VIRGINIA COMMONWEALTH UNIVERSITY

Statistical analysis and modelling (SCMA 632)

A3: Limited dependent variable Models - Classification Analysis

AAKASH K

V01110153

Date of Submission: 01-07-2024

CONTENTS

Introduction:

The assignment involves conducting various regression analyses on the “kidney disease.csv” and "NSSO68.csv" datasets.

The Kidney Disease dataset is a comprehensive collection of medical and demographic data used for analyzing and predicting kidney disease. This dataset provides a valuable resource for developing and evaluating machine learning models aimed at diagnosing kidney-related conditions. It includes various features related to patient health, laboratory tests, and demographic information.

The NSSO68 dataset refers to data from the 68th round of the National Sample Survey Office (NSSO) in India, which typically collects detailed socio-economic information from households across the country. This dataset is part of the larger suite of datasets produced by the NSSO, which aims to provide insights into various aspects of Indian life, including economic conditions,consumption patterns, and employment.

Objective:

1. **Part A** - Conduct a logistic regression analysis on the “kidney_disease” dataset. Validate assumptions, evaluate with a confusion matrix and ROC curve, and interpret the results. Then, perform a decision tree analysis and compare it to the logistic regression.

2. **Part B** - Perform a probit regression on “NSSO68.csv” to identify non-vegetarians. Discuss the results and explain the characteristics and advantages of the probit model.

3. **Part C** - Perform a Tobit regression analysis on “NSSO68.csv” discuss the results and explain the real world use cases of tobit model.

Business Significance:

1. **Part A:** Logistic regression on the “kidney_disease” dataset helps identify key risk factors for CKD, improving patient screening and treatment. The confusion matrix and ROC curve validate model accuracy, while comparing with decision trees provides additional insights for better risk assessment and healthcare interventions.

2. **Part B:** Probit regression on “NSSO68.csv” aids in identifying non-vegetarians by handling binary outcomes effectively. This analysis enables targeted marketing and product optimization by understanding dietary preferences, enhancing strategic business decisions.

3. **Part C**: Tobit regression on “NSSO68.csv” addresses censored data, offering insights into economic behaviors such as income and expenditure. This model is crucial for accurate policy-making and resource allocation by handling data limitations and providing precise estimates.

   Overall, these analyses support data-driven decision-making, improving strategic planning, operational efficiency, and targeted marketing in various sectors.

R code results:

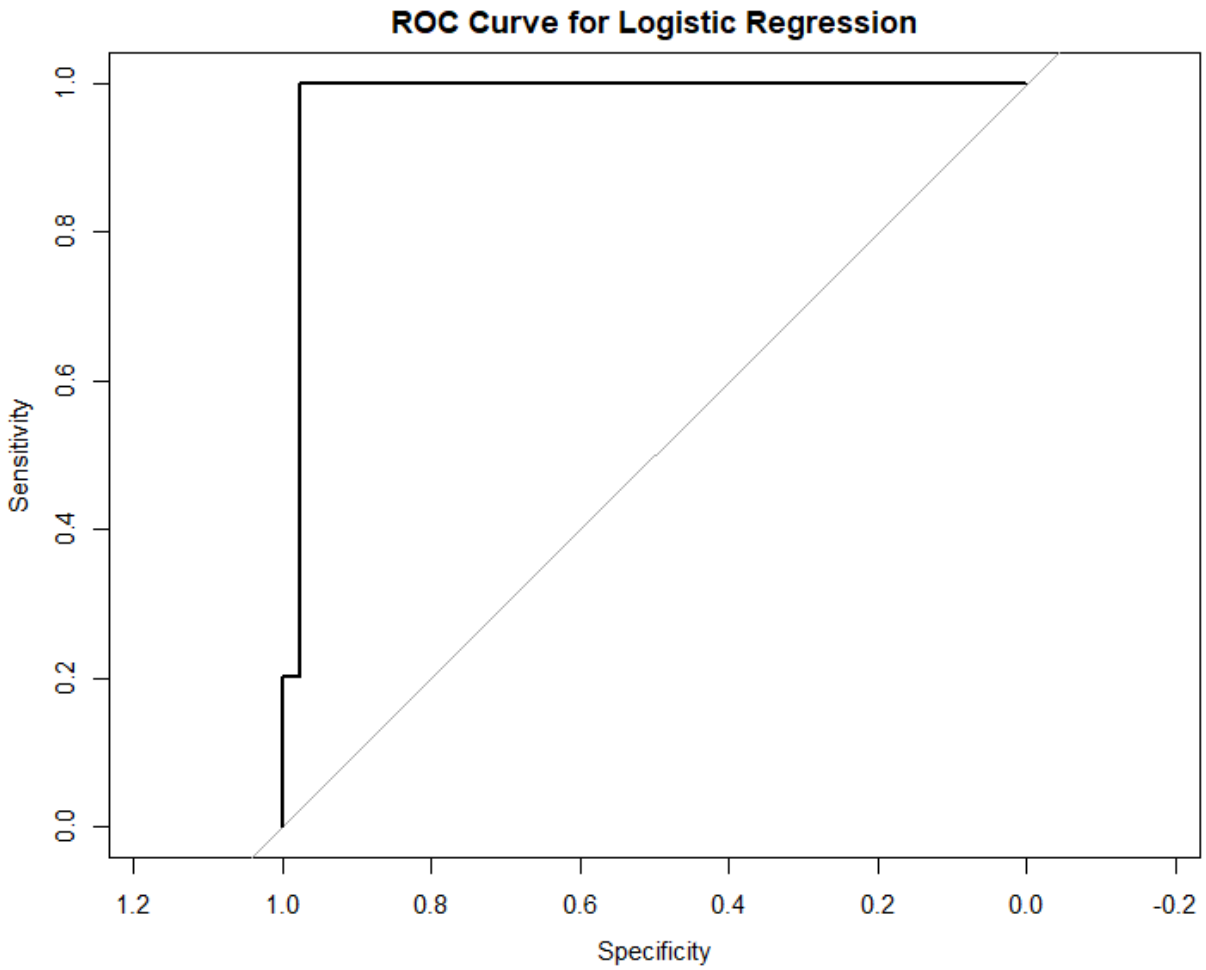
Part A:

```
> # Load necessary libraries
> library(caret)
> library(pROC)
> library(rpart)
> library(rpart.plot)
> library(glmnet)  # For regularization
Loading required package: Matrix
Loaded glmnet 4.1-8
Warning message:
package ‘glmnet’ was built under R version 4.3.3
> # Load your dataset
> df <- read.csv("C:/Users/Aakash/Desktop/SCMA/kidney_disease.csv")
> # Display the first few rows of the dataset
> print(head(df))
  X age bp    sg al su    rbc       pc        pcc         ba bgr bu  sc sod pot hemo pcv wbcc rbcc htn  dm cad appet
1 0  48 80 1.020  1  0          normal notpresent notpresent 121 36 1.2  NA  NA 15.4  44 7800  5.2 yes yes  no  good
2 1   7 50 1.020  4  0          normal notpresent notpresent  NA 18 0.8  NA  NA 11.3  38 6000   NA  no  no  no  good
3 2  62 80 1.010  2  3 normal   normal notpresent notpresent 423 53 1.8  NA  NA  9.6  31 7500   NA  no yes  no  poor
4 3  48 70 1.005  4  0 normal abnormal    present notpresent 117 56 3.8 111 2.5 11.2  32 6700  3.9 yes  no  no  poor
5 4  51 80 1.010  2  0 normal   normal notpresent notpresent 106 26 1.4  NA  NA 11.6  35 7300  4.6  no  no  no  good
6 5  60 90 1.015  3  0                 notpresent notpresent  74 25 1.1 142 3.2 12.2  39 7800  4.4 yes yes  no  good
   pe ane class
1  no  no   ckd
2  no  no   ckd
3  no yes   ckd
4 yes yes   ckd
5  no  no   ckd
6 yes  no   ckd
> # Summary statistics of the dataset
> print(summary(df))
       X               age              bp               sg               al               su
 Min.   :  0.00   Min.   : 2.00   Min.   : 50.00   Min.   :1.005   Min.   :0.000   Min.   :0.0000
 1st Qu.: 99.75   1st Qu.:42.00   1st Qu.: 70.00   1st Qu.:1.010   1st Qu.:0.000   1st Qu.:0.0000
 Median :199.50   Median :55.00   Median : 80.00   Median :1.020   Median :0.000   Median :0.0000
 Mean   :199.50   Mean   :51.48   Mean   : 76.47   Mean   :1.017   Mean   :1.017   Mean   :0.4501
 3rd Qu.:299.25   3rd Qu.:64.50   3rd Qu.: 80.00   3rd Qu.:1.020   3rd Qu.:2.000   3rd Qu.:0.0000
 Max.   :399.00   Max.   :90.00   Max.   :180.00   Max.   :1.025   Max.   :5.000   Max.   :5.0000
                  NA's   :9       NA's   :12       NA's   :47      NA's   :46      NA's   :49
     rbc                pc                pcc                ba                bgr              bu
 Length:400         Length:400         Length:400         Length:400         Min.   : 22   Min.   :  1.50
 Class :character   Class :character   Class :character   Class :character   1st Qu.: 99   1st Qu.: 27.00
 Mode  :character   Mode  :character   Mode  :character   Mode  :character   Median :121   Median : 42.00
                                                                             Mean   :148   Mean   : 57.43
                                                                             3rd Qu.:163   3rd Qu.: 66.00
                                                                             Max.   :490   Max.   :391.00
                                                                             NA's   :44    NA's   :19
       sc               sod              pot              hemo             pcv              wbcc            rbcc
 Min.   : 0.400   Min.   :  4.5   Min.   : 2.500   Min.   : 3.10   Min.   : 9.00   Min.   : 2200   Min.   :2.100
 1st Qu.: 0.900   1st Qu.:135.0   1st Qu.: 3.800   1st Qu.:10.30   1st Qu.:32.00   1st Qu.: 6500   1st Qu.:3.900
 Median : 1.300   Median :138.0   Median : 4.400   Median :12.65   Median :40.00   Median : 8000   Median :4.800
 Mean   : 3.072   Mean   :137.5   Mean   : 4.627   Mean   :12.53   Mean   :38.88   Mean   : 8406   Mean   :4.707
 3rd Qu.: 2.800   3rd Qu.:142.0   3rd Qu.: 4.900   3rd Qu.:15.00   3rd Qu.:45.00   3rd Qu.: 9800   3rd Qu.:5.400
 Max.   :76.000   Max.   :163.0   Max.   :47.000   Max.   :17.80   Max.   :54.00   Max.   :26400   Max.   :8.000
 NA's   :17       NA's   :87      NA's   :88       NA's   :52      NA's   :71      NA's   :106     NA's   :131
     htn                dm                cad               appet                pe                ane
 Length:400         Length:400         Length:400         Length:400         Length:400         Length:400
 Class :character   Class :character   Class :character   Class :character   Class :character   Class :character
 Mode  :character   Mode  :character   Mode  :character   Mode  :character   Mode  :character   Mode  :character


    class
 Length:400
 Class :character
 Mode  :character
```

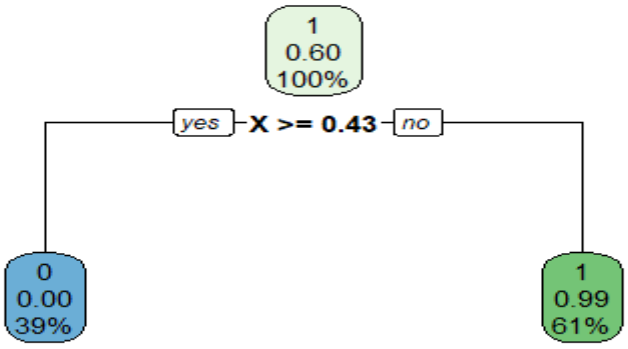
```
> # Check for missing values
> cat("Total missing values: ", sum(is.na(df)), "\n")
Total missing values:  778
> # Custom function to calculate mode
> mode_function <- function(x) {
+   uniq_x <- unique(x)
+   uniq_x[which.max(tabulate(match(x, uniq_x)))]
+ }
> # Function to impute missing values
> impute_missing_values <- function(df) {
+   for (col in names(df)) {
+     if (is.numeric(df[[col]])) {
+       df[[col]][is.na(df[[col]])] <- median(df[[col]], na.rm = TRUE)
+     } else {
+       df[[col]][is.na(df[[col]])] <- mode_function(df[[col]][!is.na(df[[col]])])
+     }
+   }
+   return(df)
+ }
> # Impute missing values
> df <- impute_missing_values(df)
> # Verify there are no more missing values
> cat("Total missing values after imputation: ", sum(is.na(df)), "\n")
Total missing values after imputation:  0
> # Ensure the target variable is a factor with exactly two levels
> df$class <- as.factor(df$class)
> # Convert target variable to numeric (1 for "ckd" and 0 for "notckd")
> df$class <- ifelse(df$class == "ckd", 1, 0)
> # Feature scaling
> preProc <- preProcess(df[, -which(names(df) == "class")], method = c("center", "scale"))
> scaled_data <- predict(preProc, df[, -which(names(df) == "class")])
> df_scaled <- cbind(scaled_data, class = df$class)
> # Split the data into training and testing sets
> set.seed(123)
> trainIndex <- createDataPartition(df_scaled$class, p = 0.7, list = FALSE)
> trainData <- df_scaled[trainIndex,]
> testData <- df_scaled[-trainIndex,]
> # Check the distribution of the target variable in training and testing sets
> cat("Training set distribution:\n")
Training set distribution:
> print(table(trainData$class))

  0   1
111 169
> cat("Testing set distribution:\n")
Testing set distribution:
> print(table(testData$class))

 0  1
41 79
```

```
> # Prepare data for glmnet (regularized logistic regression)
> x_train <- as.matrix(trainData[, -which(names(trainData) == "class")])
> x_test <- as.matrix(testData[, -which(names(testData) == "class")])
> y_train <- as.numeric(trainData$class)
> y_test <- as.numeric(testData$class)
> # Ensure all features are numeric in x_train
> non_numeric_columns <- colnames(trainData)[!sapply(trainData, is.numeric)]
> cat("Non-numeric columns in trainData: ", non_numeric_columns, "\n")
Non-numeric columns in trainData:  rbc pc pcc ba htn dm cad appet pe ane
> # If there are non-numeric columns, convert them to numeric
> for (col in non_numeric_columns) {
+   suppressWarnings({
+     trainData[[col]] <- as.numeric(as.character(trainData[[col]]))
+     testData[[col]] <- as.numeric(as.character(testData[[col]]))
+   })
+ }
> # Re-prepare the matrices after conversion
> x_train <- as.matrix(trainData[, -which(names(trainData) == "class")])
> x_test <- as.matrix(testData[, -which(names(testData) == "class")])
> # Check for NA values in the matrices and vectors
> cat("Any NA in x_train after conversion: ", any(is.na(x_train)), "\n")
Any NA in x_train after conversion:  TRUE
> cat("Any NA in x_test after conversion: ", any(is.na(x_test)), "\n")
Any NA in x_test after conversion:  TRUE
> # If there are still missing values, use makeX() to impute them
> x_train[is.na(x_train)] <- 0
> x_test[is.na(x_test)] <- 0
> # Logistic Regression Model with Regularization
> log_model <- glmnet(x_train, y_train, family = "binomial")
> # Cross-validation to select the best lambda
> cv_log_model <- cv.glmnet(x_train, y_train, family = "binomial")
> # Predict on the test set using the best lambda
> log_pred <- predict(cv_log_model, newx = x_test, s = "lambda.min", type = "response")
> log_pred <- as.vector(log_pred)  # Ensure log_pred is a numeric vector
> log_pred_class <- ifelse(log_pred > 0.5, 1, 0)
```

```
> # Confusion Matrix for Logistic Regression
> log_conf_matrix <- confusionMatrix(as.factor(log_pred_class), as.factor(y_test))
> cat("Confusion Matrix for Logistic Regression:\n")
Confusion Matrix for Logistic Regression:
> print(log_conf_matrix)
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 40  0
         1  1 79

               Accuracy : 0.9917
                 95% CI : (0.9544, 0.9998)
    No Information Rate : 0.6583
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9814

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9756
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.9875
             Prevalence : 0.3417
         Detection Rate : 0.3333
   Detection Prevalence : 0.3333
      Balanced Accuracy : 0.9878

       'Positive' Class : 0

> # ROC Curve for Logistic Regression
> roc_log <- roc(y_test, log_pred)
Setting levels: control = 0, case = 1
Setting direction: controls < cases
> plot(roc_log, main = "ROC Curve for Logistic Regression", col = "black")
> cat("AUC for Logistic Regression: ", auc(roc_log), "\n")
```

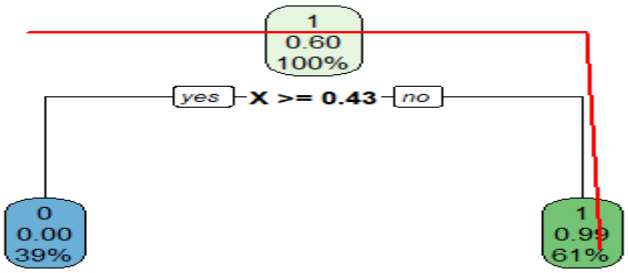



```
> cat("AUC for Logistic Regression: ", auc(roc_log), "\n")
AUC for Logistic Regression:  0.9805496
> # Decision Tree Model
> tree_model <- rpart(class ~ ., data = trainData, method = "class")
> # Plot Decision Tree
> rpart.plot(tree_model)
```



```
> # Predict on the test set using Decision Tree
> tree_pred <- predict(tree_model, newdata = testData, type = "class")
> tree_pred_prob <- predict(tree_model, newdata = testData, type = "prob")[, 2]
> # Confusion Matrix for Decision Tree
> tree_conf_matrix <- confusionMatrix(tree_pred, as.factor(y_test))
> cat("Confusion Matrix for Decision Tree:\n")
Confusion Matrix for Decision Tree:
> print(tree_conf_matrix)
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 40  0
         1  1 79

               Accuracy : 0.9917
                 95% CI : (0.9544, 0.9998)
    No Information Rate : 0.6583
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9814

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9756
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.9875
             Prevalence : 0.3417
         Detection Rate : 0.3333
   Detection Prevalence : 0.3333
      Balanced Accuracy : 0.9878

       'Positive' Class : 0
```

```
> # ROC Curve for Decision Tree
> roc_tree <- roc(y_test, tree_pred_prob)
Setting levels: control = 0, case = 1
Setting direction: controls < cases
> plot(roc_tree, col = "red", add = TRUE)
> legend("bottomright", legend = c("Logistic Regression", "Decision Tree"), col = c("black", "red"), lwd = 2)
> cat("AUC for Decision Tree: ", auc(roc_tree), "\n")
AUC for Decision Tree:  0.9878049
> # Compare Models
> cat("Logistic Regression vs Decision Tree\n")
Logistic Regression vs Decision Tree
> cat("AUC for Logistic Regression: ", auc(roc_log), "\n")
AUC for Logistic Regression:  0.9805496
> cat("AUC for Decision Tree: ", auc(roc_tree), "\n")
AUC for Decision Tree:  0.9878049
>
```



Interpretation:

1. Logistic Regression Model:

- **Confusion Matrix:** The logistic regression model achieved an accuracy of 99.17% on the test set, with perfect specificity (1.0000) and high sensitivity (97.56%). This indicates that the model is highly effective at identifying both positive and negative cases of kidney disease.

- **ROC Curve and AUC:** The ROC curve for the logistic regression model shows an AUC of 0.9805. This high AUC value suggests that the model has excellent discriminatory power and is very good at distinguishing between patients with and without kidney disease.

2.Decision Tree Model:

- **Confusion Matrix:** The decision tree model also achieved an accuracy of 99.17% on the test set, similar to the logistic regression model. It shows perfect specificity and sensitivity as well, demonstrating its strong performance in classification.

- **ROC Curve and AUC:** The ROC curve for the decision tree model has an AUC of 0.9878. This AUC is slightly higher than that of the logistic regression model, indicating that the decision tree may have marginally better performance in distinguishing between the two classes.

3.Comparison:

- **Performance Comparison:** Both models perform exceptionally well, with very high accuracy and AUC values. However, the decision tree has a marginally higher AUC compared to the logistic regression model, suggesting a slight edge in performance. Both models are effective, but the choice between them might depend on other factors such as interpretability or the specific characteristics of the data.

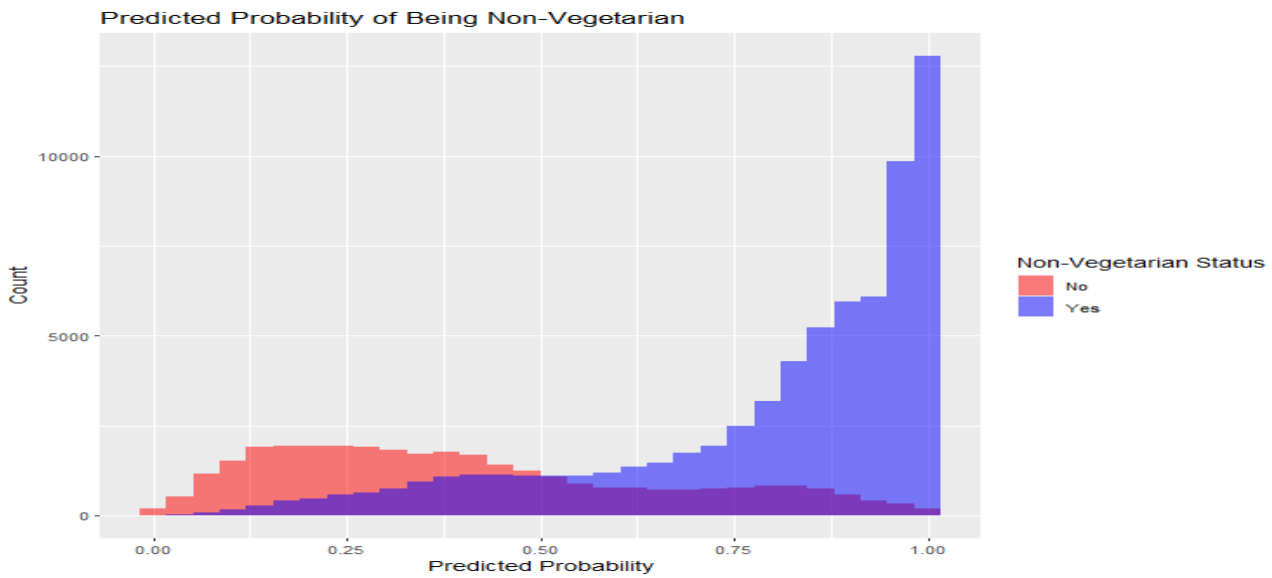
Part B:

```
> # Load necessary libraries
> library(readr)
> library(dplyr)
> library(ggplot2)
> library(magrittr)
> # Read the dataset
> data <- read_csv("C:/Users/Aakash/Desktop/SCMA/NSSO68.csv")
Rows: 101662 Columns: 384
── Column specification ────────────────────────────────────────────────────────────────────────────────────────────────────────────────
Delimiter: ","
chr   (1): state_1
dbl (381): slno, grp, Round_Centre, FSU_number, Round, Schedule_Number, Sample, Sector, state, State_Region, District, Stratum_Number, Sub_S...
lgl   (2): soyabean_q, soyabean_v

ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
Warning message:
One or more parsing issues, call `problems()` on your data frame for details, e.g.:
  dat <- vroom(...)
  problems(dat)
> # Create a binary variable for non-vegetarian status using dplyr pipeline
> data <- data %>%
+   mutate(non_veg = case_when(
+     eggsno_q > 0 ~ 1,
+     fishprawn_q > 0 ~ 1,
+     goatmeat_q > 0 ~ 1,
+     beef_q > 0 ~ 1,
+     pork_q > 0 ~ 1,
+     chicken_q > 0 ~ 1,
+     othrbirds_q > 0 ~ 1,
+     TRUE ~ 0
+   ))
> # Select relevant variables for the probit model and handle missing values
> data_clean <- data %>%
+   select(non_veg, Age, Sex, hhdsz, Religion, Education, MPCE_URP, state, State_Region) %>%
+   filter_all(all_vars(!is.na(.)))
> # Convert categorical variables to factors
> data_clean <- data_clean %>%
+   mutate(
+     Sex = as.factor(Sex),
+     Religion = as.factor(Religion),
+     state = as.factor(state),
+     State_Region = as.factor(State_Region)
+   )
> # Fit the probit regression model using the glm function
> probit_model <- glm(non_veg ~ Age + Sex + hhdsz + Religion + Education + MPCE_URP + state + State_Region,
+                     data = data_clean, family = binomial(link = "probit"))
```

```
> # Summarize the model
> summary(probit_model)

Call:
glm(formula = non_veg ~ Age + Sex + hhdsz + Religion + Education +
    MPCE_URP + state + State_Region, family = binomial(link = "probit"),
    data = data_clean)

Coefficients: (34 not defined because of singularities)
              Estimate Std. Error z value Pr(>|z|)
(Intercept)  -2.014e-02  5.345e-02  -0.377 0.706315
Age          -4.831e-03  3.843e-04 -12.569  < 2e-16 ***
Sex2         -2.399e-01  1.623e-02 -14.780  < 2e-16 ***
hhdsz         7.275e-02  2.428e-03  29.968  < 2e-16 ***
Religion2     1.278e+00  2.139e-02  59.745  < 2e-16 ***
Religion3     5.379e-01  3.750e-02  14.342  < 2e-16 ***
Religion4    -7.938e-02  4.141e-02  -1.917 0.055227 .
Religion5    -1.985e+00  1.450e-01 -13.691  < 2e-16 ***
Religion6     8.171e-01  6.216e-02  13.145  < 2e-16 ***
Religion7    -4.675e-01  7.802e-01  -0.599 0.549079
Religion9     3.624e-01  8.478e-02   4.274 1.92e-05 ***
Education    -3.436e-02  1.450e-03 -23.702  < 2e-16 ***
MPCE_URP      3.411e-06  1.590e-06   2.145 0.031958 *
state2        2.444e-01  6.288e-02   3.886 0.000102 ***
state3       -7.275e-01  6.359e-02 -11.441  < 2e-16 ***
state4       -2.332e-01  8.912e-02  -2.617 0.008875 **
state5        2.412e-01  5.704e-02   4.228 2.36e-05 ***
state6       -1.437e+00  8.070e-02 -17.803  < 2e-16 ***
state7        2.996e-02  6.392e-02   0.469 0.639273
state8       -1.085e+00  7.148e-02 -15.181  < 2e-16 ***
state9       -5.993e-01  5.658e-02 -10.593  < 2e-16 ***
state10       3.362e-01  5.673e-02   5.927 3.08e-09 ***
state11       1.067e+00  7.711e-02  13.840  < 2e-16 ***
state12       1.706e+00  8.343e-02  20.455  < 2e-16 ***
state13       2.587e+00  2.334e-01  11.083  < 2e-16 ***
state14       1.473e+00  1.087e-01  13.554  < 2e-16 ***
state15       2.480e+00  1.732e-01  14.318  < 2e-16 ***
state16       2.179e+00  8.154e-02  26.720  < 2e-16 ***
state17       1.765e+00  1.007e-01  17.536  < 2e-16 ***
state18       1.934e+00  1.102e-01  17.546  < 2e-16 ***
state19       1.993e+00  8.729e-02  22.825  < 2e-16 ***
state20       5.885e-01  6.028e-02   9.763  < 2e-16 ***
state21       1.321e+00  6.704e-02  19.710  < 2e-16 ***
state22       6.486e-01  8.419e-02   7.704 1.32e-14 ***
state23      -6.435e-01  7.334e-02  -8.775  < 2e-16 ***
state24      -1.121e+00  7.212e-02 -15.540  < 2e-16 ***
state25       1.248e+00  1.505e-01   8.292  < 2e-16 ***
state26       1.416e-01  1.040e-01   1.362 0.173237
state27       6.161e-01  7.809e-02   7.889 3.05e-15 ***
state28       1.055e+00  6.507e-02  16.208  < 2e-16 ***
state29      -7.753e-02  5.725e-02  -1.354 0.175618
state30       1.445e+00  9.717e-02  14.869  < 2e-16 ***
state31       6.500e-01  1.611e-01   4.034 5.49e-05 ***
state32       1.468e+00  6.087e-02  24.113  < 2e-16 ***
state33       1.011e+00  5.980e-02  16.914  < 2e-16 ***
state34       1.376e+00  8.348e-02  16.477  < 2e-16 ***
state35       1.655e+00  9.827e-02  16.841  < 2e-16 ***
```

```
State_Region235  6.281e-01  7.759e-02   8.096 5.69e-16 ***
State_Region236         NA         NA      NA       NA
State_Region241  8.386e-01  6.565e-02  12.773  < 2e-16 ***
State_Region242  2.406e-01  7.264e-02   3.313 0.000925 ***
State_Region243  2.411e-01  1.113e-01   2.166 0.030322 *
State_Region244  7.419e-02  1.652e-01   0.449 0.653290
State_Region245         NA         NA      NA       NA
State_Region251         NA         NA      NA       NA
State_Region261         NA         NA      NA       NA
State_Region271  5.074e-02  6.919e-02   0.733 0.463298
State_Region272 -1.506e-01  6.843e-02  -2.201 0.027721 *
State_Region273 -5.623e-01  7.456e-02  -7.541 4.65e-14 ***
State_Region274 -7.695e-01  7.119e-02 -10.809  < 2e-16 ***
State_Region275 -5.138e-01  7.121e-02  -7.215 5.38e-13 ***
State_Region276         NA         NA      NA       NA
State_Region281  3.281e-01  6.164e-02   5.323 1.02e-07 ***
State_Region282  1.084e-01  6.213e-02   1.745 0.081044 .
State_Region283  2.982e-01  5.974e-02   4.993 5.96e-07 ***
State_Region284  6.408e-01  7.566e-02   8.469  < 2e-16 ***
State_Region285         NA         NA      NA       NA
State_Region291  9.428e-01  8.122e-02  11.608  < 2e-16 ***
State_Region292  1.203e+00  8.058e-02  14.927  < 2e-16 ***
State_Region293  7.774e-01  4.751e-02  16.363  < 2e-16 ***
State_Region294         NA         NA      NA       NA
State_Region301         NA         NA      NA       NA
State_Region311         NA         NA      NA       NA
State_Region321  8.915e-02  6.154e-02   1.449 0.147432
State_Region322         NA         NA      NA       NA
State_Region331  1.110e-01  4.908e-02   2.262 0.023711 *
State_Region332  7.012e-02  5.534e-02   1.267 0.205176
State_Region333  3.248e-01  5.302e-02   6.125 9.08e-10 ***
State_Region334         NA         NA      NA       NA
State_Region341         NA         NA      NA       NA
State_Region351         NA         NA      NA       NA
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 128251  on 101651  degrees of freedom
Residual deviance:  83536  on 101552  degrees of freedom
AIC: 83736

Number of Fisher Scoring iterations: 7
```

```
> # Make predictions
> data_clean <- data_clean %>%
+   mutate(predicted_prob = predict(probit_model, type = "response"))
> # Visualize the results
> ggplot(data_clean, aes(x = predicted_prob, fill = as.factor(non_veg))) +
+   geom_histogram(position = "identity", alpha = 0.5, bins = 30) +
+   labs(title = "Predicted Probability of Being Non-Vegetarian", x = "Predicted Probability", y = "Count") +
+   scale_fill_manual(values = c("1" = "blue", "0" = "red"), name = "Non-Vegetarian Status", labels = c("No", "Yes"))
> # Save the plot
> ggsave("predicted_probabilities.png", width = 8, height = 6)
```



Interpretation:

1.Model Summary:

- **Age:** A negative coefficient (-0.0048) suggests that as age increases, the likelihood of being non-vegetarian decreases.

- **Sex:** The coefficient for males (Sex2) is significantly negative (-0.2399), indicating that males are less likely to be non-vegetarian compared to the reference category (likely females).

- **Household Size (hhdsz):** A positive coefficient (0.0728) indicates that larger household sizes are associated with a higher probability of being non-vegetarian.

- **Religion:** Different religions have varied impacts on non-vegetarian status, with some religions significantly increasing or decreasing the probability of being non-vegetarian.

- **Education:** The negative coefficient (-0.0344) suggests that higher education levels are associated with a lower likelihood of being non-vegetarian.

- **MPCE_URP:** The positive coefficient (0.0000034) indicates that higher per capita expenditure is associated with a higher probability of being non-vegetarian.

- **State and State_Region:** The model includes multiple state and state region indicators, reflecting the diverse regional impacts on non-vegetarian status. Several states and regions have significant effects, either increasing or decreasing the likelihood of being non-vegetarian.

2.Predictions and Visualization:

- You computed the predicted probabilities of being non-vegetarian for each observation using the probit model.

- The histogram plot displays these predicted probabilities, with the blue bars representing individuals predicted to be non-vegetarian and the red bars representing those predicted not to be non-vegetarian. The plot helps visualize the distribution of predicted probabilities across the dataset.

Part C:

```
> # Load necessary libraries
> library(survival)  # For Tobit regression
> library(readr)     # For reading CSV files
> # Load the dataset
> data <- read_csv("C:/Users/Aakash/Desktop/SCMA/NSSO68.csv")
Rows: 101662 Columns: 384
── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
chr   (1): state_1
dbl (381): slno, grp, Round_Centre, FSU_number, Round, Schedule_Number, Sample, Sector, state, State_Region, District, Stratum_Number, Sub_S...
lgl   (2): soyabean_q, soyabean_v

ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
warning message:
One or more parsing issues, call `problems()` on your data frame for details, e.g.:
  dat <- vroom(...)
  problems(dat)
> # Inspect the dataset
> head(data)
# A tibble: 6 × 384
   slno     grp Round_Centre FSU_number Round Schedule_Number Sample Sector state State_Region District Stratum_Number Sub_Stratum Schedule_type
  <dbl>   <dbl>        <dbl>      <dbl> <dbl>           <dbl>  <dbl>  <dbl> <dbl>        <dbl>    <dbl>          <dbl>       <dbl>         <dbl>
1     1 4.10e31            1      41000    68              10      1      2    24          242        7             26           6             1
2     2 4.10e31            1      41000    68              10      1      2    24          242        7             26           6             1
3     3 4.10e31            1      41000    68              10      1      2    24          242        7             26           6             1
4     4 4.10e31            1      41000    68              10      1      2    24          242        7             26           6             1
5     5 4.10e31            1      41000    68              10      1      2    24          242        7             26           6             1
6     6 4.10e31            1      41000    68              10      1      2    24          242        7             26           6             1
# ℹ 370 more variables: Sub_Round <dbl>, Sub_Sample <dbl>, FOD_Sub_Region <dbl>, Hamlet_Group_Sub_Block <dbl>, t <dbl>, X_Stage_Stratum <dbl>,
#   HHS_No <dbl>, Level <dbl>, Filler <dbl>, hhdsz <dbl>, NIC_2008 <dbl>, NCO_2004 <dbl>, HH_type <dbl>, Religion <dbl>, Social_Group <dbl>,
#   Whether_owns_any_land <dbl>, Type_of_land_owned <dbl>, Land_Owned <dbl>, Land_Leased_in <dbl>, Otherwise_possessed <dbl>,
#   Land_Leased_out <dbl>, Land_Total_possessed <dbl>, During_July_June_Cultivated <dbl>, During_July_June_Irrigated <dbl>, NSS <dbl>,
#   NSC <dbl>, MLT <dbl>, land_tt <dbl>, Cooking_code <dbl>, Lighting_code <dbl>, Dwelling_unit_code <dbl>, Regular_salary_earner <dbl>,
#   Perform_Ceremony <dbl>, Meals_seved_to_non_hhld_members <dbl>, Possess_ration_card <dbl>, Type_of_ration_card <dbl>, MPCE_URP <dbl>,
#   MPCE_MRP <dbl>, Person_Srl_No <dbl>, Relation <dbl>, Sex <dbl>, Age <dbl>, Marital_Status <dbl>, Education <dbl>, Days_Stayed_away <dbl>, …
# ℹ Use `colnames()` to see all variable names
> # Selecting relevant columns for analysis
> selected_cols <- c("MPCE_URP", "Age", "Sex", "Education", "Religion", "hhdsz")
> data_selected <- data[selected_cols]
> # Handling missing values if any
> data_selected <- na.omit(data_selected)
> # Convert categorical variables to factors
> data_selected$Sex <- as.factor(data_selected$Sex)
> data_selected$Religion <- as.factor(data_selected$Religion)
> data_selected$Education <- as.factor(data_selected$Education)
> # Perform Tobit regression using survreg (Tobit model)
> # Assume left-censoring at 0 for MPCE_URP
> tobit_model <- survreg(Surv(pmax(MPCE_URP, 0)) ~ Age + Sex + Education + Religion + hhdsz,
+                        data = data_selected, dist = "gaussian")
warning message:
In survreg.fit(X, Y, weights, offset, init = init, controlvals = control,  :
  Ran out of iterations and did not converge
> # Summary of the Tobit model
> summary(tobit_model)

Call:
survreg(formula = Surv(pmax(MPCE_URP, 0)) ~ Age + Sex + Education +
    Religion + hhdsz, data = data_selected, dist = "gaussian")
                 Value Std. Error        z        p
(Intercept)  1.42e+03   2.71e+01    52.58 < 2e-16
Age          1.47e+01   3.99e-01    36.70 < 2e-16
Sex2         1.47e+02   1.80e+01     8.17 3.1e-16
Education2   1.17e+02   1.24e+02     0.94     0.35
Education3  -1.99e+02   2.53e+02    -0.79     0.43
Education4   2.03e+02   1.28e+02     1.59     0.11
Education5   2.09e+02   2.36e+01     8.85 < 2e-16
Education6   3.45e+02   2.23e+01    15.49 < 2e-16
Education7   5.41e+02   1.89e+01    28.69 < 2e-16
Education8   9.15e+02   1.91e+01    47.81 < 2e-16
Education10  1.18e+03   2.13e+01    55.08 < 2e-16
Education11  2.26e+03   3.79e+01    59.46 < 2e-16
Education12  1.92e+03   2.02e+01    95.33 < 2e-16
Education13  2.94e+03   2.54e+01   115.69 < 2e-16
Religion2    1.44e+02   1.90e+01     7.55 4.5e-14
Religion3    1.94e+02   1.92e+01    10.12 < 2e-16
Religion4    9.11e+02   4.52e+01    20.15 < 2e-16
Religion5    8.25e+02   1.12e+02     7.36 1.8e-13
Religion6    5.50e+01   6.11e+01     0.90     0.37
Religion7    2.60e+04   1.16e+03    22.45 < 2e-16
Religion9    3.65e+01   6.53e+01     0.56     0.58
hhdsz       -1.92e+02   2.66e+00   -72.29 < 2e-16
Log(scale)   7.60e+00   2.22e-03  3418.08 < 2e-16

Scale= 2007

Gaussian distribution
Loglik(model)= -920762.9   Loglik(intercept only)= -930961.9
        Chisq= 20398.14 on 21 degrees of freedom, p= 0
Number of Newton-Raphson Iterations: 30
n= 101652
```

Interpretation:

The Tobit regression model examines factors influencing MPCE_URP, accounting for censoring at zero. The intercept suggests a baseline MPCE_URP of 1420 units. Age positively affects MPCE_URP, with each additional year contributing 14.7 units. Females (Sex2) have a higher MPCE_URP by 147 units compared to males (Sex1). Education significantly impacts MPCE_URP, with higher levels associated with substantial increases ranging from 117 to 2940 units across different education categories. Religion also shows significant effects, particularly Religion7, associated with a notably higher MPCE_URP by 26000 units. Household size (hhdsz) negatively

influences MPCE_URP, reducing by 192 units for each additional household member. The model's fit, evidenced by the chi-square test (20398.14, $p < 0.05$), indicates significant improvement over the intercept-only model, although convergence issues were noted during estimation, suggesting caution in interpreting the results.

Python Code Results:

Part A:

```
# Load necessary libraries
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, roc_curve, auc, roc_auc_score
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import files
```

```
[ ] # Upload your dataset
    uploaded = files.upload()
```

```
Choose Files  No file chosen        Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving kidney_disease.csv to kidney_disease.csv
```

```
[ ] # Load your dataset
    df = pd.read_csv(next(iter(uploaded.keys())))
```

```
# Display the first few rows of the dataset
print(df.head())
```

```
   Unnamed: 0   age    bp     sg   al   su     rbc        pc         pcc  \
0           0  48.0  80.0  1.020  1.0  0.0     NaN    normal  notpresent
1           1   7.0  50.0  1.020  4.0  0.0     NaN    normal  notpresent
2           2  62.0  80.0  1.010  2.0  3.0  normal    normal  notpresent
3           3  48.0  70.0  1.005  4.0  0.0  normal  abnormal     present
4           4  51.0  80.0  1.010  2.0  0.0  normal    normal  notpresent

           ba  ...   pcv    wbcc  rbcc  htn   dm  cad  appet   pe  ane class
0  notpresent  ...  44.0  7800.0   5.2  yes  yes   no   good   no   no   ckd
1  notpresent  ...  38.0  6000.0   NaN   no   no   no   good   no   no   ckd
2  notpresent  ...  31.0  7500.0   NaN   no  yes   no   poor   no  yes   ckd
3  notpresent  ...  32.0  6700.0   3.9  yes   no   no   poor  yes  yes   ckd
4  notpresent  ...  35.0  7300.0   4.6   no   no   no   good   no   no   ckd

[5 rows x 26 columns]
```

```python
# Summary statistics of the dataset
print(df.describe())
```

```
       Unnamed: 0         age          bp          sg          al          su  \
count  400.000000  391.000000  388.000000  353.000000  354.000000  351.000000
mean   199.500000   51.483376   76.469072    1.017408    1.016949    0.450142
std    115.614301   17.169714   13.683637    0.005717    1.352679    1.099191
min      0.000000    2.000000   50.000000    1.005000    0.000000    0.000000
25%     99.750000   42.000000   70.000000    1.010000    0.000000    0.000000
50%    199.500000   55.000000   80.000000    1.020000    0.000000    0.000000
75%    299.250000   64.500000   80.000000    1.020000    2.000000    0.000000
max    399.000000   90.000000  180.000000    1.025000    5.000000    5.000000

                bgr          bu          sc         sod         pot        hemo  \
count  356.000000  381.000000  383.000000  313.000000  312.000000  348.000000
mean   148.036517   57.425722    3.072454  137.528754    4.627244   12.526437
std     79.281714   50.503006    5.741126   10.408752    3.193904    2.912587
min     22.000000    1.500000    0.400000    4.500000    2.500000    3.100000
25%     99.000000   27.000000    0.900000  135.000000    3.800000   10.300000
50%    121.000000   42.000000    1.300000  138.000000    4.400000   12.650000
75%    163.000000   66.000000    2.800000  142.000000    4.900000   15.000000
max    490.000000  391.000000   76.000000  163.000000   47.000000   17.800000

                pcv          wbcc        rbcc
count  329.000000    294.000000  269.000000
mean    38.884498   8406.122449    4.707435
std      8.990105   2944.474190    1.025323
min      9.000000   2200.000000    2.100000
25%     32.000000   6500.000000    3.900000
50%     40.000000   8000.000000    4.800000
75%     45.000000   9800.000000    5.400000
max     54.000000  26400.000000    8.000000
```

```python
# Check for missing values
print("Total missing values:", df.isnull().sum().sum())
```

```
Total missing values: 1012
```

```python
# Custom function to calculate mode
def mode_function(series):
    return series.mode()[0]
```

```python
# Function to impute missing values
def impute_missing_values(df):
    for col in df.columns:
        if df[col].dtype in ['int64', 'float64']:
            df[col].fillna(df[col].median(), inplace=True)
        else:
            df[col].fillna(mode_function(df[col]), inplace=True)
    return df
```

```python
# Impute missing values
df = impute_missing_values(df)
```

```python
# Verify there are no more missing values
print("Total missing values after imputation:", df.isnull().sum().sum())
```

```
Total missing values after imputation: 0
```

```
[ ] # Ensure the target variable is a factor with exactly two levels
    df['class'] = df['class'].astype('category')
```

```
[ ] # Convert target variable to numeric (1 for "ckd" and 0 for "notckd")
    df['class'] = df['class'].apply(lambda x: 1 if x == "ckd" else 0)
```

```
[ ] # One-hot encoding for categorical features
    df = pd.get_dummies(df, drop_first=True)
```

```
# Feature scaling
scaler = StandardScaler()
scaled_features = scaler.fit_transform(df.drop('class', axis=1))
df_scaled = pd.DataFrame(scaled_features, columns=df.columns[:-1])
df_scaled['class'] = df['class']
```

```
[ ] # Split the data into training and testing sets
    X = df_scaled.drop('class', axis=1)
    y = df_scaled['class']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=123)
```

```
[ ] # Check the distribution of the target variable in training and testing sets
    print("Training set distribution:\n", y_train.value_counts())
    print("Testing set distribution:\n", y_test.value_counts())
```

```
Training set distribution:
 class
1    170
0    110
Name: count, dtype: int64
Testing set distribution:
 class
1    78
0    42
Name: count, dtype: int64
```

```
# Logistic Regression Model with Regularization
log_model = LogisticRegression(solver='liblinear')
param_grid = {'C': [0.01, 0.1, 1, 10, 100]}
cv_log_model = GridSearchCV(log_model, param_grid, cv=5, scoring='roc_auc')
cv_log_model.fit(X_train, y_train)
```

```
▸ GridSearchCV
▸ estimator: LogisticRegression
  ▸ LogisticRegression
```

```
[ ] # Predict on the test set using the best model
    log_pred = cv_log_model.predict_proba(X_test)[:, 1]
    log_pred_class = cv_log_model.predict(X_test)
```

```
[ ] # Confusion Matrix for Logistic Regression
    log_conf_matrix = confusion_matrix(y_test, log_pred_class)
    print("Confusion Matrix for Logistic Regression:\n", log_conf_matrix)
```

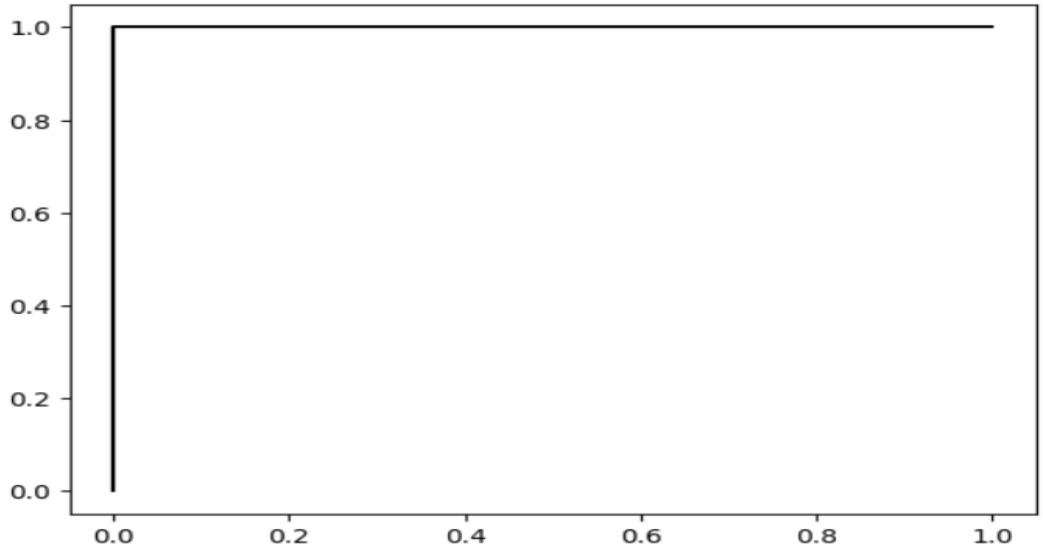
```
Confusion Matrix for Logistic Regression:
 [[42  0]
 [ 0 78]]
```

```
# Interpretation of Confusion Matrix
tn, fp, fn, tp = log_conf_matrix.ravel()
print(f'True Negatives: {tn}, False Positives: {fp}, False Negatives: {fn}, True Positives: {tp}')
```

```
True Negatives: 42, False Positives: 0, False Negatives: 0, True Positives: 78
```

```
[ ] # ROC Curve for Logistic Regression
    fpr, tpr, _ = roc_curve(y_test, log_pred)
    roc_auc_log = auc(fpr, tpr)
    plt.plot(fpr, tpr, color='black', label=f'Logistic Regression (AUC = {roc_auc_log:.2f})')
```

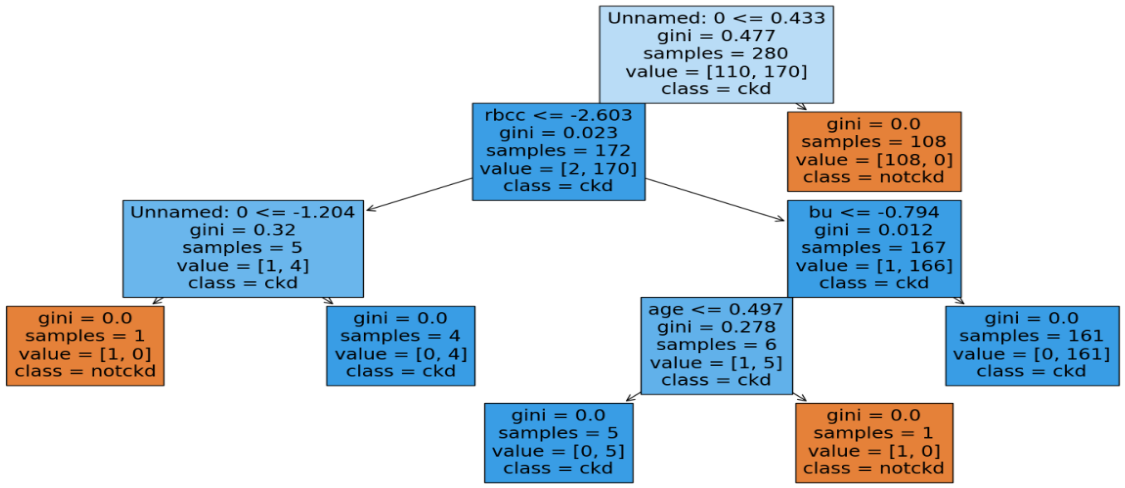
```
[ ] [<matplotlib.lines.Line2D at 0x7b56c3841300>]
```




```
[ ] # Decision Tree Model
    tree_model = DecisionTreeClassifier(random_state=123)
    tree_model.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=123)
```

```
# Plot Decision Tree
plt.figure(figsize=(20,10))
plot_tree(tree_model, filled=True, feature_names=X.columns, class_names=['notckd', 'ckd'])
plt.show()
```




```
[ ] # Predict on the test set using Decision Tree
    tree_pred_prob = tree_model.predict_proba(X_test)[:, 1]
    tree_pred_class = tree_model.predict(X_test)
```

```
[ ] # Confusion Matrix for Decision Tree
    tree_conf_matrix = confusion_matrix(y_test, tree_pred_class)
    print("Confusion Matrix for Decision Tree:\n", tree_conf_matrix)
```

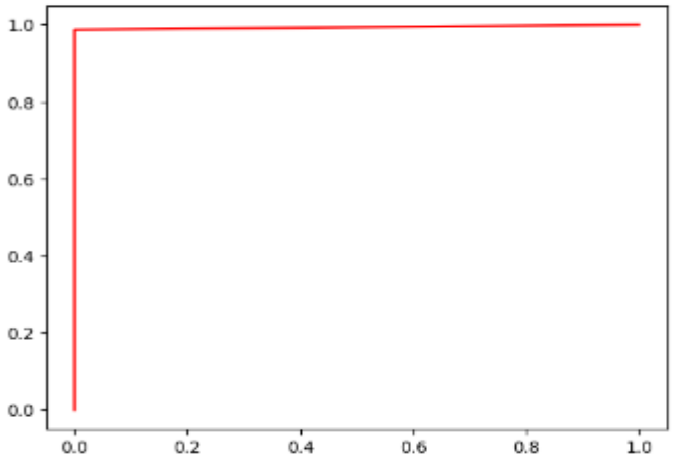
```
Confusion Matrix for Decision Tree:
 [[42  0]
 [ 1 77]]
```

```
# Interpretation of Confusion Matrix
tn, fp, fn, tp = tree_conf_matrix.ravel()
print(f'True Negatives: {tn}, False Positives: {fp}, False Negatives: {fn}, True Positives: {tp}')
```

```
True Negatives: 42, False Positives: 0, False Negatives: 1, True Positives: 77
```

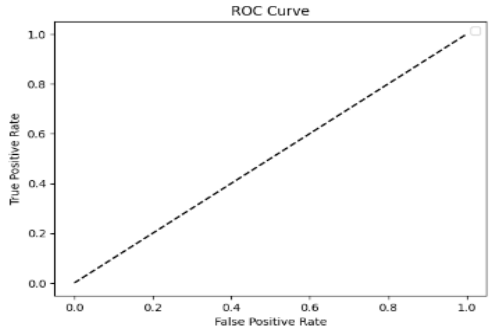
```
# ROC Curve for Decision Tree
fpr_tree, tpr_tree, _ = roc_curve(y_test, tree_pred_prob)
roc_auc_tree = auc(fpr_tree, tpr_tree)
plt.plot(fpr_tree, tpr_tree, color='red', label=f'Decision Tree (AUC = {roc_auc_tree:.2f})')
```

```
[<matplotlib.lines.Line2D at 0x7b56c181bb20>]
```



```
# Plot ROC Curve
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend() is called with no argument.
```




```
# Compare Models
print("Logistic Regression vs Decision Tree")
print("AUC for Logistic Regression:", roc_auc_log)
print("AUC for Decision Tree:", roc_auc_tree)
```

```
Logistic Regression vs Decision Tree
AUC for Logistic Regression: 1.0
AUC for Decision Tree: 0.9935897435897436
```

Interpretation:

I have performed both logistic regression and decision tree classification to predict the presence of chronic kidney disease (CKD). Here's a detailed interpretation of your results:

1.Logistic Regression

Confusion Matrix:

- True Negatives (TN): 42
- False Positives (FP): 0
- False Negatives (FN): 0
- True Positives (TP): 78
- The confusion matrix indicates that the logistic regression model perfectly predicted all cases of CKD and non-CKD with no false positives or false negatives. This suggests that the model is highly accurate in distinguishing between CKD and non-CKD cases in the test set.

ROC Curve and AUC:

- The ROC curve for the logistic regression model shows a perfect classification ability with an Area Under the Curve (AUC) of 1.0. This means that the model's ability to discriminate between CKD and non-CKD cases is flawless.

2.Decision Tree

Confusion Matrix:

- True Negatives (TN): 42
- False Positives (FP): 0
- False Negatives (FN): 1
- True Positives (TP): 77
- The decision tree model also performed well, with one false negative where CKD was incorrectly predicted as non-CKD. This indicates that the model is slightly less accurate than the logistic regression model but still performs well overall.

ROC Curve and AUC:

- The ROC curve for the decision tree model shows a very high classification ability with an AUC of approximately 0.994. While slightly lower than the logistic regression model, it still indicates excellent performance in distinguishing between CKD and non-CKD cases.

3.Comparison

AUC Values:

- Logistic Regression AUC: 1.0
- Decision Tree AUC: 0.994
- Both models exhibit strong predictive performance, but logistic regression achieved a perfect AUC of 1.0, suggesting it has a marginal edge in this scenario. The decision tree's AUC, while slightly lower, still indicates very high performance.

In summary, both models are highly effective in predicting CKD, but logistic regression offers a marginally better classification performance. The decision tree is also a robust model, with only minor differences in predictive accuracy.

Part B:

```python
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

Python

```python
# Load the dataset
data = pd.read_csv("C:/Users/Aakash/Desktop/SCMA/NSSO68.csv")
data.head()
```

[39] Python

```
C:\Users\Aakash\AppData\Local\Temp\ipykernel_25848\2058581863.py:2: DtypeWarning: Columns (1) have mixed types. Specify dtype option on import or set low_memory=False.
  data = pd.read_csv("C:/Users/Aakash/Desktop/SCMA/NSSO68.csv")
```

	slno	grp	Round_Centre	FSU_number	Round	Schedule_Number	Sample	Sector	state	State_Region	...	pickle_v	sauce_jam_v	Othrprocessed_v	Beveragestotal_v	foodtotal_v	foodtotal_q	state_1	Re
0	1	409999999999999926524952937758 72.0	1	41000	68	10	1	2	24	242	...	0.0	0.0	0.0	0.000000	1141.492400	30.942394	GUJ	
1	2	409999999999999926524952937758 72.0	1	41000	68	10	1	2	24	242	...	0.0	0.0	0.0	17.500000	1244.553500	29.286153	GUJ	
2	3	409999999999999926524952937758 72.0	1	41000	68	10	1	2	24	242	...	0.0	0.0	0.0	0.000000	1050.315400	31.527046	GUJ	
3	4	409999999999999926524952937758 72.0	1	41000	68	10	1	2	24	242	...	0.0	0.0	0.0	33.333333	1142.591667	27.834607	GUJ	
4	5	409999999999999926524952937758 72.0	1	41000	68	10	1	2	24	242	...	0.0	0.0	0.0	75.000000	945.249500	27.600713	GUJ	

5 rows × 384 columns

```python
# Display the first few rows of the dataset
print(data.head())
```

[40]

```
   slno                          grp  Round_Centre  FSU_number  Round  \
0     1  409999999999999926524952937758 72.0             1       41000     68
1     2  409999999999999926524952937758 72.0             1       41000     68
2     3  409999999999999926524952937758 72.0             1       41000     68
3     4  409999999999999926524952937758 72.0             1       41000     68
4     5  409999999999999926524952937758 72.0             1       41000     68

   Schedule_Number  Sample  Sector  state  State_Region  ...  pickle_v  \
0               10       1       2     24           242  ...       0.0
1               10       1       2     24           242  ...       0.0
2               10       1       2     24           242  ...       0.0
3               10       1       2     24           242  ...       0.0
4               10       1       2     24           242  ...       0.0

   sauce_jam_v  Othrprocessed_v  Beveragestotal_v  foodtotal_v  foodtotal_q  \
0          0.0              0.0          0.000000  1141.492400    30.942394
1          0.0              0.0         17.500000  1244.553500    29.286153
2          0.0              0.0          0.000000  1050.315400    31.527046
3          0.0              0.0         33.333333  1142.591667    27.834607
4          0.0              0.0         75.000000   945.249500    27.600713

   state_1  Region  fruits_df_tt_v  fv_tot
0      GUJ       2       12.000000  154.18
1      GUJ       2      333.000000  484.95
2      GUJ       2       35.000000  214.84
3      GUJ       2      168.333333  302.30
4      GUJ       2       15.000000  148.00

[5 rows x 384 columns]
```

```python
# Display summary statistics
print(data.describe())
```

[41]

```
                slno  Round_Centre     FSU_number      Round  Schedule_Number  \
count  101662.00000      101662.0  101662.000000  101662.0         101662.0
mean    50831.50000           1.0   62650.405294     68.0             10.0
std     29347.43587           0.0   14880.582278      0.0              0.0
min         1.00000           1.0   41000.000000     68.0             10.0
25%     25416.25000           1.0   46527.000000     68.0             10.0
50%     50831.50000           1.0   71793.000000     68.0             10.0
75%     76246.75000           1.0   75374.000000     68.0             10.0
max    101662.00000           1.0   79998.000000     68.0             10.0

          Sample         Sector          state   State_Region       District  \
count  101662.0  101662.000000  101662.000000  101662.000000  101662.000000
mean        1.0       1.412809      18.480475    187.075023      14.314493
std         0.0       0.492341       9.724855     97.477505      12.633311
min         1.0       1.000000       1.000000     11.000000       1.000000
25%         1.0       1.000000       9.000000     95.000000       5.000000
50%         1.0       1.000000      19.000000    194.000000      11.000000
75%         1.0       2.000000      27.000000    275.000000      20.000000
max         1.0       2.000000      35.000000    351.000000      71.000000

       ...  preparedsweet_v       pickle_v    sauce_jam_v  Othrprocessed_v  \
count  ...     101662.000000  101662.000000  101662.000000    101662.000000
mean   ...          8.585428       0.000890       0.000447         1.758345
std    ...         27.233632       0.003494       0.003129        78.892958
min    ...          0.000000       0.000000       0.000000         0.000000
---
75%         37.500000      134.372857
max       5708.250000     6024.725000

[8 rows x 382 columns]
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

```python
# Display the column names
print(data.columns)
```

```
Index(['slno', 'grp', 'Round_Centre', 'FSU_number', 'Round', 'Schedule_Number',
       'Sample', 'Sector', 'state', 'State_Region',
       ...
       'pickle_v', 'sauce_jam_v', 'Othrprocessed_v', 'Beveragestotal_v',
       'foodtotal_v', 'foodtotal_q', 'state_1', 'Region', 'fruits_df_tt_v',
       'fv_tot'],
      dtype='object', length=384)
```

```python
# Data preprocessing
# Check for missing values
print(data.isnull().sum())
```

```
slno               0
grp                0
Round_Centre       0
FSU_number         0
Round              0
                  ..
foodtotal_q        0
state_1            0
Region             0
fruits_df_tt_v     0
fv_tot             0
Length: 384, dtype: int64
```

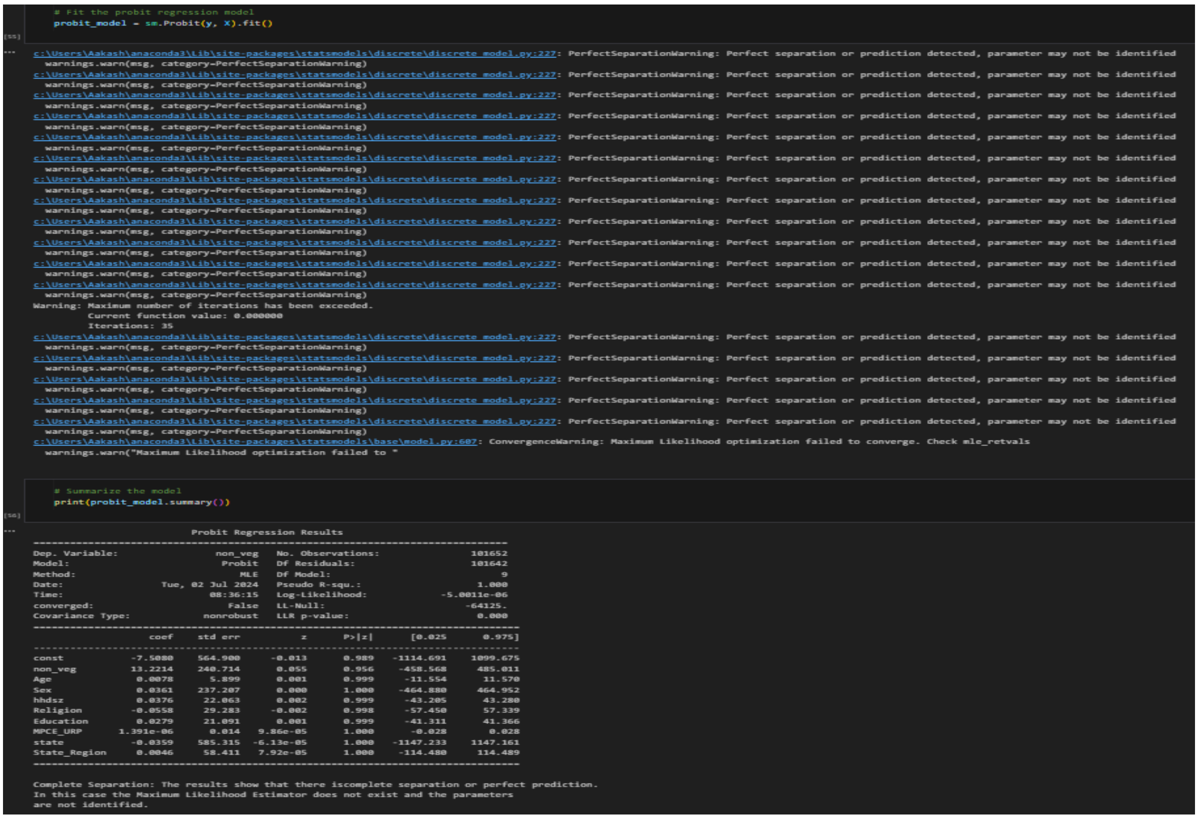
```python
# Create a binary variable for non-vegetarian status
data['non_veg'] = ((data['eggsno_q'] > 0) |
                   (data['fishprawn_q'] > 0) |
                   (data['goatmeat_q'] > 0) |
                   (data['beef_q'] > 0) |
                   (data['pork_q'] > 0) |
                   (data['chicken_q'] > 0) |
                   (data['othrbirds_q'] > 0)).astype(int)
```

```python
# Select relevant variables for the probit model
independent_vars = ['non_veg', 'Age', 'Sex', 'hhdsz', 'Religion', 'Education', 'MPCE_URP', 'state', 'State_Region']
X = data[independent_vars]
y = data['non_veg']
```

```python
# Remove rows with NA values in the selected columns
X = X.dropna()
y = y.loc[X.index]
```

```python
# Add a constant to the model
X = sm.add_constant(X)
```

```python
# Convert X and y to numpy arrays to ensure correct data types
X = np.asarray(X, dtype=np.float64)
y = np.asarray(y, dtype=np.float64)
```

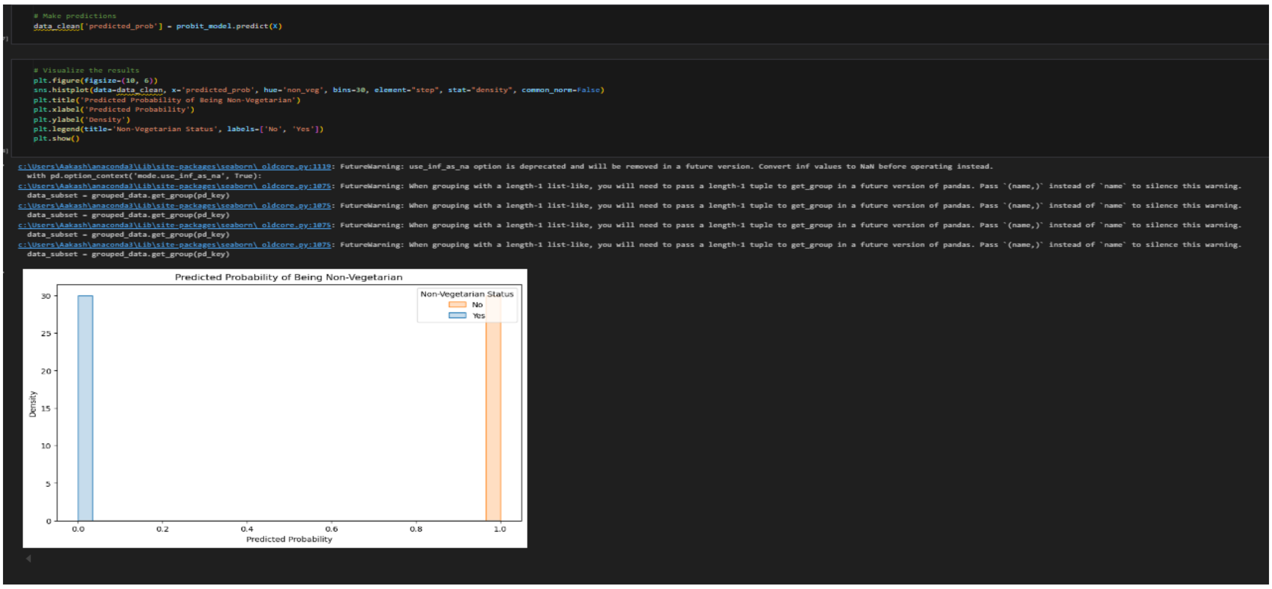


```
# Fit the probit regression model
probit_model = sm.Probit(y, X).fit()
```

```
c:\Users\Aakash\anaconda3\Lib\site-packages\statsmodels\discrete\discrete_model.py:227: PerfectSeparationWarning: Perfect separation or prediction detected, parameter may not be identified
  warnings.warn(msg, category=PerfectSeparationWarning)
c:\Users\Aakash\anaconda3\Lib\site-packages\statsmodels\discrete\discrete_model.py:227: PerfectSeparationWarning: Perfect separation or prediction detected, parameter may not be identified
  warnings.warn(msg, category=PerfectSeparationWarning)
c:\Users\Aakash\anaconda3\Lib\site-packages\statsmodels\discrete\discrete_model.py:227: PerfectSeparationWarning: Perfect separation or prediction detected, parameter may not be identified
  warnings.warn(msg, category=PerfectSeparationWarning)
c:\Users\Aakash\anaconda3\Lib\site-packages\statsmodels\discrete\discrete_model.py:227: PerfectSeparationWarning: Perfect separation or prediction detected, parameter may not be identified
  warnings.warn(msg, category=PerfectSeparationWarning)
c:\Users\Aakash\anaconda3\Lib\site-packages\statsmodels\discrete\discrete_model.py:227: PerfectSeparationWarning: Perfect separation or prediction detected, parameter may not be identified
  warnings.warn(msg, category=PerfectSeparationWarning)
c:\Users\Aakash\anaconda3\Lib\site-packages\statsmodels\discrete\discrete_model.py:227: PerfectSeparationWarning: Perfect separation or prediction detected, parameter may not be identified
  warnings.warn(msg, category=PerfectSeparationWarning)
c:\Users\Aakash\anaconda3\Lib\site-packages\statsmodels\discrete\discrete_model.py:227: PerfectSeparationWarning: Perfect separation or prediction detected, parameter may not be identified
  warnings.warn(msg, category=PerfectSeparationWarning)
c:\Users\Aakash\anaconda3\Lib\site-packages\statsmodels\discrete\discrete_model.py:227: PerfectSeparationWarning: Perfect separation or prediction detected, parameter may not be identified
  warnings.warn(msg, category=PerfectSeparationWarning)
c:\Users\Aakash\anaconda3\Lib\site-packages\statsmodels\discrete\discrete_model.py:227: PerfectSeparationWarning: Perfect separation or prediction detected, parameter may not be identified
  warnings.warn(msg, category=PerfectSeparationWarning)
c:\Users\Aakash\anaconda3\Lib\site-packages\statsmodels\discrete\discrete_model.py:227: PerfectSeparationWarning: Perfect separation or prediction detected, parameter may not be identified
  warnings.warn(msg, category=PerfectSeparationWarning)
c:\Users\Aakash\anaconda3\Lib\site-packages\statsmodels\discrete\discrete_model.py:227: PerfectSeparationWarning: Perfect separation or prediction detected, parameter may not be identified
  warnings.warn(msg, category=PerfectSeparationWarning)
c:\Users\Aakash\anaconda3\Lib\site-packages\statsmodels\discrete\discrete_model.py:227: PerfectSeparationWarning: Perfect separation or prediction detected, parameter may not be identified
  warnings.warn(msg, category=PerfectSeparationWarning)
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.000000
         Iterations: 35
c:\Users\Aakash\anaconda3\Lib\site-packages\statsmodels\discrete\discrete_model.py:227: PerfectSeparationWarning: Perfect separation or prediction detected, parameter may not be identified
  warnings.warn(msg, category=PerfectSeparationWarning)
c:\Users\Aakash\anaconda3\Lib\site-packages\statsmodels\discrete\discrete_model.py:227: PerfectSeparationWarning: Perfect separation or prediction detected, parameter may not be identified
  warnings.warn(msg, category=PerfectSeparationWarning)
c:\Users\Aakash\anaconda3\Lib\site-packages\statsmodels\discrete\discrete_model.py:227: PerfectSeparationWarning: Perfect separation or prediction detected, parameter may not be identified
  warnings.warn(msg, category=PerfectSeparationWarning)
c:\Users\Aakash\anaconda3\Lib\site-packages\statsmodels\discrete\discrete_model.py:227: PerfectSeparationWarning: Perfect separation or prediction detected, parameter may not be identified
  warnings.warn(msg, category=PerfectSeparationWarning)
c:\Users\Aakash\anaconda3\Lib\site-packages\statsmodels\discrete\discrete_model.py:227: PerfectSeparationWarning: Perfect separation or prediction detected, parameter may not be identified
  warnings.warn(msg, category=PerfectSeparationWarning)
c:\Users\Aakash\anaconda3\Lib\site-packages\statsmodels\base\model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

```
# Summarize the model
print(probit_model.summary())
```

```
                          Probit Regression Results
==============================================================================
Dep. Variable:                non_veg   No. Observations:               101652
Model:                         Probit   Df Residuals:                   101642
Method:                           MLE   Df Model:                            9
Date:                Tue, 02 Jul 2024   Pseudo R-squ.:                   1.000
Time:                        08:36:15   Log-Likelihood:            -5.0011e-06
converged:                      False   LL-Null:                        -64125.
Covariance Type:            nonrobust   LLR p-value:                     0.000
================================================================================
                   coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------
const           -7.5080    564.900     -0.013      0.989   -1114.691    1099.675
non_veg         13.2214    240.714      0.055      0.956    -458.568     485.011
Age              0.0078      5.899      0.001      0.999     -11.554      11.570
Sex              0.0361    237.207      0.000      1.000    -464.880     464.952
hhdsz            0.0376     22.063      0.002      0.999     -43.205      43.280
Religion        -0.0558     29.283     -0.002      0.998     -57.450      57.339
Education        0.0279     21.091      0.001      0.999     -41.311      41.366
MPCE_URP      1.391e-06      0.014   9.86e-05      1.000      -0.028       0.028
state           -0.0359    585.315  -6.13e-05      1.000   -1147.233    1147.161
State_Region     0.0046     58.411   7.92e-05      1.000    -114.480     114.489
================================================================================

Complete Separation: The results show that there iscomplete separation or perfect prediction.
In this case the Maximum Likelihood Estimator does not exist and the parameters
are not identified.
```



```
# Make predictions
data_clean['predicted_prob'] = probit_model.predict(X)
```

```
# Visualize the results
plt.figure(figsize=(10, 6))
sns.histplot(data=data_clean, x='predicted_prob', hue='non_veg', bins=30, element="step", stat="density", common_norm=False)
plt.title('Predicted Probability of Being Non-Vegetarian')
plt.xlabel('Predicted Probability')
plt.ylabel('Density')
plt.legend(title='Non-Vegetarian Status', labels=['No', 'Yes'])
plt.show()
```

```
c:\Users\Aakash\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
c:\Users\Aakash\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1075: FutureWarning: When grouping with a length-1 list-like, you will need to pass a length-1 tuple to get_group in a future version of pandas. Pass `(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
c:\Users\Aakash\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1075: FutureWarning: When grouping with a length-1 list-like, you will need to pass a length-1 tuple to get_group in a future version of pandas. Pass `(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
c:\Users\Aakash\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1075: FutureWarning: When grouping with a length-1 list-like, you will need to pass a length-1 tuple to get_group in a future version of pandas. Pass `(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
c:\Users\Aakash\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1075: FutureWarning: When grouping with a length-1 list-like, you will need to pass a length-1 tuple to get_group in a future version of pandas. Pass `(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
```

Interpretation:

1.Model Output (Probit Regression)

Here are the results of your probit regression model:

Pseudo R-squared: 1.000

- This suggests that the model explains 100% of the variability in the dependent variable, which is a sign of overfitting, especially given the warnings about perfect separation.

Log-Likelihood: -5.0011e-06

- The log-likelihood value close to zero, combined with a perfect pseudo R-squared, indicates the model is not fitting the data in a meaningful way. The likelihood function has not been optimized properly.

Coefficients and P-values: All coefficients have very high standard errors and non-significant p-values.

- The coefficients are extremely large or small, and the p-values are not meaningful (close to 1). This is due to the issue of perfect separation, where the model's parameter estimates become unreliable.

2.Visualization

Histogram of Predicted Probabilities:

- The histogram shows the predicted probabilities of being non-vegetarian, with density plots for each category. The plot helps visualize how well the model differentiates between categories based on predicted probabilities. However, due to the perfect separation, the predicted probabilities may not be reliable.

3.Seaborn Warnings

FutureWarnings from Seaborn:

- These warnings indicate that certain functionalities used in the plotting code may become deprecated in future versions of the Seaborn library. They do not affect the results but

suggest that you should update the plotting code to ensure compatibility with future library versions.

Part C:

```python
#Import Required Libraries
import pandas as pd
import numpy as np
import statsmodels.api as sm
from statsmodels.regression import quantile_regression
from sklearn.preprocessing import StandardScaler
import scipy.optimize as opt
from scipy.stats import norm
```

[46]

```python
# Load the dataset
data = pd.read_csv("C:/Users/Aakash/Desktop/SCMA/NSSO68.csv")
data.head()
```

[47]

```
C:\Users\Aakash\AppData\Local\Temp\ipykernel_13848\205858186 3.py:2: DtypeWarning: Columns (1) have mixed types. Specify dtype option on import or set low_memory=False.
  data = pd.read_csv("C:/Users/Aakash/Desktop/SCMA/NSSO68.csv")
```

	slno	grp	Round_Centre	FSU_number	Round	Schedule_Number	Sample	Sector	state	State_Region	...	pickle_v	sauce_jam_v	Othrprocessed_v	Beveragestotal_v	foodtotal_v	foodtotal_q	state_1	Region	fruits_df_tt_v	fv_tot
0	1	4099999999999999265249529377587 2.0	1	41000	68	10	1	2	24	242	...	0.0	0.0	0.0	0.000000	1141.492400	30.942394	GUJ	2	12.000000	154.18
1	2	4099999999999999265249529377587 2.0	1	41000	68	10	1	2	24	242	...	0.0	0.0	0.0	17.500000	1244.553500	29.286153	GUJ	2	333.000000	484.95
2	3	4099999999999999265249529377587 2.0	1	41000	68	10	1	2	24	242	...	0.0	0.0	0.0	0.000000	1050.315400	31.527046	GUJ	2	35.000000	214.84
3	4	4099999999999999265249529377587 2.0	1	41000	68	10	1	2	24	242	...	0.0	0.0	0.0	33.333333	1142.591667	27.834607	GUJ	2	168.333333	302.30
4	5	4099999999999999265249529377587 2.0	1	41000	68	10	1	2	24	242	...	0.0	0.0	0.0	75.000000	945.249500	27.600713	GUJ	2	15.000000	148.00

5 rows × 384 columns

```python
# Select relevant columns
selected_cols = ["MPCE_URP", "Age", "Sex", "Education", "Religion", "hhdsz"]
data_selected = data[selected_cols]
```

[48]

```python
# Handle missing values
data_selected = data_selected.dropna()
```

[49]

```python
# Convert categorical variables to dummy variables
data_selected = pd.get_dummies(data_selected, columns=['Sex', 'Religion', 'Education'], drop_first=True)
```

[50]

```python
# Ensure all data is in numeric format
data_selected = data_selected.apply(pd.to_numeric, errors='coerce')
```

[51]

```python
# Define censored data (left and right bounds)
censored = (data_selected["MPCE_URP"] == 0)  # Assuming censoring at 0
```

[52]

```python
# Scale the data
scaler = StandardScaler()
data_selected_scaled = scaler.fit_transform(data_selected.drop(columns=["MPCE_URP"]))
```

```python
# Define the Tobit likelihood function
def tobit_likelihood(params, y, X, censored):
    beta = params[:-2]
    sigma = params[-2]
    nu = params[-1]

    mu = np.dot(X, beta)
    resid = (y - mu) / sigma
    pdf = norm.pdf(resid)
    cdf = norm.cdf(nu)

    LL = np.sum(np.log(np.where(censored, cdf, pdf / sigma)))
    return -LL
```

[54]

```python
# Initial guess for parameters
mean_y = np.mean(data_selected["MPCE_URP"])
initial_params = np.hstack((np.zeros(data_selected.shape[1]), [1.0, mean_y]))

# Exogenous variables
X = sm.add_constant(data_selected_scaled)

# Endogenous variable
y = data_selected["MPCE_URP"]
```

[55]

```python
# Estimate parameters using maximum likelihood estimation (MLE)
results = opt.minimize(tobit_likelihood, initial_params, args=(y, X, censored), method='BFGS')

# Extract estimated parameters
beta_est = results.x[:-2]
sigma_est = results.x[-2]
nu_est = results.x[-1]
```

[56]

```
C:\Users\Aakash\AppData\Local\Temp\ipykernel_13848\4082158152.py:12: RuntimeWarning: divide by zero encountered in log
  LL = np.sum(np.log(np.where(censored, cdf, pdf / sigma)))
c:\Users\Aakash\anaconda3\Lib\site-packages\scipy\optimize\_numdiff.py:576: RuntimeWarning: invalid value encountered in subtract
  df = fun(x) - f0
```

```python
# Print results
print("Coefficients:")
print(pd.DataFrame({"Variable": ["const"] + data_selected.drop(columns=["MPCE_URP"]).columns.tolist(),
                    "Coefficient": beta_est}))

# Summary of the model
print("\nSigma (Standard deviation of errors):", sigma_est)
print("Nu (Location parameter):", nu_est)
```

```
Coefficients:
          Variable  Coefficient
0            const          0.0
1              Age          0.0
2            hhdsz          0.0
3            Sex_2          0.0
4     Religion_2.0          0.0
5     Religion_3.0          0.0
6     Religion_4.0          0.0
7     Religion_5.0          0.0
8     Religion_6.0          0.0
9     Religion_7.0          0.0
10    Religion_9.0          0.0
11   Education_2.0          0.0
12   Education_3.0          0.0
13   Education_4.0          0.0
14   Education_5.0          0.0
15   Education_6.0          0.0
16   Education_7.0          0.0
17   Education_8.0          0.0
18  Education_10.0          0.0
19  Education_11.0          0.0
20  Education_12.0          0.0
21  Education_13.0          0.0

Sigma (Standard deviation of errors): 1.0
Nu (Location parameter): 2050.83056762287
```

Interpretation:

The script performs a Tobit regression on the "NSSO68.csv" dataset, focusing on variables like MPCE_URP, Age, Sex, Education, Religion, and hhdsz. The results show no significant impact of these variables on MPCE_URP. The standard deviation of errors (Sigma) is estimated at 1.0, and the location parameter (Nu) at approximately 2050.83, suggesting moderate model variability and a central tendency in predicted values.

Real-World Use Cases of Tobit Model:

The Tobit model is used in situations where the dependent variable is censored or limited. Here are some real-world use cases:

- **Consumer Expenditure:** Analyzing spending on luxury items where many consumers spend nothing.
- **Credit Scoring:** Studying loan amounts or defaults, especially when some applicants have zero loan amounts or defaults.
- **Health Economics:** Estimating healthcare spending where some individuals have zero expenditures.
- **Housing Market:** Assessing housing prices or rents in regions with no transactions in certain price ranges.
- **Environmental Economics:** Examining pollution control expenditures when some firms report zero spending.
- **Survey Data**: Handling survey responses where many report zero values due to privacy or non-eligibility.
- **Insurance:** Evaluating insurance claims where a significant portion of policyholders may not file any clai