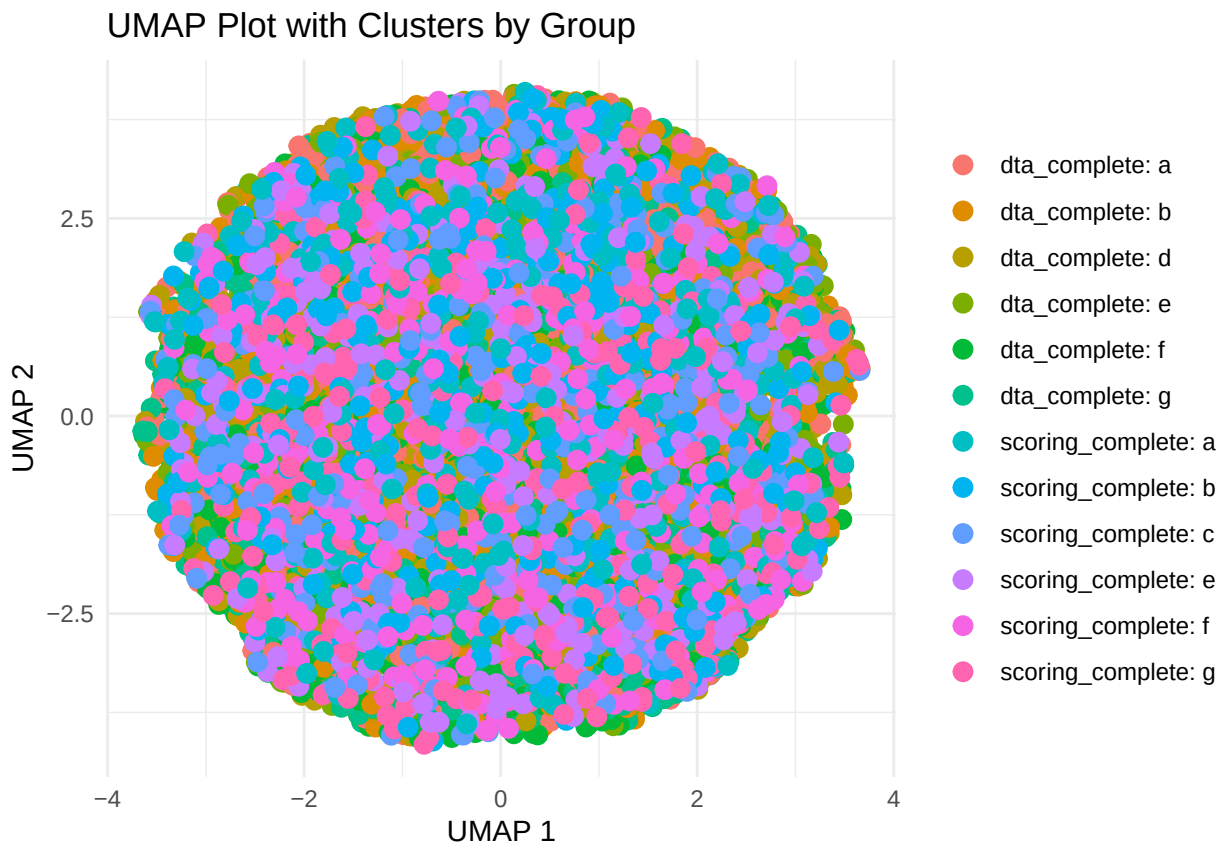

```r
# Rename 'group' column in both datasets
dta_complete_group_renamed <- dta_complete %>%
  mutate(group = paste("dta_complete:", group))

scoring_complete_group_renamed <- scoring_complete %>%
  mutate(group = paste("scoring_complete:", group))

# Create a renamed copy of the data without the "outcome" field
dta_renamed <- dta_complete_group_renamed %>% select(-outcome)

# Combine dta_renamed and scoring_complete vertically (row-wise)
combined_part_2_data <- bind_rows(dta_renamed, scoring_complete_group_renamed)

# Retain the 'group' column separately
group_column <- combined_part_2_data$group

# Select only the numeric columns for t-SNE (excluding 'group')
numeric_data <- combined_part_2_data %>% select(where(is.numeric))

# Perform t-SNE on the numeric data
tsne_result <- Rtsne(
  numeric_data,
  dims = 2,
  perplexity = 30,
  verbose = FALSE
)
```

```r
# Convert the t-SNE result into a data frame
tsne_data <- as.data.frame(tsne_result$Y)

# Rename the t-SNE components for clarity
colnames(tsne_data) <- c("TSNE1", "TSNE2")

# Add the 'group' column back to the t-SNE data
tsne_data_with_group <- cbind(group = group_column, tsne_data)

# Create a 2D scatter plot with different colors for each group
ggplot(tsne_data_with_group, aes(x = TSNE1, y = TSNE2, color = group)) +
  geom_point(size = 3) +
  labs(title = "t-SNE Plot with Clusters by Group", x = "t-SNE 1", y = "t-SNE 2") +
  theme_minimal() +
  theme(legend.title = element_blank())  # Remove legend title
```




```r
# Rename 'group' column in both datasets
dta_complete_group_renamed <- dta_complete %>%
  mutate(group = paste("dta_complete:", group))

scoring_complete_group_renamed <- scoring_complete %>%
  mutate(group = paste("scoring_complete:", group))

# Create a renamed copy of the data without the "outcome" field
dta_renamed <- dta_complete_group_renamed %>% select(-outcome)
```

```r
# Combine dta_renamed and scoring_complete vertically (row-wise)
combined_part_2_data <- bind_rows(dta_renamed, scoring_complete_group_renamed)

# Retain the 'group' column separately
group_column <- combined_part_2_data$group

# Select only the numeric columns for UMAP (excluding 'group')
numeric_data <- combined_part_2_data %>% select(where(is.numeric))

# Perform UMAP on the numeric data
umap_result <- umap(numeric_data)

# Convert UMAP result into a data frame
umap_data <- as.data.frame(umap_result$layout)

# Rename the UMAP components for clarity
colnames(umap_data) <- c("UMAP1", "UMAP2")

# Add the 'group' column back to the UMAP data
umap_data_with_group <- cbind(group = group_column, umap_data)

# Create a 2D scatter plot with different colors for each group
ggplot(umap_data_with_group, aes(x = UMAP1, y = UMAP2, color = group)) +
  geom_point(size = 3) +
  labs(title = "UMAP Plot with Clusters by Group", x = "UMAP 1", y = "UMAP 2") +
  theme_minimal() +
  theme(legend.title = element_blank())  # Remove legend title
```



Question a.

Describe the statistical learning problem. Explain the difficulties in training a model with the above dataset.

This learning problem was initially considered to fall under **semi-supervised learning**, which combines labeled data (10,000 rows with targets from the "dta" set) with a smaller amount of unlabeled data (2,000 rows without targets from the "scoring" set). The goal is typically to leverage both datasets to improve model performance, learning the structure of the data from the unlabeled dataset and refining predictions on the labeled dataset.

However, **clustering on the "scoring" dataset has shown poor performance**, indicating that the unlabeled data may not contribute meaningful structure for semi-supervised methods. As a result, we are likely to opt for a more traditional **supervised learning approach** using only the labeled "dta" dataset. While **semi-supervised learning** remains possible, it may not provide the expected improvements in this case.

Key Challenges:

1. **Handling Missing Values (NaNs in "R"):** Missing data in the independent variable "R" complicates model training. Imputation methods such as mean, median, or model-based imputation can be employed, but they carry risks of bias or information loss, which could affect model performance.
2. **Bias Between Datasets**: The distribution of features in the "scoring" dataset may differ significantly from the labeled "dta" dataset. If the 2,000 rows without a target differ in distribution from the 10,000 labeled rows, the model may struggle to generalize well, further limiting the effectiveness of semi-supervised learning.

3. **Model Complexity**: Semi-supervised learning techniques like self-training, co-training, or graph-based methods can efficiently utilize the unlabeled data, but they often involve complex, computationally expensive processes. Given the poor clustering results, these methods may not offer a substantial benefit over simpler, fully supervised models.

4. **Feature Engineering for Unlabeled Data**: Extracting meaningful patterns from the 2,000-row unlabeled dataset has proven challenging. Methods such as clustering and dimensionality reduction (e.g., PCA) haven't provided clear groupings, which diminishes the potential for effective semi-supervised learning.

5. **Class Imbalance**: If class imbalance exists in the target variable within the labeled dataset, the model may be prone to overfitting to the majority class. This risk increases in the absence of class information from the unlabeled data, which could otherwise help balance the minority class representation.

6. **Semi-supervised Algorithms**: Although semi-supervised algorithms like Label Propagation, Self-learning, and Generative Models are designed to handle such cases, they may not offer significant advantages when the structure of the unlabeled data (from the "scoring" set) is not informative. In contrast, conventional supervised algorithms, such as decision trees or gradient-boosting models, may perform better in this scenario.

Conclusion:

Given the current challenges and the poor performance of clustering on the "scoring" dataset, **a supervised learning approach** using only the labeled "dta" dataset is likely to yield better results. Semi-supervised learning remains an option, but without clear structure in the unlabeled data, it is less promising in this particular case. Specific strategies for data preparation, model selection, and evaluation will be key to building a robust model that handles the missing values, potential biases, and any class imbalances effectively.

Question b.

The variable group has different factor levels in the data and scoring set. How will this difference impact your predictive model? What solutions might you offer?

The presence of different groups in the training and scoring sets can affect a model's ability to generalize effectively. When unseen categories (e.g., "c" in scoring) appear in future data, the model may struggle to predict accurately since it hasn't learned from these groups. This can lead to unreliable predictions, especially if the model relies heavily on categorical features. Overfitting to seen categories (e.g., "d") could further reduce generalization, as the model may become too specialized in patterns specific to the training set. Techniques like one-hot encoding or ordinal encoding can create challenges when categories differ between datasets, as missing or misrepresented categories can skew predictions.

Imputation strategies or the introduction of `NaN` values, followed by one-hot encoding, can help mitigate these issues by allowing the model to treat missing or unseen categories as distinct features. This improves the model's flexibility and ability to handle future data with different group distributions. However, too much imputation or consolidation of categories can reduce the specificity of learned patterns, creating a trade-off between accuracy for seen categories and generalization across unseen ones. Balancing these techniques is crucial for maintaining robust model performance across both current and future datasets.

Possible approaches:

1. One option is to remove the "c" and "d" groups from the dataset entirely. However, this approach is suboptimal, as it would reduce the size of both the training and scoring sets, potentially limiting model performance.

2. Alternatively, we could apply **ordinal encoding** to the "group" field, mapping each category to a numeric value such as: {"a": 0, "b": 1, "c": 2, "d": 3, "e": 4, "f": 5}. This would preserve the ordinal nature of the categories, if applicable.

3. A third option involves **one-hot encoding**, where each group is represented as a binary feature. This approach allows the model to treat the categories as distinct, without imposing any ordinal structure.

4. Another strategy is **category imputation**, where we impute the missing "c" category in the training set and "d" in the scoring set, thereby ensuring consistency across datasets.

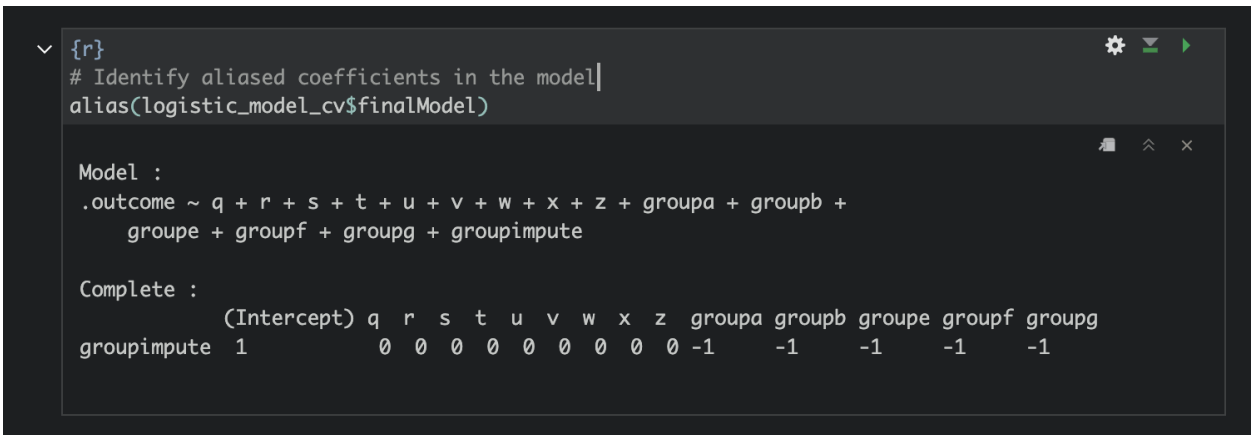
5. We could **combine the "c" and "d" categories** into a single group, as they were statistically indistinguishable based on the results of a Kolmogorov-Smirnov (KS) test. This would reduce the complexity of the categorical variable without losing important distinctions.

6. Another potential approach is to set the "d" values in the original training dataset to NaN and strategically introduce additional NaN values in the "group" field. This selective nullification can help the model generalize better by simulating missing or less certain data. After introducing these NaN values, we can then apply **one-hot encoding** to the "group" field. This allows the model to treat the missing categories as a separate feature, potentially improving its ability to handle unseen categories during scoring or in future data.

Selected approach: 6.

Question c.

Train a model to predict outcome. You are free to use any package in R. Explain the logical progression that leads you to your final solution. Be prepared to interrogate relationships within the model (i.e., how each variable impacts the model and their relationship to the outcome)



```r
{r}
# Identify aliased coefficients in the model
alias(logistic_model_cv$finalModel)
```

```
Model :
.outcome ~ q + r + s + t + u + v + w + x + z + groupa + groupb +
    groupe + groupf + groupg + groupimpute

Complete :
            (Intercept) q  r  s  t  u  v  w  x  z  groupa groupb groupe groupf groupg
groupimpute  1          0  0  0  0  0  0  0  0  0 -1     -1     -1     -1     -1
```

Pre-process the Training Set

```r
dta_complete$group[dta_complete$group == "d"] <- NA

# First, identify the non-NA rows
non_na_rows <- which(!is.na(dta_complete$group))

# Determine 20% of the non-NA rows
```

```r
set.seed(123)  # Set a seed for reproducibility (optional)
num_to_replace <- round(0.2 * length(non_na_rows))

# Randomly select 20% of the non-NA rows
rows_to_replace <- sample(non_na_rows, num_to_replace)

# Replace the selected rows with NA
dta_complete$group[rows_to_replace] <- NA

# Replace NA values in the "group" column with the string "impute"
dta_complete$group[is.na(dta_complete$group)] <- "impute"
```

```r
# Ensure the "group" column is a factor with the specified levels
dta_complete$group <- factor(dta_complete$group, levels = c("a", "b", "e", "f", "g", "impute"))

# One-hot encode the "group" column
one_hot_encoded <- model.matrix( ~ group - 1, data = dta_complete)

# Convert the result to a data frame (optional)
one_hot_encoded_df <- as.data.frame(one_hot_encoded)

# Optionally, you can bind this back to the original dataset
dta_complete_one_hot <- cbind(dta_complete, one_hot_encoded_df)

# Drop the original "group" column from dta_complete_one_hot
dta_complete_one_hot <- dta_complete_one_hot[, !names(dta_complete_one_hot) %in% "group"]

dta_complete_one_hot <- dta_complete_one_hot[, !(colnames(dta_complete_one_hot) %in% "groupimpute")]
```

Pre-process the Scoring Set

```r
scoring_complete$group[scoring_complete$group == "c"] <- NA

# Replace NA values in the "group" column with the string "impute"
scoring_complete$group[is.na(scoring_complete$group)] <- "impute"
```

```r
# Ensure the "group" column is a factor with the specified levels
scoring_complete$group <- factor(scoring_complete$group,
                                 levels = c("a", "b", "e", "f", "g", "impute"))

# One-hot encode the "group" column
one_hot_encoded <- model.matrix(~ group - 1, data = scoring_complete)

# Convert the result to a data frame (optional)
one_hot_encoded_df <- as.data.frame(one_hot_encoded)

# Optionally, you can bind this back to the original dataset
scoring_complete_one_hot <- cbind(scoring_complete, one_hot_encoded_df)

# Drop the original "group" column from scoring_complete_one_hot
scoring_complete_one_hot <- scoring_complete_one_hot[, !names(scoring_complete_one_hot) %in% "group"]
```

```r
scoring_complete_one_hot <- scoring_complete_one_hot[, !(colnames(scoring_complete_one_hot) %in% "group
```

Get a Holdout Set from the Training Set

```r
# Convert the 'outcome' column to a binary factor with levels 0 and 1
dta_complete_one_hot$outcome <- factor(dta_complete_one_hot$outcome, levels = c(0, 1))

# Verify the conversion
str(dta_complete_one_hot$outcome)
```

```
##  Factor w/ 2 levels "0","1": 1 2 1 1 1 1 1 1 2 1 ...
```

```r
# Set a seed for reproducibility (optional)
set.seed(123)

# Step 1: Define the number of rows in the dataset
n <- nrow(dta_complete_one_hot)

# Step 2: Randomly sample 10% of the rows for the holdout set
holdout_indices <- sample(1:n, size = round(0.1 * n))

# Step 3: Create the holdout set (10% of the data)
holdout_set <- dta_complete_one_hot[holdout_indices, ]

# Step 4: Create the training set (remaining 90% of the data)
training_set <- dta_complete_one_hot[-holdout_indices, ]

# Step 5: Inspect the sizes of the training and holdout sets
cat("Training set size:", nrow(training_set), "\n")
```

```
## Training set size: 9000
```

```r
cat("Holdout set size:", nrow(holdout_set), "\n")
```

```
## Holdout set size: 1000
```

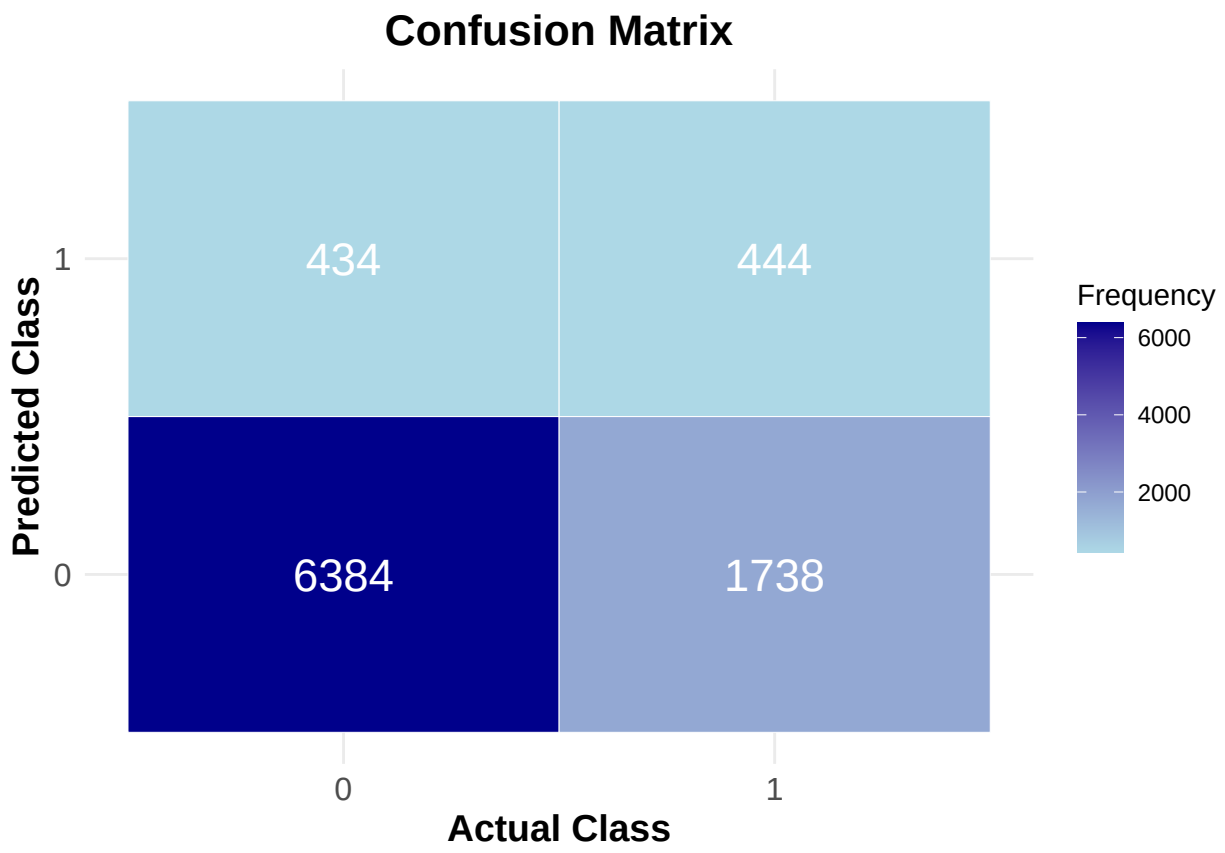
```r
# Step 1: Set up k-fold cross-validation (e.g., 5 folds)
train_control <- trainControl(method = "cv", number = 5)  # 5-fold cross-validation

# Step 2: Fit a logistic regression model using cross-validation
logistic_model_cv <- train(outcome ~ .,
                           data = training_set,
                           method = "glm",
                           family = binomial,
                           trControl = train_control)

# Step 3: View the results of the cross-validation
print(logistic_model_cv)
```

```
## Generalized Linear Model
##
## 9000 samples
##   14 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 7200, 7200, 7200, 7199, 7201
## Resampling results:
##
##   Accuracy   Kappa
##   0.7585547  0.1766811
```

```r
# Step 4: Make predictions on the training set (optional)
predicted_probabilities_cv <- predict(logistic_model_cv, type = "prob")[, 2]

# Step 5: For predicted classes (0 or 1)
predicted_classes_cv <- ifelse(predicted_probabilities_cv > 0.5, 1, 0)

# Step 6: Evaluate model performance (optional)
confusion_matrix_cv <- confusionMatrix(as.factor(predicted_classes_cv), training_set$outcome)

# Print the confusion matrix
print(confusion_matrix_cv)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 6384 1738
##          1  434  444
##
##                Accuracy : 0.7587
##                  95% CI : (0.7497, 0.7675)
##     No Information Rate : 0.7576
##     P-Value [Acc > NIR] : 0.4084
##
##                   Kappa : 0.1755
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9363
##             Specificity : 0.2035
##          Pos Pred Value : 0.7860
##          Neg Pred Value : 0.5057
##              Prevalence : 0.7576
##          Detection Rate : 0.7093
##    Detection Prevalence : 0.9024
##       Balanced Accuracy : 0.5699
##
##        'Positive' Class : 0
##
```

```r
# Extract the confusion matrix table from the caret result
conf_matrix <- as.table(confusion_matrix_cv$table)

# Convert it into a data frame for ggplot
conf_matrix_df <- as.data.frame(conf_matrix)

# Rename columns for readability
colnames(conf_matrix_df) <- c("Prediction", "Reference", "Frequency")

# Create the confusion matrix heatmap
ggplot(conf_matrix_df,
       aes(x = Reference, y = Prediction, fill = Frequency)) +
  geom_tile(color = "white") +  # Use tiles with white borders
  scale_fill_gradient(low = "lightblue", high = "darkblue") +  # Color gradient
  geom_text(aes(label = Frequency),
            color = "white",
            size = 6) +  # Add counts on tiles
  theme_minimal() +  # Use a minimal theme for clean visuals
  labs(title = "Confusion Matrix", x = "Actual Class", y = "Predicted Class") +
  theme(
    plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
    # Centered and bold title
    axis.title.x = element_text(size = 14, face = "bold"),
    axis.title.y = element_text(size = 14, face = "bold"),
    axis.text = element_text(size = 12)  # Larger axis labels for readability
  )
```



Get the Cross-Validated ROC and AUC (Training Set)

```
# Calculate ROC curve
roc_curve <- roc(training_set$outcome, predicted_probabilities_cv)
```

```
## Setting levels: control = 0, case = 1
```

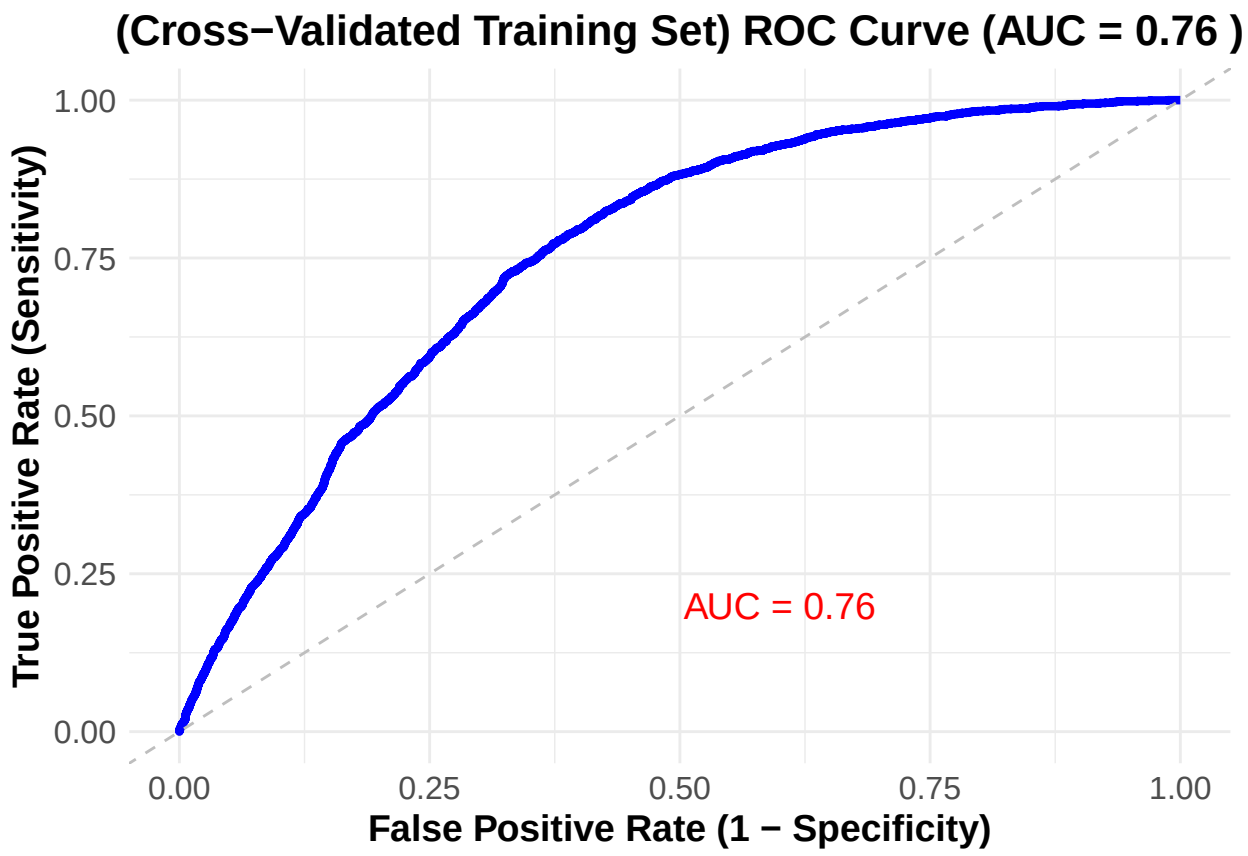
```
## Setting direction: controls < cases
```

```
# Print the AUC value
auc_value <- auc(roc_curve)
print(auc_value)
```

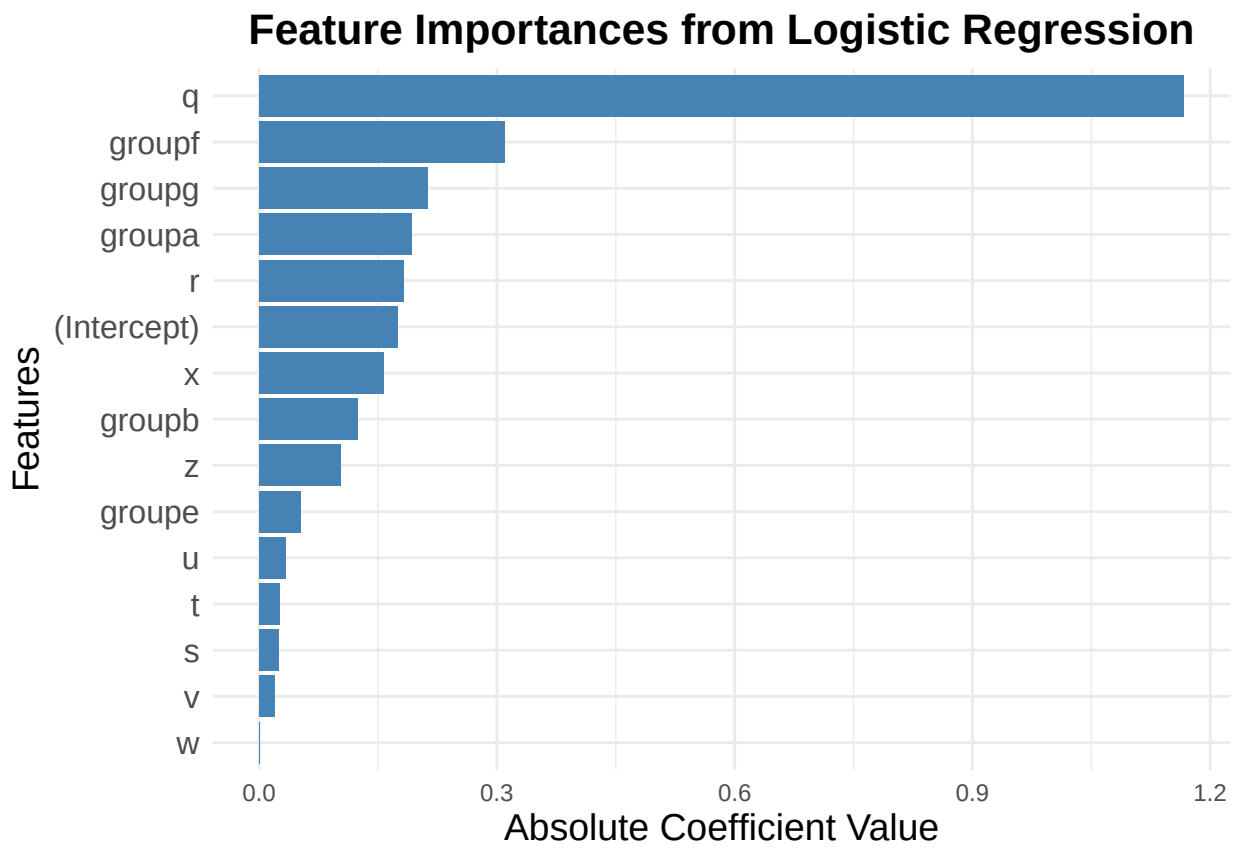
```
## Area under the curve: 0.7558
```

```
# Create a data frame from the ROC curve for plotting
roc_df <- data.frame(
  FalsePositiveRate = 1 - roc_curve$specificities,
  TruePositiveRate = roc_curve$sensitivities
)

# Plot the ROC curve using ggplot2
```

```r
ggplot(roc_df, aes(x = FalsePositiveRate, y = TruePositiveRate)) +
  geom_line(color = "blue", size = 1.5) +  # ROC line
  geom_abline(linetype = "dashed", color = "gray") +  # Diagonal line for random classifier
  labs(
    title = paste("(Cross-Validated Training Set) ROC Curve (AUC =", round(auc_value, 2), ")"),
    x = "False Positive Rate (1 - Specificity)",
    y = "True Positive Rate (Sensitivity)"
  ) +
  theme_minimal() +  # Clean and minimal theme
  theme(
    plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
    # Centered and bold title
    axis.title.x = element_text(size = 14, face = "bold"),
    axis.title.y = element_text(size = 14, face = "bold"),
    axis.text = element_text(size = 12)  # Larger axis labels for readability
  ) +
  annotate(
    "text",
    x = 0.6,
    y = 0.2,
    label = paste("AUC =", round(auc_value, 2)),
    size = 5,
    color = "red"
  )  # AUC annotation
```



```r
# Extracting the coefficients from the logistic model
feature_importances <- coef(logistic_model_cv$finalModel)

# Create a data frame for plotting
feature_importances_df <- data.frame(
  Feature = names(feature_importances),
  Importance = abs(feature_importances)
)

# Sort the features by importance
feature_importances_df <- feature_importances_df[order(-feature_importances_df$Importance), ]

# Create a bar plot using ggplot2
ggplot(feature_importances_df, aes(x = reorder(Feature, Importance), y = Importance)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  coord_flip() +  # Flip coordinates to make the plot horizontal
  theme_minimal() +
  labs(title = "Feature Importances from Logistic Regression", x = "Features", y = "Absolute Coefficien
  theme(
    axis.text.y = element_text(size = 12),
    axis.title = element_text(size = 14),
    plot.title = element_text(size = 16, face = "bold", hjust = 0.5)
  )
```

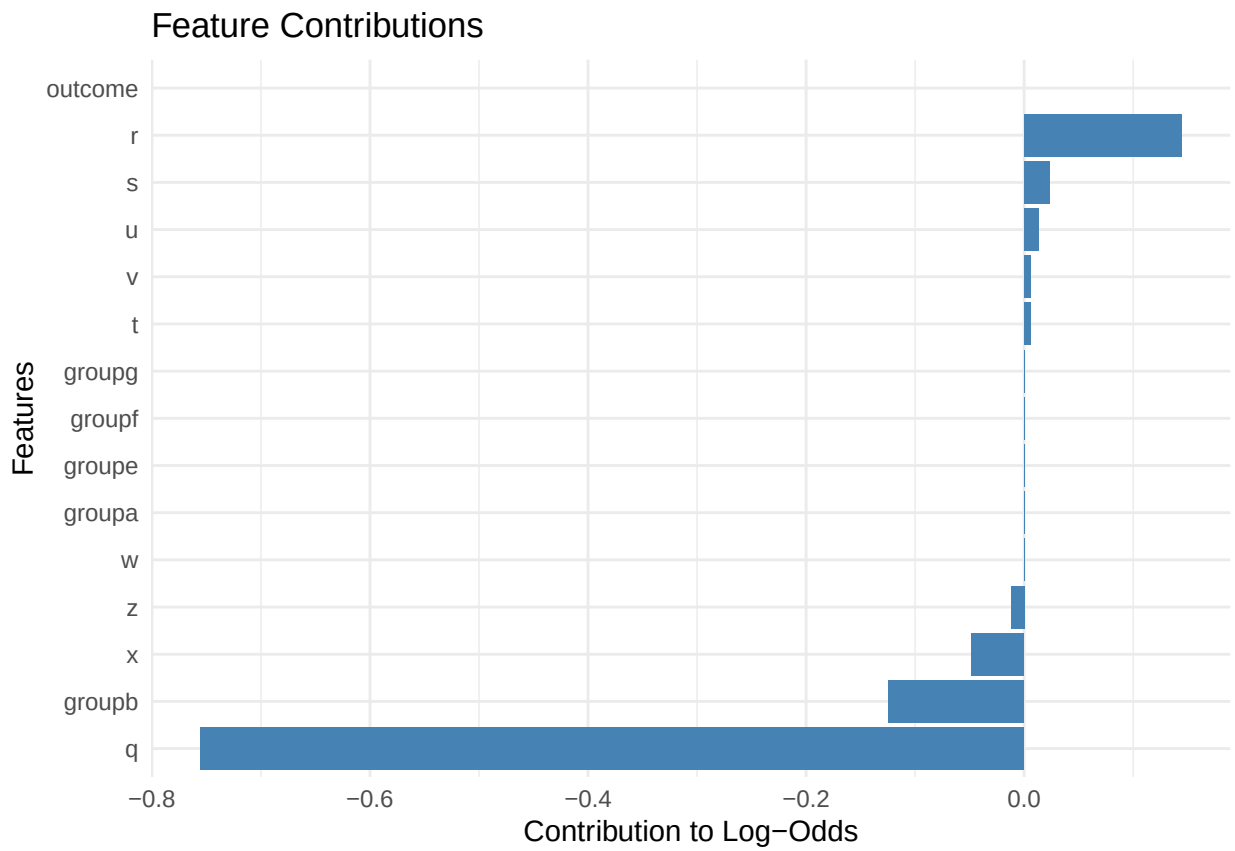


```r
# Predict probabilities using the caret model
predicted_probs <- predict(logistic_model_cv, newdata = training_set, type = "prob")

# Assuming binary classification, extract the probability of the positive class
# Adjust 'Class1' with the actual name of your positive class in the model output
predicted_probs_positive <- predicted_probs[, "1"]

# Extract the feature importances (coefficients)
feature_importances <- coef(logistic_model_cv$finalModel)

# Calculate feature contributions for the first observation
# For each feature, calculate its contribution based on feature importance
contributions <- training_set[1, ] * feature_importances
```

```
## Warning in Ops.factor(left, right): '*' not meaningful for factors
```

```r
# Create a data frame for plotting feature contributions for the first observation
contributions_df <- data.frame(Feature = colnames(training_set),
                               Contributions = as.numeric(contributions))

# Example bar plot for the first observation
ggplot(contributions_df, aes(x = reorder(Feature, Contributions), y = Contributions)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  coord_flip() +
  theme_minimal() +
  labs(title = "Feature Contributions", x = "Features", y = "Contribution to Log-Odds")
```

```
## Warning: Removed 1 row containing missing values or values outside the scale range
## (`geom_bar()`).
```



Top Contributing Features:

- Feature q has the largest negative contribution to the log-odds, meaning it is a strong predictor in favor of the negative class for this specific observation.
- Other features (r, s, and groupb) have positive contributions but are relatively small compared to q.

Question d.

Give an estimate of the AUC you expect on the scoring set. Discuss the choice of AUC as the performance measure (e.g. instead of accuracy).

Get AUC and ROC on the Holdout Set

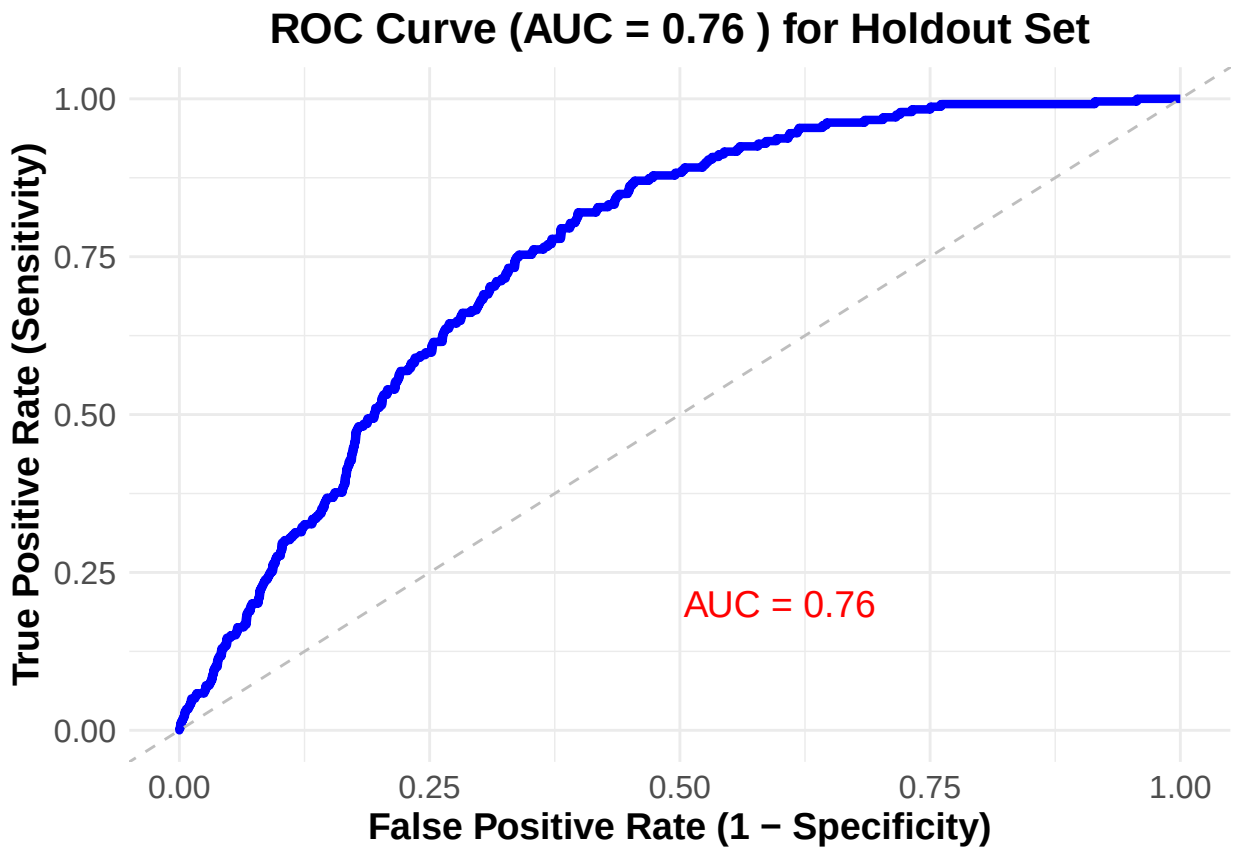
```
# Step 1: Make predictions on the holdout set
# Predict the probabilities for the holdout set using the default response for glm (which gives probabi
predicted_probabilities_holdout <- predict(logistic_model_cv$finalModel,
                                           newdata = holdout_set,
                                           type = "response")

# Step 2: Calculate ROC curve and AUC for the holdout set
roc_curve_holdout <- roc(holdout_set$outcome, predicted_probabilities_holdout)
```

```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```

```
# Print the AUC value for the holdout set
auc_value_holdout <- auc(roc_curve_holdout)
print(paste("AUC for holdout set:", auc_value_holdout))
```

```
## [1] "AUC for holdout set: 0.75723970331924"
```

```
# Step 3: Create a data frame from the ROC curve for plotting
roc_df_holdout <- data.frame(
  FalsePositiveRate = 1 - roc_curve_holdout$specificities,
  TruePositiveRate = roc_curve_holdout$sensitivities
)

# Step 4: Plot the ROC curve for the holdout set using ggplot2
ggplot(roc_df_holdout, aes(x = FalsePositiveRate, y = TruePositiveRate)) +
  geom_line(color = "blue", size = 1.5) +  # ROC line
  geom_abline(linetype = "dashed", color = "gray") +  # Diagonal line for random classifier
  labs(
    title = paste(
      "ROC Curve (AUC =",
      round(auc_value_holdout, 2),
      ") for Holdout Set"
    ),
    x = "False Positive Rate (1 - Specificity)",
    y = "True Positive Rate (Sensitivity)"
  ) +
  theme_minimal() +  # Clean and minimal theme
  theme(
    plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
    # Centered and bold title
    axis.title.x = element_text(size = 14, face = "bold"),
    axis.title.y = element_text(size = 14, face = "bold"),
    axis.text = element_text(size = 12)  # Larger axis labels for readability
  ) +
  annotate(
    "text",
    x = 0.6,
    y = 0.2,
    label = paste("AUC =", round(auc_value_holdout, 2)),
    size = 5,
    color = "red"
  )  # AUC annotation
```



AUC (Area Under the ROC Curve) is a better evaluation metric than accuracy for imbalanced datasets because it considers the performance of a model across all possible classification thresholds, focusing on the trade-off between true positive and false positive rates. Accuracy can be misleading in imbalanced datasets, as it may overestimate performance by focusing on the majority class, ignoring the minority class where the model might be performing poorly. AUC, on the other hand, provides a more balanced assessment by measuring how well the model distinguishes between the positive and negative classes, regardless of their proportions.