R-code Analysis – 2

~~~
```{r}
library(readxl)
library(caret)
library(dplyr)
survey <- read_excel("D:/survey.xlsx")
head(survey)
```
```{r}
dim(survey)
```

```{r}
data <- na.omit(survey)
```



```{r}
library(dplyr)

survey <- survey %>%
  mutate(satisfaction = case_when(
    `Overall Quality of Life` >= 1 & `Overall Quality of Life` <= 5 ~ 0,
    `Overall Quality of Life` > 5 & `Overall Quality of Life` <= 10 ~ 1
  ))

```

```{r}
head(survey)
```
```{r}
dim(survey)
```
```{r}
survey <- survey %>% select(-`Overall Quality of Life`)
```

```{r}
lapply(survey, function(col) unique(col))
```
~~~

~~~
```{r}
# Step 0: Ensure satisfaction is factor with levels 0 and 1
survey$satisfaction <- factor(survey$satisfaction, levels = c(0, 1))

# Step 1: Create test set with equal number of class 0 and class 1
set.seed(123)

# Find indices for each class
class0_idx <- which(survey$satisfaction == 0)
class1_idx <- which(survey$satisfaction == 1)

# Number of samples for each class in test set (use smaller class size)
n_test <- min(length(class0_idx), length(class1_idx), 100)  # limit if needed

test_idx_0 <- sample(class0_idx, n_test)
test_idx_1 <- sample(class1_idx, n_test)

test_idx <- c(test_idx_0, test_idx_1)
test_data <- survey[test_idx, ]

# Step 2: Create train set from remaining data
train_data <- survey[-test_idx, ]

# Step 3: Upsample the minority class in training data
set.seed(123)
train_balanced <- upSample(x = train_data %>% select(-satisfaction),
                           y = train_data$satisfaction,
                           yname = "satisfaction")
train_data<-train_balanced
# Check balance
cat("Test set class balance:\n")
print(prop.table(table(test_data$satisfaction)))

cat("\nTraining set class balance (after upsampling):\n")
print(prop.table(table(train_balanced$satisfaction)))

```
```{r}
prop.table(table(survey$satisfaction))         # original distribution
prop.table(table(train_data$satisfaction))   # training distribution
prop.table(table(test_data$satisfaction))    # testing distribution

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

~~~
logit_model <- glm(satisfaction ~ ., data = train_balanced, family = binomial)

summary(logit_model)


```
```{r}
prob <- predict(logit_model, newdata = test_data, type = "response")


pred <- ifelse(prob >= 0.5, 1, 0)
pred <- factor(pred, levels = c(0, 1))  # make sure levels match

conf_mat <- confusionMatrix(pred, test_data$satisfaction)

print(conf_mat)
```

```{r}
# Variable importance
importance <- varImp(logit_model)
print(importance)

```
```{r}
library(ggplot2)

# Convert to data frame for plotting
importance_df <- as.data.frame(importance)
importance_df$Variable <- rownames(importance_df)

# Plot
ggplot(importance_df, aes(x = reorder(Variable, Overall), y = Overall)) +
  geom_col(fill = "steelblue") +
  coord_flip() +
  labs(title = "Variable Importance", x = "Variable", y = "Importance")

```
```{r}
library(car)
vif(logit_model)
```
```{r}
# Extract coefficients from summary
~~~

~~~
coefs_summary <- summary(logit_model)$coefficients

# Remove intercept row
coefs_summary <- coefs_summary[rownames(coefs_summary) != "(Intercept)", ]

# Extract p-values
p_values <- coefs_summary[, "Pr(>|z|)"]

# Get variable names with p-value < 0.05
significant_vars <- names(p_values[p_values < 0.05])

# Print them
print(" Significant predictors (p < 0.05):")
print(significant_vars)

```


```{r}
# Build formula with only significant predictors
formula_significant <- as.formula(paste("satisfaction ~", paste(significant_vars, collapse = " +
")))
model_sig <- glm(formula_significant, data = train_balanced, family = binomial)

# View summary
summary(model_sig)

```
```{r}
# Extract coefficients from summary
coefs_summary <- summary(model_sig)$coefficients

# Remove intercept row
coefs_summary <- coefs_summary[rownames(coefs_summary) != "(Intercept)", ]

# Extract p-values
p_values <- coefs_summary[, "Pr(>|z|)"]

# Get variable names with p-value < 0.05
significant_vars <- names(p_values[p_values < 0.05])

# Print them
print(" Significant predictors (p < 0.05):")
print(significant_vars)
~~~

~~~
```
```{r}
# Build formula with only significant predictors
#  This is correct

formula_significant <- as.formula(paste("satisfaction ~", paste(significant_vars, collapse = " +
")))
model_sig_2 <- glm(formula_significant, data = train_balanced, family = binomial)

# View summary
summary(model_sig_2)

```


```{r}
# Predict probabilities
pred_probs <- predict(model_sig_2, newdata = test_data, type = "response")

# Convert to class labels (0 or 1)
pred_class <- ifelse(pred_probs >= 0.5, 1, 0)

# Convert both to factors with the same levels
pred_class <- factor(pred_class, levels = c(0, 1))
true_class <- factor(test_data$satisfaction, levels = c(0, 1))

# Evaluate
library(caret)
confusionMatrix(data = pred_class, reference = true_class)

```
```{r}
# Variable importance
importance <- varImp(model_sig_2)
print(importance)

```
```{r}
library(ggplot2)

# Convert to data frame for plotting
importance_df <- as.data.frame(importance)
importance_df$Variable <- rownames(importance_df)

# Plot
~~~

~~~
ggplot(importance_df, aes(x = reorder(Variable, Overall), y = Overall)) +
  geom_col(fill = "steelblue") +
  coord_flip() +
  labs(title = "Variable Importance", x = "Variable", y = "Importance")

```


```{r}
vif(model_sig_2)
```
```{r}
tree.satisfaction<-tree(satisfaction ~ ., train_data)
summary(tree.satisfaction)

```
```{r}
plot(tree.satisfaction)
text(tree.satisfaction,pretty=0)
```



```{r}
type.pred <- predict(tree.satisfaction, newdata = test_data, type = "class")
misclass_rate <- mean(type.pred != test_data$satisfaction)
cat("Misclassification Rate on Test Set:", misclass_rate, "\n")
accuracy<-mean(type.pred == test_data$satisfaction)
accuracy
```
```{r}
cv.satisfaction<-cv.tree(tree.satisfaction)
plot(cv.satisfaction$size,cv.satisfaction$dev,type="b")
```
```{r}
prune.satisfaction<-prune.tree(tree.satisfaction,best=25)
plot(prune.satisfaction)
text(prune.satisfaction, pretty=0)
```
```{r}
type.pred <- predict(prune.satisfaction, newdata = test_data, type = "class")
misclass_rate <- mean(type.pred != test_data$satisfaction)
cat("✅ Misclassification Rate on Test Set:", misclass_rate, "\n")
accuracy<-mean(type.pred == test_data$satisfaction)
accuracy
```
~~~

~~~
```{r}
confusionMatrix(data = type.pred, reference = test_data$satisfaction)
~~~