Analysis 1 : Rcode

````
---
title: "final project binary"
author: "Deepika Gollamandala"
date: "2025-06-09"
output:
  word_document: default
  html_document: default
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

## R Markdown



```{r, warning = FALSE}
library(survey)
library(sampling)
library('ISLR')
library('ggplot2')
library(MASS)
library(tree)

library(dplyr)
```

```{r}
````
````

load the dataset

yyc_survey <- read.csv("Citizen_Satisfaction_Survey.csv")

#column names of the dataset

names(yyc_survey)

#dim of the dataset

dim(yyc_survey)

```

```{r}

head(yyc_survey)

```

```{r}

table(yyc_survey$qwave)

```

We only want to use 2021 survey data:

```{r}

library(dplyr)

# filter to include only the survey responses from year 2021

filtered_df <- yyc_survey %>% filter(qwave == 'Year-2021')

# checking to ensure we only have 2021 survey data

dim(filtered_df)

unique(filtered_df$qwave)

```
```

Select the columns to include only demographic features and response variable:

```{r}
filtered_df <- filtered_df %>% select(s4qt, q39, q34, q37, q38, q32x, q40, q29x, q30, q2a)


```

```{r}
dim(filtered_df)
```

Rename the column names for ease.

```{r}
renamed_df <- filtered_df %>% rename("Quadrant" = s4qt, "Income" = q39, "Tenancy" = q34,
"Years_in_yyc" = q37, "Education" = q38, "Children" = q32x, "Minority" = q40, "Gender" = q29x, "Age"
= q30, "Satisfaction_level" = q2a )


```

```{r}
# check the column names
names(renamed_df)
```

```{r}
# check the data types
str(renamed_df)
```

Check the levels in our response variable

```{r}
table(renamed_df$Satisfaction_level)
```

```

```{r}
table(renamed_df$Age)

table(renamed_df$Years_in_yyc)

table(renamed_df$Education)

table(renamed_df$Income)

sum(table(renamed_df$Income))
```

We want to remove the one record which has a value of 11 for quality of life. A score of 11 meant that the respondent does'nt know what to rate the quality of life in Calgary

```{r}
#remove the row with satisfaction Rating of "11"

renamed_df <- renamed_df %>% filter(Satisfaction_level != 11)


# inspect the levels

table(renamed_df$Satisfaction_level)
```

transform the Satisfaction_level column into a binary column with levels "Yes" and "No". The new column name is "Satisfaction". When the value of Satisfaction_level is <= 5, value is "Not Satisfied" or "No" and when Satisfaction_level is >5, it is "Satisfied" or "Yes"

```{r}
demographic_df <- renamed_df %>% mutate(Satisfaction = ifelse(Satisfaction_level <= 5, "No", "Yes"))


# check the column names

names(demographic_df)


# check the levels of our new column.

table(demographic_df$Satisfaction)

```

```{r}

str(demographic_df)

```

All our exploratory variables are categorical but are in wrong datatype int. We need to first convert them to be categorical.

```{r}

demographic_df <- demographic_df %>%

  mutate(across(where(is.integer), as.factor))

str(demographic_df)

```

Converting our response variable into factor

```{r}

demographic_df <- demographic_df %>% mutate(Satisfaction = factor(Satisfaction, levels = c("No", "Yes")))

str(demographic_df$Satisfaction)

```

Check the class proportion in the dataset

```{r}

table(demographic_df$Satisfaction)

prop.table(table(demographic_df$Satisfaction))

```

Splitting the data into train and test. We use stratified sampling to ensure that both train and test datasets have a similar proportion of "No" and "Yes" in Satisfation.

```{r warning = FALSE}
library(caret)
set.seed(2024)

# Create stratified split (70% train, 30% test)
train_index <- createDataPartition(demographic_df$Satisfaction, p = 0.7, list = FALSE)

# Split data
train_data <- demographic_df[train_index, ]
test_data <- demographic_df[-train_index, ]

```

```{r}
# To ensure we have the same proportion of classes of our response variable in test and train

prop.table(table(train_data$Satisfaction))
prop.table(table(test_data$Satisfaction))

```
```{r}
# Check the dimensions of train and test
dim(train_data)
dim(test_data)
```
```

Applying logistic model

```{r}
model1 <- glm(Satisfaction ~ .-Satisfaction_level, data = train_data, family = binomial)

summary(model1)
```

```{r}
# making predictions on the test set

pred1 <- predict(model1, test_data, type = "response" )
```

```{r}
# setting the probability threshold.

predicted_class <- ifelse(pred1 > 0.5, "Yes", "No")

head(predicted_class)
```

```{r}
# Accuracy and classification performance

Actuals <- test_data$Satisfaction

table(predicted_class, Actuals)


Accuracy <- mean(predicted_class == Actuals)

Accuracy
```

Considering only the significant variables and constructing a model

```{r}
```

model_sgx <- glm(Satisfaction ~ . - Satisfaction_level -Quadrant -Children -Minority , data = train_data, family = binomial)

summary(model_sgx)

```

```{r}

# predicting from the model with only significant variables

pred_sgx <- predict(model_sgx, test_data, type = "response" )

predicted_class_sgx <- ifelse(pred_sgx > 0.5, "Yes", "No")

Actual <- test_data$Satisfaction

table(predicted_class_sgx, Actual)

Accuracy_sgx <- mean(predicted_class_sgx == test_data$Satisfaction)

Accuracy_sgx

```

To capture more of "No":

Increased the weight of "No" and also increase the threshhold level to 0.7 to capture more "No"

```{r}

model2 <- glm(Satisfaction ~ . - Satisfaction_level-Minority -Children - Quadrant, data = train_data, family = binomial, weights = ifelse(train_data$Satisfaction == "No", 2, 1))

```

```{r}

pred2 <- predict(model2, test_data, type = "response" )

predicted_incNO <- ifelse(pred2 > 0.75, "Yes", "No")

Actuals <- test_data$Satisfaction

table(predicted_incNO, Actuals)

Accuracy2 <- mean(predicted_incNO == Actuals)

Accuracy2

```

Upsampling the train data

```{r}

set.seed(2024)

library(sampling)

library(survey)

idx <- sampling:::strata(train_data, stratanames = c("Satisfaction"), size = c(2500,2000), method = "srswr")

```

```{r}

train_data_upsampled <- train_data[idx$ID_unit, ]

testing_data <- test_data

dim(train_data)     # original train set

dim(train_data_upsampled)  # upsampled train set.

```

```{r}

```
dim(testing_data)

dim(train_data_upsampled)

table(train_data_upsampled$Satisfaction)

table(testing_data$Satisfaction)
```

Fitting the model on upsampled data

```{r}
model3 <- glm(Satisfaction ~ . - Satisfaction_level-Quadrant -Children -Minority , data = train_data_upsampled, family = binomial)


```

Making predictions on test data that has the class proportions of original dataset.

```{r}
pred3 <- predict(model3, testing_data, type = "response" )

predicted_upsamp <- ifelse(pred3 > 0.5, "Yes", "No")

Actual <- testing_data$Satisfaction

table(predicted_upsamp, Actual)


Accuracy3 <- mean(predicted_upsamp == testing_data$Satisfaction)

Accuracy3
```

```{r}
precision <- posPredValue(factor(predicted_upsamp, levels = c("No", "Yes")), testing_data$Satisfaction)

recall <- sensitivity(factor(predicted_upsamp, levels = c("No", "Yes")), testing_data$Satisfaction)
```

print(precision)

print(recall)

```

## K-fold cross validation

```{r}

set.seed(2024)

folds<-createFolds(demographic_df$Satisfaction, k=10)

```

CV error for Logistic model from upsampled train data

```{r}

library(MASS)

log_misclassification <- c()

for (i in 1:10) {

  trainIndex <- unlist(folds[-i])

  testIndex <- unlist(folds[i])

  trainData <- demographic_df[trainIndex, ]

  testData <- demographic_df[testIndex, ]

  idx2 <- sampling:::strata(trainData, stratanames = c("Satisfaction"), size = c(2500,2000), method = "srswr")

  trainData_upsampled <- trainData[idx2$ID_unit, ]

  log_cv_upsampled <- glm(Satisfaction ~ . - Satisfaction_level-Quadrant -Children -Minority , data = trainData_upsampled, family = binomial)

  pred_log_cv <- predict(log_cv_upsampled, testData, type = "response" )

  predicted_satis <- ifelse(pred_log_cv > 0.5, "Yes", "No")

```
  log_misclassification[i] <- mean(predicted_satis != testData$Satisfaction)
}

log_cv_error <- mean(log_misclassification)
cat("The cross validation error for log model is: ", log_cv_error)
log_misclassification
```

CV accuracy for Logistic model from upsampled train data

```{r warning = FALSE}
library(MASS)
log_accuracy <- c()
for (i in 1:10) {
  trainIndex <- unlist(folds[-i])
  testIndex <- unlist(folds[i])

  trainData <- demographic_df[trainIndex, ]
  testData <- demographic_df[testIndex, ]

  idx2 <- sampling:::strata(trainData, stratanames = c("Satisfaction"), size = c(2500,2000), method = "srswr")
  trainData_upsampled <- trainData[idx2$ID_unit, ]

  log_cv_upsampled <- glm(Satisfaction ~ . - Satisfaction_level-Quadrant -Children -Minority , data = trainData_upsampled, family = binomial)

  pred_log_cv <- predict(log_cv_upsampled, testData, type = "response" )
  predicted_satis <- ifelse(pred_log_cv > 0.5, "Yes", "No")
```

log_accuracy[i] <- mean(predicted_satis == testData$Satisfaction)

print(table(predicted_satis, testData$Satisfaction))

print(log_accuracy[i])

}

log_cv_accuracy <- mean(log_accuracy)

cat("The cross validation accuracy for log model is: ", log_cv_accuracy, "\n")

log_accuracy

` ` `

TREE MODEL

` ` `{r}

Tree model

library(tree)

fit the model on train data

class_tree_model <- tree(Satisfaction ~.-Satisfaction_level, data = train_data)

plot the tree

summary(class_tree_model)

plot(class_tree_model)

text(class_tree_model, pretty = 0)

` ` `

` ` `{r}

Predict Satisfaction on test set

````
tree_pred <- predict(class_tree_model, test_data, type = "class")

table(tree_pred, test_data$Satisfaction)

# Compute accuracy
conf_matrix <- confusionMatrix(tree_pred, test_data$Satisfaction)

# Print accuracy
print(conf_matrix$overall["Accuracy"])

```

```{r}
set.seed(2024)
cv.class <- cv.tree(class_tree_model, FUN = prune.misclass, K = 10)
plot(cv.class$size, cv.class$dev, type = "b")
```

```{r}
# fit the model on upsampled train data
tree_model2 <- tree(Satisfaction ~.-Satisfaction_level, data = train_data_upsampled)



# plot the tree
summary(tree_model2)
plot(tree_model2)
text(tree_model2, pretty = 0)
```
````
````

````
```{r}
# Predict Satisfaction on test set
tree_upsamp <- predict(tree_model2, test_data, type = "class")

table(tree_upsamp, test_data$Satisfaction)

Accuracy = mean(tree_upsamp == test_data$Satisfaction)
Accuracy

```
```{r}
cv.class2 = cv.tree(tree_model2, FUN=prune.misclass)
plot(cv.class2$size,cv.class2$dev, type="b")
```

```{r}
prune.satisfaction = prune.tree(tree_model2, best = 3)
plot(prune.satisfaction)
text(prune.satisfaction, pretty = 0)
```

```{r}
satisfaction_pruned = predict(prune.satisfaction, test_data, type = "class")

table(satisfaction_pruned, test_data$Satisfaction)
accuracy <- mean(satisfaction_pruned == test_data$Satisfaction)
accuracy
```
````