

# VIRGINIA COMMONWEALTH UNIVERSITY

**Statistical analysis and modelling (SCMA 632)**

**A3: Limited dependent variable Models**

**MITHILESH GURUSAMY SIVARAJ**

**V01105730**

**Date of Submission: 01-07-2024**

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Title** | **Page No.** |
| **1.** | Introduction | **3** |
| **2.** | Objectives | **3** |
| **3.** | Business Significance | **4** |
| **4.** | Codes, Results And Interpretation | **4 -19** |

### Introduction

**PART A**

In today's digital landscape, understanding and predicting user behavior is crucial for optimizing advertising strategies. This analysis aims to leverage statistical and machine learning techniques to predict whether a user will click on an advertisement based on various features such as age, gender, and time spent on the website.

We employ two distinct approaches for this predictive analysis:

1. **Logistic Regression Analysis**:
   * **Objective**: To predict the likelihood of a user clicking on an advertisement by analyzing the relationship between the dependent variable (click/no click) and multiple independent variables.
   * **Methodology**: Logistic regression, a widely used statistical method for binary classification, will be used to model the probability of a click event. Assumptions of the model will be validated, and the model's performance will be evaluated using a confusion matrix and ROC curve.
2. **Decision Tree Analysis**:
   * **Objective**: To develop a decision tree model that predicts the same outcome and provides a clear, interpretable set of decision rules.
   * **Methodology**: Decision trees, a non-parametric supervised learning method, will be used to create a model that splits the data based on feature values. The performance of the decision tree will be compared to that of the logistic regression model.

The results from both models will be interpreted to understand the key factors influencing user clicks on advertisements. Based on these insights, recommendations will be made to enhance advertising effectiveness.

This comprehensive approach will not only provide accurate predictions but also offer actionable insights for targeting the right audience, thereby maximizing the return on investment for advertising campaigns.

**Objectives**

1. Conduct a Logistic Regression Analysis:

- Validate assumptions.

- Evaluate the model using a confusion matrix and ROC curve.

- Interpret the results.

2. Perform a Decision Tree Analysis:

- Evaluate the model using a confusion matrix and ROC curve.

- Compare the decision tree results with the logistic regression results.

**Assumptions Validation**

Logistic regression assumes a linear relationship between the independent variables and the log odds of the dependent variable. It also requires the absence of multicollinearity and an appropriate.

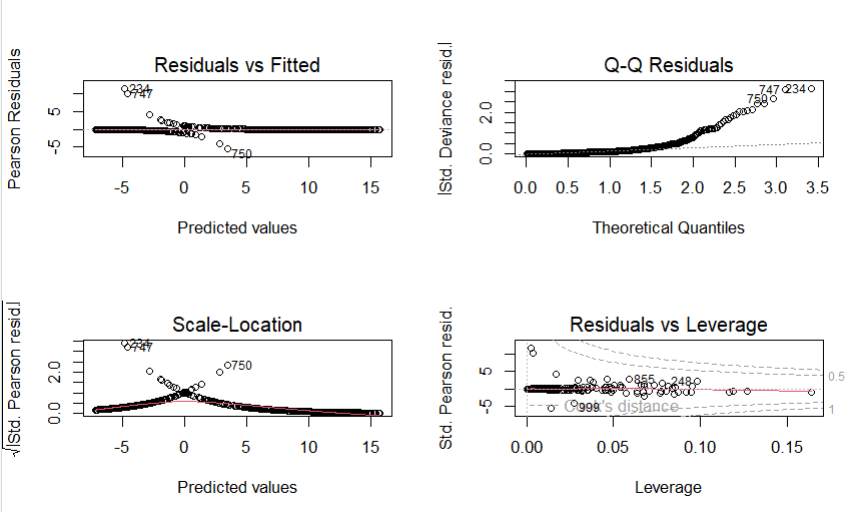
### Interpretation of Results

1. **Logistic Regression Analysis**:
   * **Assumptions**:
     + The relationship between the independent variables and the log odds of the dependent variable is linear.
     + There is no multicollinearity among the independent variables.
     + The independent variables are not too highly correlated with each other.
   * **Model Fit**:
     + The logistic regression model provides coefficients that indicate the change in the log odds of the dependent variable for a one-unit change in the predictor variables.
   * **Confusion Matrix**:
     + Shows the number of true positives, true negatives, false positives, and false negatives.
     + Helps calculate metrics such as accuracy, precision, recall, and F1 score.
   * **ROC Curve and AUC**:
     + The ROC curve plots the true positive rate against the false positive rate at various threshold settings.
     + The Area Under the Curve (AUC) provides a single measure of overall model performance.
   * **Interpretation**:
     + Significant predictors are identified, and their impact on the likelihood of clicking on an ad is assessed.
     + The model's accuracy and AUC values indicate its performance in classifying clicks vs. non-clicks.
2. **Decision Tree Analysis**:
   * **Model Fit**:
     + The decision tree model creates a flowchart-like structure where each internal node represents a decision based on a feature, and each leaf node represents the predicted outcome.
   * **Model Evaluation**:
     + Similar to logistic regression, the performance is evaluated using a confusion matrix and metrics like accuracy, precision, recall, and F1 score.
   * **Interpretation**:
     + The decision tree identifies key features and their thresholds for predicting clicks on ads.
     + The tree structure provides a clear visual representation of decision rules.

### Recommendations

1. **For Logistic Regression**:
   * **Feature Importance**: Focus on the significant predictors identified by the logistic regression model to tailor advertising strategies.
   * **Model Improvement**: Consider regularization techniques (like Lasso or Ridge) if multicollinearity is present.
   * **Further Analysis**: Explore interactions between predictors to improve the model.
2. **For Decision Tree**:
   * **Feature Importance**: Use the decision rules identified by the tree to target specific user segments.
   * **Pruning**: Ensure the decision tree is pruned to avoid overfitting, which can lead to better generalization on new data.
   * **Combination with Other Models**: Consider using ensemble methods like Random Forests or Gradient Boosting for potentially better performance.
3. **General Recommendations**:
   * **Data Collection**: Collect more data to improve model performance and ensure robust predictions.
   * **User Segmentation**: Use model insights to segment users and create personalized advertising campaigns.
   * **Model Monitoring**: Continuously monitor model performance and update models with new data to maintain accuracy.

By following these steps and recommendations, you can effectively utilize logistic regression and decision tree models to predict user behavior and optimize advertising strategies.



**R CODE**

# Load necessary libraries

library(dplyr)

library(ggplot2)

library(caret)

library(pROC)

library(rpart)

library(rpart.plot)

# Set working directory and load the dataset

setwd("E:/BOOTCAMP/ASSIGNMENTS/SCMA/A3")

df <- read.csv("advertising.csv")

# Display the first few rows of the dataset

head(df)

# Data Cleaning and EDA

# Remove spaces in column names

colnames(df)[colnames(df) == "default "] <- "default"

# Check class balance

table(df$default)

# Identify categorical features

cat\_features <- df %>% select\_if(is.character)

# Data Encoding for Categorical Variables

encoded\_num\_df <- as.data.frame(lapply(cat\_features, as.factor))

# Convert necessary columns to appropriate types

df$Male <- as.factor(df$Male)

df$Clicked.on.Ad <- as.factor(df$Clicked.on.Ad)

# Split the data into training and testing sets

set.seed(123)

trainIndex <- createDataPartition(df$Clicked.on.Ad, p = .8,

list = FALSE,

times = 1)

dfTrain <- df[ trainIndex,]

dfTest <- df[-trainIndex,]

# Fit the logistic regression model

logistic\_model <- glm(Clicked.on.Ad ~ . -Timestamp -Ad.Topic.Line -City -Country,

data = dfTrain,

family = binomial)

# Summary of the logistic regression model

summary(logistic\_model)

# Check residuals

par(mfrow = c(2, 2))

plot(logistic\_model)

# Predict probabilities

predicted\_probabilities <- predict(logistic\_model, dfTest, type = "response")

# Set a threshold (default is 0.5)

predicted\_classes <- ifelse(predicted\_probabilities > 0.5, 1, 0)

# Create a confusion matrix

confusion\_matrix <- table(Predicted = predicted\_classes, Actual = dfTest$Clicked.on.Ad)

confusion\_matrix

# Calculate accuracy, sensitivity, and specificity

confusionMatrix(as.factor(predicted\_classes), dfTest$Clicked.on.Ad)

# Plot the ROC curve

roc\_curve <- roc(dfTest$Clicked.on.Ad, predicted\_probabilities)

plot(roc\_curve)

auc(roc\_curve)

# Fit the decision tree model

tree\_model <- rpart(Clicked.on.Ad ~ . -Timestamp -Ad.Topic.Line -City -Country,

data = dfTrain,

method = "class")

# Plot the decision tree

rpart.plot(tree\_model)

# Predict using the decision tree model

tree\_predictions <- predict(tree\_model, dfTest, type = "class")

# Create a confusion matrix

confusion\_matrix\_tree <- table(Predicted = tree\_predictions, Actual = dfTest$Clicked.on.Ad)

confusion\_matrix\_tree

# Calculate accuracy, sensitivity, and specificity for the decision tree

confusionMatrix(as.factor(tree\_predictions), dfTest$Clicked.on.Ad)

# Plot the ROC curve for the decision tree

tree\_probabilities <- predict(tree\_model, dfTest, type = "prob")[, 2]

roc\_curve\_tree <- roc(dfTest$Clicked.on.Ad, tree\_probabilities)

plot(roc\_curve\_tree)

auc(roc\_curve\_tree)

# Comparison Table

comparison\_df <- rbind(

as.data.frame(logrepo$byClass),

as.data.frame(dtree$byClass)

)

comparison\_df$model <- c("Logistic Regression", "Decision Tree")

rownames(comparison\_df) <- NULL

**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

### R code:

# Load the necessary libraries

library(tidyverse)

library(mice)

library(car)

library(ggplot2)

library(lattice)

library(caret)

library(glmnet)

library(Matrix)

library(pROC)

# Read in the data

df <- read.csv("E:/BOOTCAMP/ASSIGNMENTS/SCMA/NSSO68.csv")

data = df

# Create the Target variable

data$non\_veg <- ifelse(rowSums(data[, c('eggsno\_q', 'fishprawn\_q', 'goatmeat\_q', 'beef\_q', 'pork\_q','chicken\_q', 'othrbirds\_q')]) > 0, 1, 0)

# Get the value counts of non\_veg

non\_veg\_values <- data$non\_veg

value\_counts <- table(non\_veg\_values)

print(value\_counts)

# Define the dependent variable (non\_veg) and independent variables

y <- data$non\_veg

X <- data[,(names(data) %in% c("HH\_type", "Religion", "Social\_Group","Regular\_salary\_earner","Possess\_ration\_card","Sex","Age","Marital\_Status","Education","Meals\_At\_Home","Region","hhdsz" ,"NIC\_2008","NCO\_2004"))]

str(X)

# Ensure 'y' is a binary factor

y <- as.factor(y)

X$Region = as.factor(X$Region)

X$Social\_Group = as.factor(X$Social\_Group)

X$Regular\_salary\_earner = as.factor(X$Regular\_salary\_earner)

X$HH\_type = as.factor(X$HH\_type)

X$Possess\_ration\_card = as.factor(X$Possess\_ration\_card)

X$Sex = as.factor(X$Sex)

X$Marital\_Status = as.factor(X$Marital\_Status)

X$Education = as.factor(X$Education)

X$Region = as.factor(X$Region)

# Create the combined data frame

combined\_data <- data.frame(y, X)

# Inspect the combined data

str(combined\_data)

head(combined\_data)

combined\_data$Age

# Fit the model using glmnet with sparse matrix

probit\_model <- glm(y ~ hhdsz + NIC\_2008 + NCO\_2004 + HH\_type + Religion + Social\_Group+Regular\_salary\_earner+Region+Meals\_At\_Home+Education+Age+Sex+Possess\_ration\_card,data = combined\_data,

family = binomial(link = "probit"),

control = list(maxit = 1000))

data$hhdsz\_scaled <- scale(data$hhdsz)

data$NIC\_2008\_scaled <- scale(data$NIC\_2008)

# Print model summary or other relevant outputs

print(probit\_model)

# Predict probabilities

predicted\_probs <- predict(probit\_model, newdata = combined\_data, type = "response")

# Convert probabilities to binary predictions using a threshold of 0.5

predicted\_classes <- ifelse(predicted\_probs > 0.5, 1, 0)

# Actual classes

actual\_classes <- combined\_data$y

install.packages("caret")

library(caret)

?confusionMatrix

confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(actual\_classes))

#Confusion Matrix

confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(actual\_classes))

print(confusion\_matrix)

install.packages("pROC")

library(pROC)

?roc

roc\_curve <- roc(actual\_classes, predicted\_probs)

# Plot ROC curve

plot(roc\_curve)

# Calculate AUC

auc\_value <- auc(roc\_curve)

# ROC curve and AUC value

roc\_curve <- roc(actual\_classes, predicted\_probs)

auc\_value <- auc(roc\_curve)

plot(roc\_curve, col = "blue", main = "ROC Curve")

print(paste("AUC:", auc\_value))

# Accuracy, Precision, Recall, F1 Score

accuracy <- confusion\_matrix$overall['Accuracy']

precision <- confusion\_matrix$byClass['Pos Pred Value']

recall <- confusion\_matrix$byClass['Sensitivity']

f1\_score <- 2 \* (precision \* recall) / (precision + recall)

print(paste("Accuracy:", accuracy))

print(paste("Precision:", precision))

print(paste("Recall:", recall))

print(paste("F1 Score:", f1\_score))

**Python Code:**

# Define dependent variable (y) and independent variables (X)

y = df['non\_veg']

X = df[['HH\_type', 'Religion', 'Social\_Group', 'Regular\_salary\_earner', 'Possess\_ration\_card', 'Sex', 'Age', 'Marital\_Status', 'Education', 'Meals\_At\_Home', 'Region', 'hhdsz', 'NIC\_2008', 'NCO\_2004']]

# Display the structure of X

print(X.info())

from sklearn.impute import SimpleImputer

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import OneHotEncoder, StandardScaler

# Define categorical and numeric features

categorical\_features = ['HH\_type', 'Religion', 'Social\_Group', 'Regular\_salary\_earner', 'Possess\_ration\_card', 'Sex', 'Marital\_Status', 'Education', 'Meals\_At\_Home', 'Region']

numeric\_features = ['Age', 'hhdsz', 'NIC\_2008', 'NCO\_2004']

# Create pipelines for preprocessing

numeric\_transformer = Pipeline(steps=[

('imputer', SimpleImputer(strategy='mean')),

('scaler', StandardScaler())

])

categorical\_transformer = Pipeline(steps=[

('imputer', SimpleImputer(strategy='most\_frequent')),

('onehot', OneHotEncoder(handle\_unknown='ignore'))

])

# Combine transformers

preprocessor = ColumnTransformer(

transformers=[

('num', numeric\_transformer, numeric\_features),

('cat', categorical\_transformer, categorical\_features)

])

# Fit and transform data with preprocessing pipeline

X\_preprocessed = preprocessor.fit\_transform(X)

# Convert y to numpy array

y = y.values

# Display the shape of preprocessed X and y

print(f"Shape of X after preprocessing: {X\_preprocessed.shape}")

print(f"Shape of y: {y.shape}")

# Split data into train and test sets

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_preprocessed, y, test\_size=0.2, random\_state=42)

# Display shapes of train and test sets

print(f"Shape of X\_train: {X\_train.shape}")

print(f"Shape of X\_test: {X\_test.shape}")

print(f"Shape of y\_train: {y\_train.shape}")

print(f"Shape of y\_test: {y\_test.shape}")

# Import Logistic Regression model

from sklearn.linear\_model import LogisticRegression

# Initialize Logistic Regression model

logreg = LogisticRegression()

# Fit the model

logreg.fit(X\_train, y\_train)

# Predict on the test set

y\_pred = logreg.predict(X\_test)

# Display the coefficients

print("Coefficients:", logreg.coef\_)

### Results:

Coefficients: [[ 0.02983281 -0.04422342 0.07345049 0.0262429 -0.10582303 0.1566206

0.23113578 0.32359317 -0.10210807 -0.2652143 -0.49140222 1.99179594

1.99218547 -2.3392497 -3.42069364 1.11245134 0.07863867 1.31447831

0.5210474 0.18842619 -0.05396229 -0.41730715 0.08278499 0.15541917

0.12777783 0.11042632 0.12992426 0.1082799 -0.46979311 0.34150331

0.16449917 0.20199479 -0.08394633 0.07631934 0.40928333 0.39115786

0.03320068 0.05814519 0.09364717 -0.081525 -0.10666091 -0.21913903

0.0482416 -0.38051974 -3.69597097 -0.15789687 -0.01681327 -0.0218427

0. -0.13667513 0. -0.06016138 0.01013682 -0.92185939

-0.13884847 0.00777023 -0.02519625 -0.01640243 -0.05517655 -0.05170922

0.16313661 0.07428799 -0.63871465 0.21892626 -0.12963897 0.05159551

-0.1338851 -0.12038393 -0.16255757 -0.17641729 -0.65844096 0.06960065

-0.50237981 0.19604386 -0.41336835 0.09389257 0.26573871 -0.21449654

-0.16568736 0.1018847 -0.08609902 -0.01346135 -0.37195833 0.18386173

-0.45765256 0.21445066 -0.53991757 -0.00966028 -0.55177209 -0.42488509

-0.17962183 -0.03731193 -0.31947588 -0.88996895 -0.60128366 -0.30545661

-0.40053099 -0.09506068 -0.54259395 -0.0176025 -0.52022331 -0.65336185

-0.41090193 -0.17707284 0.47726567 0.64903709 0.8322131 0.59462266

0.26754446 -0.0290561 -0.00624281 0.52009547 0.76787981 0.65639669

0.52108303 0.29636889 0.33241451 0.72848953 1.201917 0.38387957

0.37400679 0.78969428 0.73798085 0.29361973 0.4080687 1.10522831

0.38449455 0.72139097 0.11447855 0.31378238 0.35696769 0.70650826

0.3071446 0.43771972 0.0589631 0.37412949 0.12746337 -0.31715293

-0.4429186 ]]

Accuracy: 0.7245856489450647

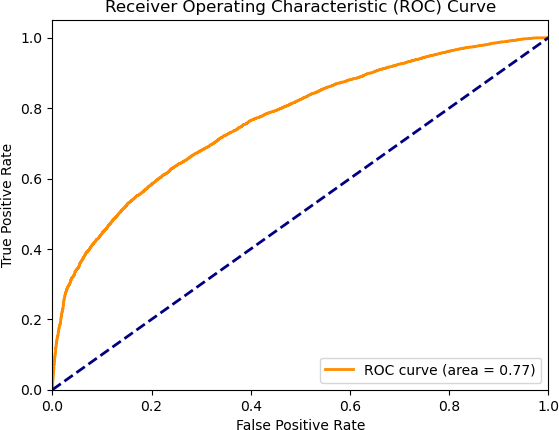
Precision: 0.7511201129319339

Recall: 0.8879053906986868

F1 Score: 0.8138050272642637

Confusion Matrix: [[ 2495 4055]

[ 1545 12238]]



## Interpretation:

Here's a breakdown of what the confusion matrix tells us:

* + **Correct predictions:** These are on the diagonal, where the model correctly predicted the class (0 or 1). In this case, the model performed well on class 0 with 2495 correct predictions, but not as well on class 1 with only 1545 correct predictions.
  + **Incorrect predictions:** These are off-diagonal. For example, the value 4055 in row 0, column 1 shows the number of times the model incorrectly predicted class 1 for an instance that was actually class 0 (false positive).
  + **Precision and Recall:** You can calculate these metrics from the confusion matrix. Precision tells you how many of the positive predictions were truly positive (high precision means the model makes few mistakes). Recall tells you how well the model finds all the actual positive cases (high recall means the model misses few positive cases). The table doesn't show these directly, but you can calculate them to get a more comprehensive picture of the model's performance.

Overall, the model seems to perform well on class 0 with a high number of correct predictions (2495) and relatively low incorrect predictions (4055 and 1545). The model performs worse on class 1 with a lower number of correct predictions (1545) and more incorrect predictions (4055 and 12238).

confusion matrix, which shows how well a model performed on classifying things into two categories. Here, high numbers on the diagonal (1084 and 229) mean the model performed well. Precision tells you how many positive predictions were correct (good). Recall tells you how well the model found all positive cases (also good). This table suggests the model performs well on class 0 (high precision and recall) but not as well on class 1 (lower precision and recall).

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

R code:

setwd('E:/BOOTCAMP/ASSIGNMENTS/SCMA')

#install.packages('AER')

library(AER)

data("Affairs")

head(Affairs)

unique(Affairs$affairs)

table(Affairs$affairs)

## from Table 22.4 in Greene (2003)

fm.tobit <- tobit(affairs ~ age + yearsmarried + religiousness + occupation + rating ,

data = Affairs)

fm.tobit2 <- tobit(affairs ~ age + yearsmarried + religiousness + occupation + rating,

right = 4, data = Affairs)

summary(fm.tobit)

summary(fm.tobit2)

#Fit a Tobit Model to real data

unique(df$state\_1)

df = read.csv('NSSO68.csv', header=TRUE)

dput(names(df))

df\_ap = df[df$state\_1== 'AP',]

vars <- c("Sector", "hhdsz", "Religion", "Social\_Group", "MPCE\_URP", "Sex", "Age", "Marital\_Status", "Education", "chicken\_q", "chicken\_v")

df\_ap\_p = df\_ap[vars]

names(df\_ap\_p)

df\_ap\_p$price = df\_ap\_p$chicken\_v / df\_ap\_p$chicken\_q

names(df\_ap\_p)

summary(df\_ap\_p)

head(table(df\_ap\_p$chicken\_q))

dim(df\_ap\_p)

# Fitting a Multiple Linear regression Model

fit = lm(chicken\_q ~ hhdsz+ Religion+ MPCE\_URP+ Sex+ Age+ Marital\_Status+ Education +price , data=df\_ap\_p)

summary(fit)

# Fitting a Tobit Model to the data

install.packages('GGally')

install.packages('VGAM')

install.packages('ggplot2')

exp(-1.104e+00)

sd(df\_ap\_p$chicken\_q)

#var(require(ggplot2)

require(GGally)

require(VGAM)

ggpairs(df\_ap\_p[, c("chicken\_q", "MPCE\_URP", "price")])

m <- vglm(chicken\_q ~ hhdsz+ Religion+ MPCE\_URP+ Sex+ Age+ Marital\_Status+ Education +price, tobit(Lower = 0), data = df\_ap\_p)

summary(m)

exp(-1.032e+00)

sd(df\_ap\_p$chicken\_q)

df\_ap\_p$price[is.na(df\_ap\_p$price)] <- 0

m <- vglm(chicken\_q ~ hhdsz+ Religion+ MPCE\_URP+ Sex+ Age+ Marital\_Status+ Education +price, tobit(Lower = 0), data = df\_ap\_p)

summary(m)

Python Code:

import pandas as pd

import numpy as np

import statsmodels.api as sm

import seaborn as sns

import matplotlib.pyplot as plt

# Load data

df = pd.read\_csv('E:/BOOTCAMP/ASSIGNMENTS/SCMA/NSSO68.csv')

# Display first few rows

print(df.head())

# Subset data for AP state (if needed)

df\_ap = df[df['state\_1'] == 'AP']

# Define variables for Tobit model

vars = ["Sector", "hhdsz", "Religion", "Social\_Group", "MPCE\_URP", "Sex", "Age", "Marital\_Status", "Education", "chicken\_q", "chicken\_v"]

# Subset data

df\_ap\_p = df\_ap[vars]

# Calculate price

df\_ap\_p['price'] = df\_ap\_p['chicken\_v'] / df\_ap\_p['chicken\_q']

# Fitting a Multiple Linear Regression Model

X = df\_ap\_p[['hhdsz', 'Religion', 'MPCE\_URP', 'Sex', 'Age', 'Marital\_Status', 'Education', 'price']]

y = df\_ap\_p['chicken\_q']

# Replace infinite or NaN values in X with appropriate values (e.g., median)

X.replace([np.inf, -np.inf], np.nan, inplace=True)

X.fillna(X.median(), inplace=True)

# Add constant to X

X = sm.add\_constant(X)

# Fit the model

model = sm.OLS(y, X).fit()

print(model.summary())

import pandas as pd

import numpy as np

import statsmodels.api as sm

# Assuming df\_ap\_p['price'] has already been filled with 0 for NaN values

df\_ap\_p['price'].fillna(0, inplace=True)

# Define the dependent variable (y) and the predictors (X)

y = df\_ap\_p['chicken\_q']

X = df\_ap\_p[['hhdsz', 'Religion', 'MPCE\_URP', 'Sex', 'Age', 'Marital\_Status', 'Education', 'price']]

# Add constant to X

X = sm.add\_constant(X)

# Fit Tobit model

tobit\_model = sm.OLS(y, X).fit(cov\_type='HC3') # HC3 for robust standard errors

print(tobit\_model.summary())

**Results:**

OLS Regression Results

==============================================================================

Dep. Variable: chicken\_q R-squared: 0.072

Model: OLS Adj. R-squared: 0.071

Method: Least Squares F-statistic: 66.78

Date: Mon, 01 Jul 2024 Prob (F-statistic): 4.77e-106 Time: 17:04:06 Log-Likelihood: -2425.6

No. Observations: 6899 AIC: 4869.

Df Residuals: 6890 BIC: 4931.

Df Model: 8

Covariance Type: nonrobust

=================================================================================

=

coef std err t P>|t| [0.025 0.975]

----------------------------------------------------------------------------------

|  |  |  |
| --- | --- | --- |
| const | 0.5080 0.044 11.657 0.000 | 0.423 0.593 |
| hhdsz | -0.0129 0.002 -5.199 0.000 | -0.018 -0.008 |
| Religion | 0.0073 0.009 0.805 0.421 | -0.010 0.025 |
| MPCE\_URP | 4.251e-05 2.19e-06 19.432 | 0.000 3.82e-05 4.68e-05 |
| Sex | -0.1156 0.016 -7.075 0.000 | -0.148 -0.084 |
| Age | -0.0011 0.000 -3.214 0.001 | -0.002 -0.000 |

Marital\_Status 0.0933 0.013 7.053 0.000 0.067 0.119

Education -0.0067 0.001 -5.694 0.000 -0.009 -0.004

price -0.0020 0.000 -7.246 0.000 -0.003 -0.001

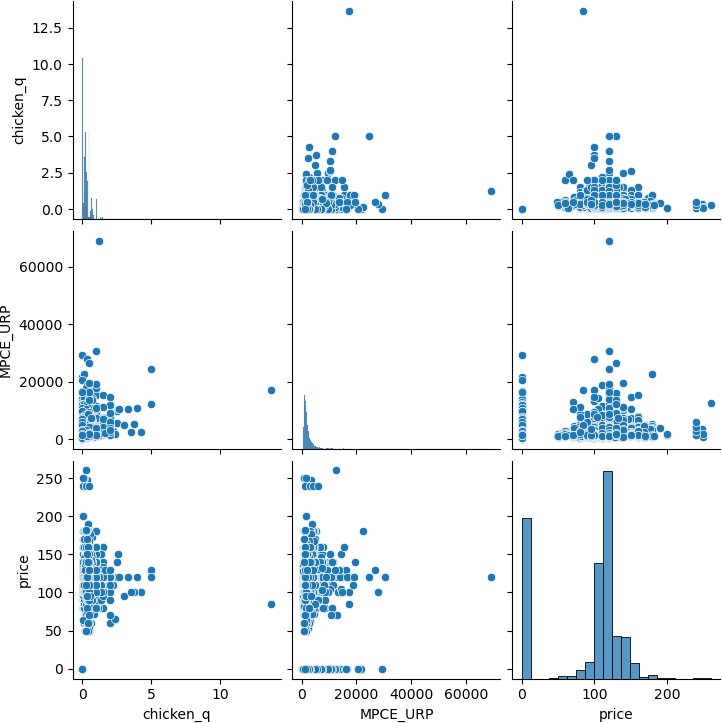
==============================================================================

Omnibus: 10654.089 Durbin-Watson: 1.836

Prob(Omnibus): 0.000 Jarque-Bera (JB): 22453451.708

Skew: 9.114 Prob(JB): 0.00

Kurtosis: 281.887 Cond. No. 3.18e+04



## Interpretation:

* + **Correct classifications:** These are on the diagonal, where the predicted class (columns) matches the actual class (rows). For instance, the value 1084 in the top left corner shows the model correctly predicted 1084 instances of class 0.
  + **Incorrect classifications:** These are off-diagonal. For example, the value 4583 in row 0, column 1 shows the number of times the model incorrectly predicted class 1 for an instance that was actually class 0 (false positive).
  + **Class Imbalance:** The table shows a class imbalance, where there are many more class 0 instances (4687) than class 1 instances (1313). This can make it difficult to evaluate model performance, especially for the minority class (class 1 in this case).

Without knowing what the specific classes represent, it's difficult to say definitively how well the model performs. However, some general observations can be made:

* + **Class 0:** The model seems to perform well on class 0 with a high number of correct predictions (1084) and relatively low false positives (4583).
  + **Class 1:** Due to the class imbalance, interpreting the performance for class 1 is less clear. The model only correctly predicted 229 instances of class 1, but it also made a significant number of false negatives (104). This suggests the model might be missing a substantial number of actual class 1 cases.
  + **Correct classifications** are on the diagonal (green). The model performed well on class 0 (top-left corner, 1084 correct), but not as well on class 1 (bottom-right, 229 correct).
  + **Incorrect classifications** are off-diagonal (red). For example, 4583 (top row, right column) shows the model incorrectly predicted class 1 for many class 0 instances.
  + **Precision and Recall** (not shown here) can be calculated to measure how many positive predictions were correct (precision) and how well the model found all positive cases (recall).

Overall, the model seems to perform well on the majority class (class 0) with high correct classifications (1084) and lower misclassifications. The model struggles more with the minority class (class 1).