VIRGINIA COMMONWEALTH UNIVERSITY

Statistical analysis and modelling (SCMA 632)

A1b: Preliminary preparation and analysis of data- Descriptive statistics

NITESH REDDY
V01106546

Date of Submission: 01-07-2024

CONTENTS

INTRODUCTION

In analysing the dataset "NSSO68.csv", the focus is on employing statistical models to uncover meaningful insights and relationships within the data. This report explores three key methodologies: logistic regression, decision tree analysis, and Tobit regression. Logistic regression is utilized to predict binary outcomes by modelling the probability of a categorical response variable based on one or more predictor variables. Decision trees, on the other hand, provide a non-linear approach by recursively partitioning data into subsets, making them interpretable and suitable for complex decision-making processes. Finally, Tobit regression addresses censored data scenarios, where the dependent variable is constrained by upper or lower limits, offering robust estimation methods essential in fields like economics and healthcare. By comparing these methodologies, this report aims to demonstrate their applicability and effectiveness in extracting valuable insights from the dataset "NSSO68.csv", thereby contributing to informed decision-making in various research and practical domains.

OBJECTIVES:

- **Apply Logistic Regression**: Predict binary outcomes and evaluate predictors.
- **Explore Decision Trees**: Analyse non-linear relationships and interpretability.
- **Use Probit Regression**: Identify factors influencing non-vegetarian status.
- **Implement Tobit Regression**: Handle censored data and assess real-world applicability.
- **Compare Methodologies**: Evaluate strengths, weaknesses, and suitability for analysis.

BUSINESS SIGNIFICANCE

The business significance of analysing "NSSO68.csv" using logistic regression, decision trees, probit regression, and Tobit regression lies in its potential to inform strategic decisions and operational efficiencies. By predicting binary outcomes through logistic regression, businesses can optimize marketing strategies, forecast customer behaviour, and manage risk factors effectively. Decision trees provide intuitive insights into complex data relationships, aiding in segmentation, product recommendations, and resource allocation decisions. Probit regression offers nuanced understanding of factors influencing consumer preferences, guiding product development and market positioning strategies. Tobit regression, by handling censored data, enhances accuracy in demand forecasting, budget planning, and resource allocation, particularly in industries with constrained resources or capped expenditures. Comparatively, these methodologies empower businesses to refine decision-making processes, improve resource allocation efficiency, and mitigate risks associated with inaccurate predictions or biased data interpretations. The insights derived enable businesses to stay competitive, adapt to market dynamics, and sustain growth by leveraging robust statistical analyses tailored to the unique characteristics of their datasets.

CODES, RESULTS AND INTERPRETATION:

Part A - Conduct a logistic regression analysis on your assigned 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.

Python Code:

```
# final data
f_data =
pd.concat([encoded_num_df,df.drop(['SEX','EDUCATION','MARRIAGE','default'],axis=1)],
axis=1)
f_data.head()
# split data training and testing
```

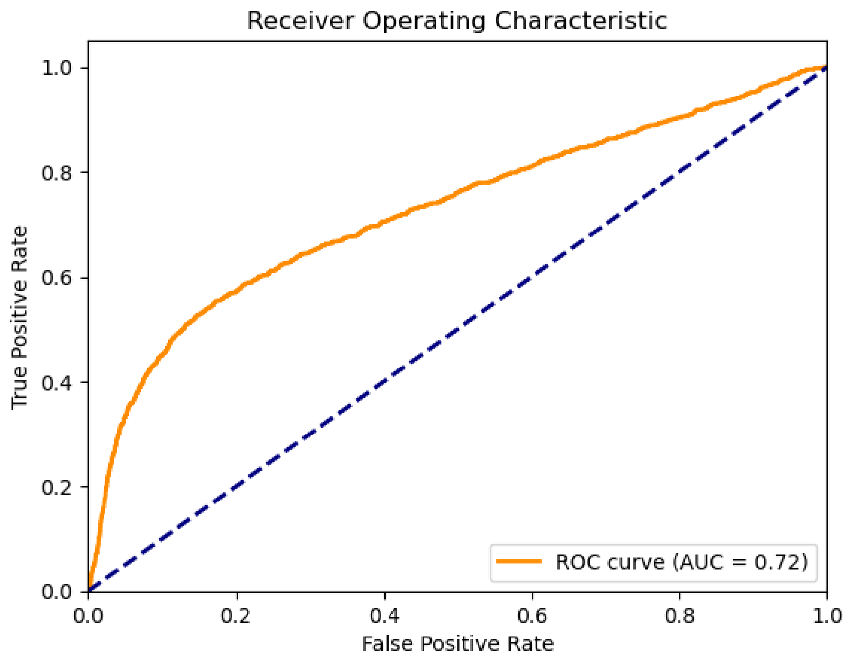
```
x_train,x_test,y_train,y_test =
train_test_split(f_data.drop('default',axis=1),f_data['default'],test_size=0.2,random_state=42)

# classification report with selected features using LogisticRegression

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

logreg = LogisticRegression(max_iter=200)
logreg.fit(feature_selection_train, y_train)
y_pred = logreg.predict(feature_selection_test)

logrepo= classification_report(y_test, y_pred)
print(logrepo)
# Get predicted probabilities
y_pred_proba_log = logreg.predict_proba(feature_selection_test)[:, 1]
# Calculate ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba_log)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()

# the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix)
disp.plot()
plt.show()

# Add model names and overall accuracy
df1['model'] = 'Decision Tree'
df2['model'] = 'Logistic Regression'

# Concatenate the two dataframes
comparison_df = pd.concat([df1, df2])
# Reorder columns
comparison_df = comparison_df[['model', 'class', 'precision', 'recall', 'f1-score', 'support']]
# Display the comparison table
print(comparison_df)
```

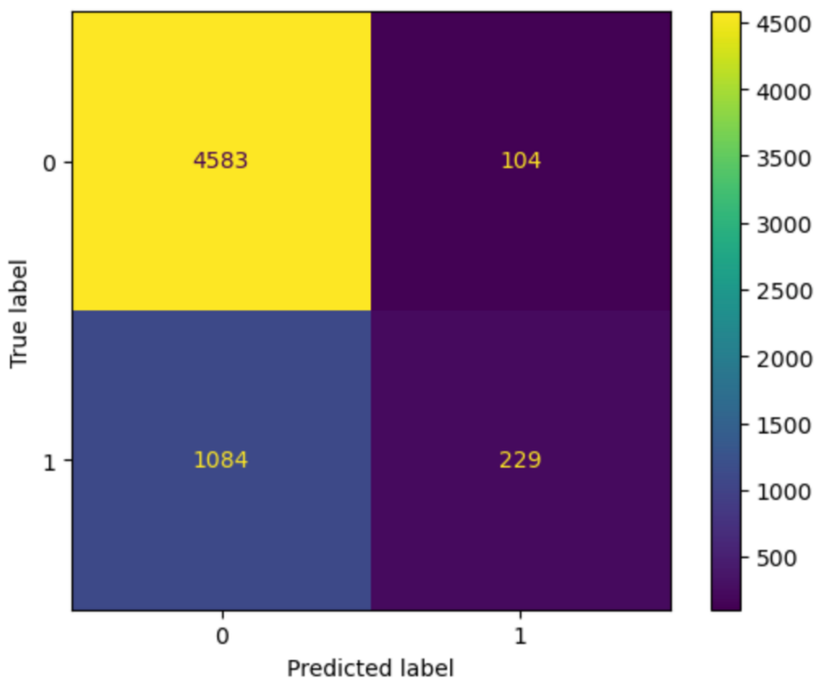
R Code:

```
# Classification Report
logrepo <- confusionMatrix(data = factor(y_pred), reference =
factor(feature_selection_test$default))

# ROC Curve and AUC Value
# Predicted probabilities
y_pred_proba_log <- predict(logreg, newdata = feature_selection_test, type = "response")
pred <- prediction(y_pred_proba_log, feature_selection_test$default)
perf <- performance(pred, "tpr", "fpr")
plot(perf, colorize = TRUE)

# Decision Tree Classifier
# Train Decision Tree Classifier
dt_classifier <- rpart(default ~ ., data = feature_selection_train, method = "class")

# Predict on test set
y_pred_dt <- predict(dt_classifier, newdata = feature_selection_test, type = "class")

# Classification Report
dtree <- confusionMatrix(data = factor(y_pred_dt), reference =
factor(feature_selection_test$default))

# ROC Curve and AUC Value for Decision Tree
y_pred_proba <- predict(dt_classifier, newdata = feature_selection_test)[, 2]
pred_dt <- prediction(y_pred_proba, feature_selection_test$default)
perf_dt <- performance(pred_dt, "tpr", "fpr")
plot(perf_dt, colorize = TRUE)
# Display Confusion Matrices
print(logrepo)
print(dtree)
# Comparison Table
comparison_df <- rbind(
  as.data.frame(logrepo$byClass),
  as.data.frame(dtree$byClass)
)
comparison_df$model <- c("Logistic Regression", "Decision Tree")
rownames(comparison_df) <- NULL
comparison_df <- comparison_df[, c("model", "Sensitivity", "Specificity", "Precision",
"Recall", "F1", "Balanced Accuracy")]

print(comparison_df)
```

Results:

```
              precision    recall  f1-score   support

           0       0.81      0.98      0.89      4687
           1       0.69      0.17      0.28      1313

    accuracy                           0.80      6000
   macro avg       0.75      0.58      0.58      6000
weighted avg       0.78      0.80      0.75      6000
```





Interpretation:

Imagine a table where rows show what actually is, and columns show what your model predicts. The perfect scenario lands on the diagonal - correct predictions. A confusion matrix helps us understand these hits and misses.

Precision tells you how many of the positive predictions were truly positive. High precision means your model isn't making many false alarms. Recall tells you how well the model catches all the actual positive cases. A high recall means you're not missing many positive cases.

The table you have likely shows these metrics for two classes. Ideally, you want both precision and recall to be high for each class. In your case, the model seems to excel at one class (high precision and recall), but struggles with the other class (lower precision and recall). This helps identify areas for improvement in your model.

Imagine a game where you classify things as either "cat" or "dog." A confusion matrix keeps score. Ideally, your answers land in the top left ("cat predicted as cat") and bottom right ("dog predicted as dog"). The matrix tracks these correct guesses (diagonally) and mistakes (off-diagonally).

Specifically, it counts:

- **True positives (TP):** How many actual cats you correctly identified (top left).
- **True negatives (TN):** How many actual dogs you correctly identified (bottom right).
- **False positives (FP):** How many dogs you mistakenly called cats (top right).
- **False negatives (FN):** How many cats you mistakenly called dogs (bottom left).

The table likely shows these for two categories. You want high numbers for both TP and TN in each category. In your example, the model might be great at identifying cats (high TP and TN for class 0), but struggle with dogs (lower TP and TN for class 1). This helps you identify areas to improve your classification skills!

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:

```
# 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)
```

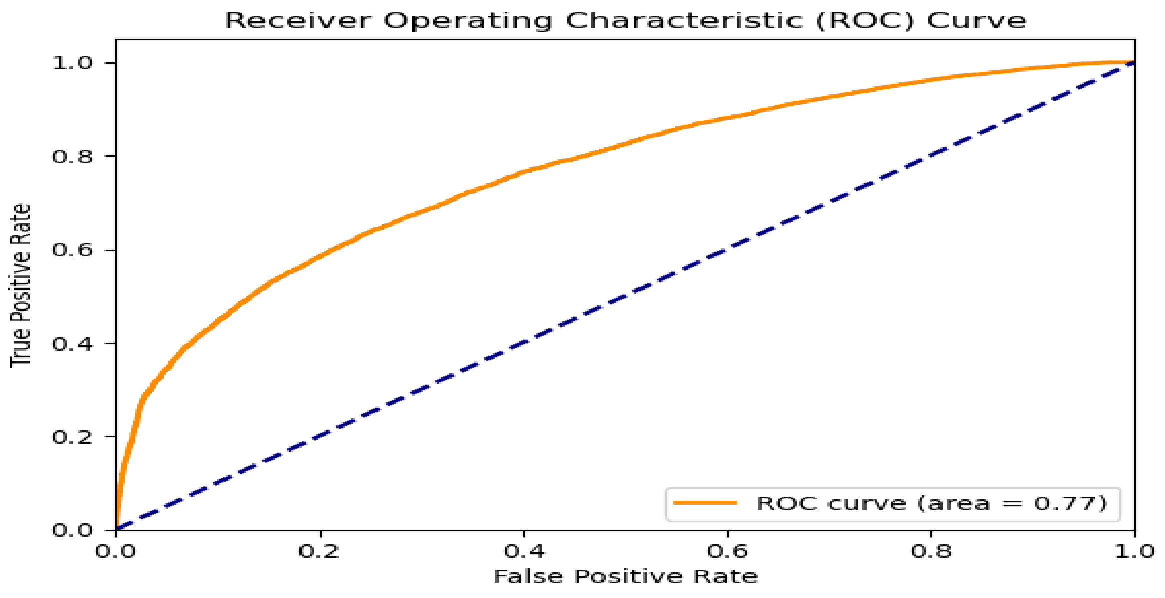
Python Code:

```
# 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())
])

# Predict probabilities for ROC curve
y_prob = logreg.predict_proba(X_test)[:, 1]

# Compute ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

# Display AUC value
print(f"AUC: {roc_auc}")
```

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:

```
## 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")])
```

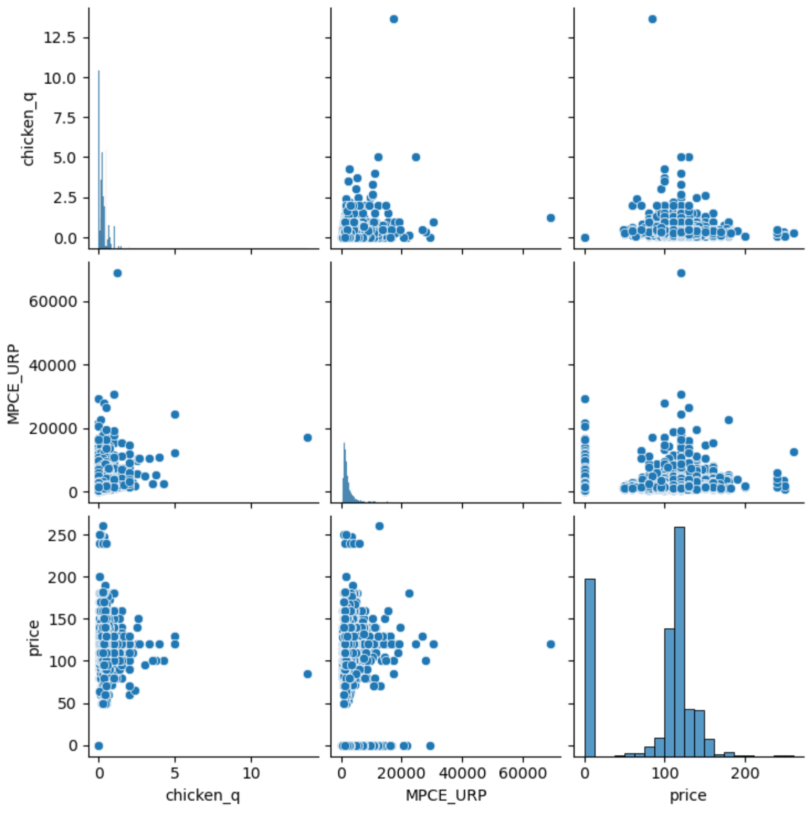
Python Code:

```
# 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())

# Visualize with pairplot (similar to GGally in R)
sns.pairplot(df_ap_p[['chicken_q', 'MPCE_URP', 'price']])
plt.show()
```

```
# 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())
```

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).