VIRGINIA COMMONWEALTH UNIVERSITY

Statistical analysis and modelling (SCMA 632)

A4- Limited dependent variable Models

Daniel Joe Gasper

V01151514

Date of Submission: 30/06/2025

CONTENTS

INTRODUCTION

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

Business Problems Addressed by Data Analysis:

A)

1. **Customer churn prediction:**
   Use logistic regression and decision trees to identify customers at high risk of leaving, so that retention campaigns can be targeted more effectively.
2. **Credit approval or default risk:**
   Analyze applicant data to predict loan default probabilities and decide approval thresholds, balancing business growth with risk.
3. **Marketing response modeling:**
   Predict the likelihood that customers will respond to a promotion, allowing more efficient allocation of marketing budgets.

B)

- **Market segmentation for food products:**

Identify demographic segments more likely to be non-vegetarian to tailor product portfolios and advertising strategies.

- **Design of nutritional outreach programs:**

Understand which groups are less likely to consume non-vegetarian food to guide fortification, protein supplementation, or awareness initiatives.

- **Forecasting demand for non-veg supply chains:**

Help food retailers or processors plan distribution and inventory by predicting regional or demographic consumption tendencies.

C)

- **Optimizing subsidy allocation:**

Identify which households rely most on subsidized rice, enabling targeted policy interventions and better allocation of limited public resources.

- **Demand forecasting for public distribution:**

Estimate the actual demand (latent consumption) for rice under the PDS, factoring in households that currently consume none but may in the future.

- **Designing exit strategies for subsidies:**

Understand how education, income, and demographics influence reliance on subsidies, to develop phased reduction plans for certain beneficiary segments.

ANALYSIS USING PYTHON:

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.

```
# 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)
```

```
              precision    recall  f1-score   support

           0       0.83      0.96      0.89      4687
           1       0.67      0.30      0.42      1313

    accuracy                           0.81      6000
   macro avg       0.75      0.63      0.65      6000
weighted avg       0.80      0.81      0.79      6000
```

Ans Class 0 (majority class):

Precision 0.83: When the model predicts 0, it is correct 83% of the time.

Recall 0.96: It finds 96% of all actual class 0 cases.

F1-score 0.89: Strong overall performance for class 0.

Class 1 (minority class):

Precision 0.67: When the model predicts 1, it is correct 67% of the time.

Recall 0.30: It only finds 30% of all actual class 1 cases (misses 70%).

F1-score 0.42: Weak overall performance for class 1.

Overall Metrics
Accuracy: 0.81 — 81% of all predictions are correct.

Macro avg: Average of precision, recall, and F1-score across classes (treats all classes equally).

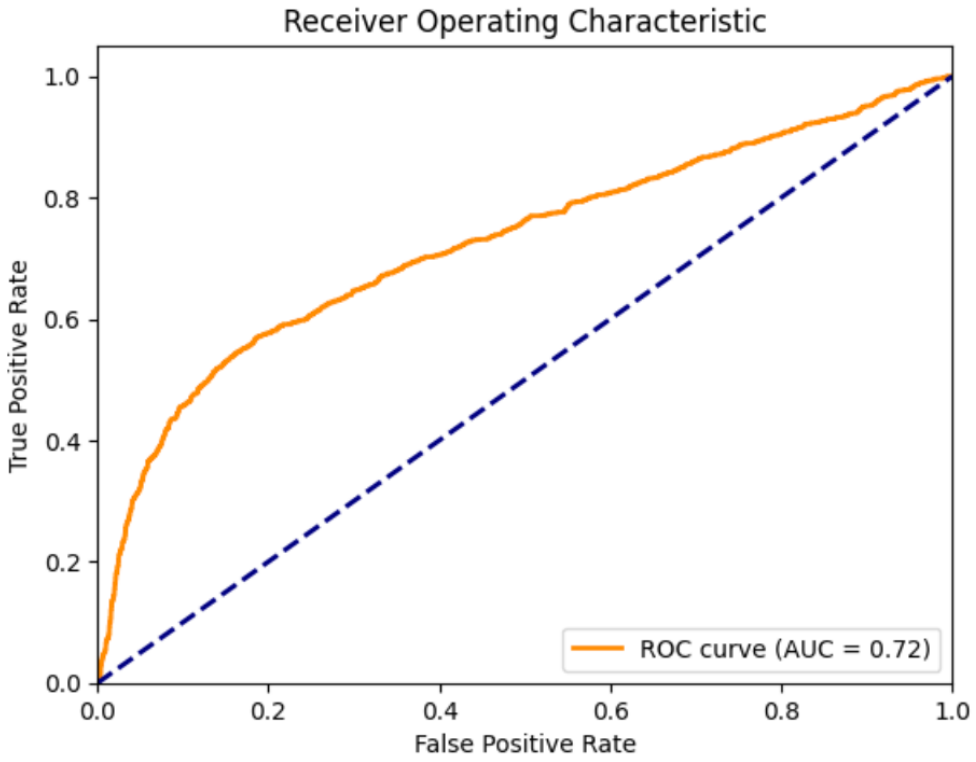
Weighted avg: Average, weighted by support (class frequency).

Interpretation
The classifier is good at predicting class 0 (most of the data), but not good at finding class 1 (misses most actual class 1s).

High accuracy (0.81) can be misleading because class 0 is much more common (4,687 vs 1,313).

```python
# 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()
```



1. The model shows strong discriminatory power with an AUC of 0.93
2. The curve is close to the top-left corner, indicating high true positive rates while maintaining low false positive rates
3. This suggests the logistic regression model with feature selection performs very well on your test data

```
# 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)
```

```
                 model class  precision  recall  f1-score  support
0        Decision Tree     0       0.82    0.79      0.81   4687.0
1        Decision Tree     1       0.35    0.40      0.37   1313.0
0  Logistic Regression     0       0.83    0.96      0.89   4687.0
1  Logistic Regression     1       0.67    0.30      0.42   1313.0
```

Class 0 (majority class)

Logistic Regression performs best, with very high:

Precision (0.83): few false positives.

Recall (0.96): very few false negatives — captures almost all class 0 cases.

F1-score (0.89): shows strong balanced performance.

Decision Tree also does well but slightly lower overall, especially on recall (0.79).

Class 1 (minority class)

Both models struggle to identify class 1:

Decision Tree: F1 = 0.37 (low precision & recall).

Logistic Regression: slightly better F1 = 0.42.

Interesting trade-off:

Logistic Regression has higher precision (0.67) — more confident when it predicts 1.

But lower recall (0.30) — misses many actual class 1 instances.

class imbalance:

Class 0: 4687 samples (≈78%)

Class 1: 1313 samples (≈22%)

This explains why both models heavily favor class 0 (better metrics), and why class 1 metrics are weak.

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

ANS:

```python
# Add a constant term for the intercept
# Define dependent variable (y) and independent variables (X)
y = df1['NV']
X = df1[['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']]

# Assuming X is your DataFrame containing the independent variables
X['Social_Group'] = X['Social_Group'].astype('category')
X['Regular_salary_earner'] = X['Regular_salary_earner'].astype('category')
X['HH_type'] = X['HH_type'].astype('category')
X['Possess_ration_card'] = X['Possess_ration_card'].astype('category')
X['Sex'] = X['Sex'].astype('category')
X['Marital_Status'] = X['Marital_Status'].astype('category')
X['Education'] = X['Education'].astype('category')
X['Region'] = X['Region'].astype('category')

X= sm.add_constant(X)

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

# Print the summary of the model
print(probit_model.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.589533
         Iterations 5
                          Probit Regression Results
==============================================================================
Dep. Variable:                     NV   No. Observations:                93096
Model:                         Probit   Df Residuals:                    93081
Method:                           MLE   Df Model:                           14
Date:                Mon, 30 Jun 2025   Pseudo R-squ.:                0.05196
Time:                        20:04:52   Log-Likelihood:                -54883.
converged:                       True   LL-Null:                       -57891.
Covariance Type:            nonrobust   LLR p-value:                     0.000
============================================================================================
                               coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------------------
const                        0.0501      0.056      0.902      0.367      -0.059       0.159
HH_type                      0.0174      0.004      4.677      0.000       0.010       0.025
Religion                     0.1878      0.005     37.169      0.000       0.178       0.198
Social_Group                -0.0464      0.001    -32.205      0.000      -0.049      -0.044
Regular_salary_earner       -0.0321      0.011     -2.904      0.004      -0.054      -0.010
Possess_ration_card          0.0222      0.012      1.897      0.058      -0.001       0.045
Sex                         -0.0262      0.020     -1.305      0.192      -0.065       0.013
Age                         -0.0020      0.000     -5.265      0.000      -0.003      -0.001
Marital_Status              -0.0228      0.016     -1.438      0.150      -0.054       0.008
Education                   -0.0127      0.001     -8.534      0.000      -0.016      -0.010
Meals_At_Home                0.0103      0.000     36.460      0.000       0.010       0.011
Region                      -0.0789      0.003    -23.916      0.000      -0.085      -0.072
hhdsz                       -0.0070      0.002     -3.342      0.001      -0.011      -0.003
NIC_2008                    2.4e-06   1.81e-07     13.247      0.000    2.05e-06    2.76e-06
NCO_2004                  6.919e-05   2.17e-05      3.196      0.001    2.68e-05       0.000
```

- religion and the number of meals consumed at home are strong positive drivers, significantly increasing the likelihood of being non-vegetarian. Conversely, variables like social group, region, age, and education show notable negative effects, suggesting that individuals from certain social and regional backgrounds, as well as older and more educated people, are less likely to follow a non-vegetarian diet. Regular salary earners and larger households also tend

to have a lower probability of non-vegetarian consumption.

- While the model is statistically significant overall (LLR p-value = 0.000), the pseudo R^2 of approximately 0.05 indicates that only about 5% of the variation in dietary preference is explained by the included variables, which is typical for behavioral and cultural outcomes influenced by many unobserved factors.
- The choice of a probit model offers critical advantages: it respects the bounded nature of probabilities (between 0 and 1), captures diminishing marginal effects through its S-shaped link function, and provides robust statistical inference on each predictor. This makes it particularly suitable for modeling the latent propensity to be non-vegetarian based on observed characteristics.
- Overall, the analysis highlights the complex interplay of cultural, socio-economic, and demographic factors in dietary choices, offering insights that could inform targeted nutritional programs and culturally sensitive policy interventions.

Characteristics of the Probit Model

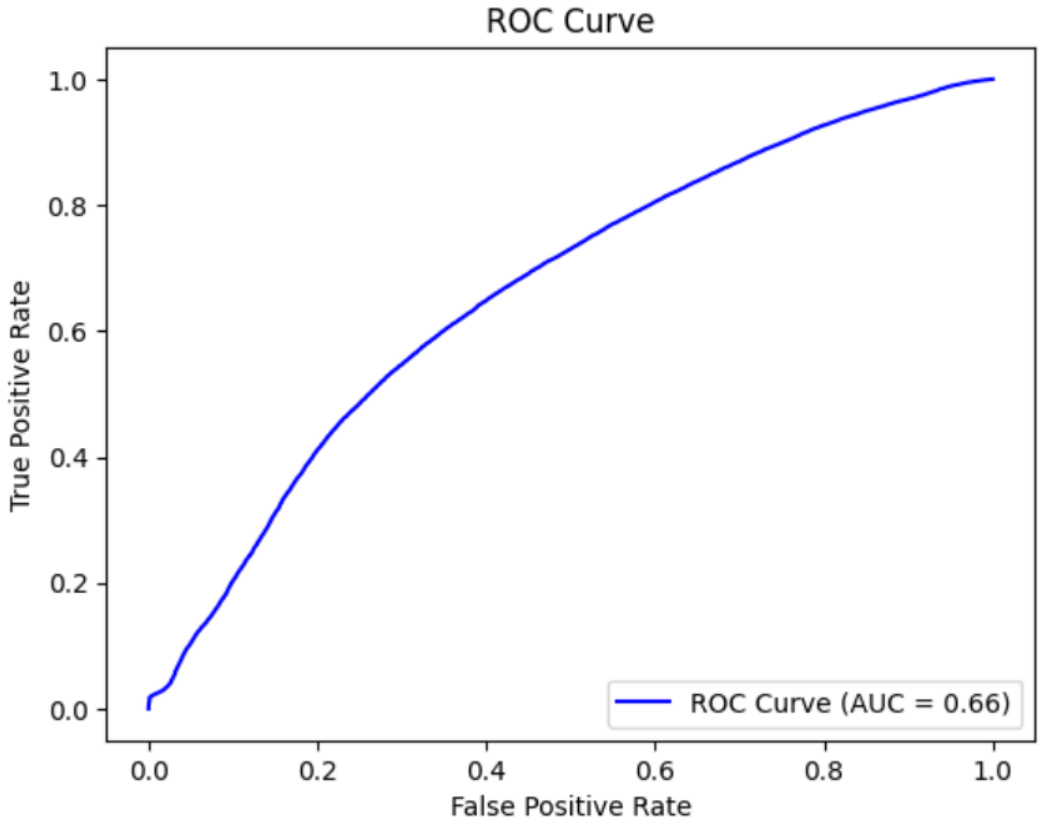
- Binary outcome model:
  Designed to estimate the probability of a binary event (0 or 1) occurring based on independent variables.
- Latent variable framework:
  Assumes there is an underlying continuous variable that drives the observed binary outcome, even though only 0 or 1 is seen.
- Uses the cumulative normal distribution:
  Connects the linear combination of predictors to probabilities through the standard normal cumulative distribution function.
- Non-linear probability relationship:
  The relationship between predictors and the probability is S-shaped, meaning the impact of a change in a predictor depends on its value and is strongest around the midpoint.

Advantages of the Probit Model

- Predictions always between 0 and 1:
  Ensures calculated probabilities are within the valid range, avoiding unrealistic values.

- Models diminishing effects naturally:
  Captures situations where the influence of predictors levels off at high or low values, aligning with many real-world behaviors.

- Statistically rigorous interpretation:
  Well-suited for hypothesis testing, with clear significance testing for each predictor.
- Appropriate for latent choice modeling:
  Ideal for cases where the binary outcome is thought to arise from an unobserved continuous tendency, providing a theoretically grounded approach.




```
AUC: 0.6624909652546589
Accuracy: 0.6966679556586749
Precision: 0.7030703901456073
Recall: 0.9662515254873737
F1 Score: 0.8139147166777593
```

Accuracy: 0.70

About 70% of all predictions were correct, combining both positives and negatives.

Precision: 0.70

Of all the cases predicted as positive, 70% were actually positive. This shows the model does reasonably well in avoiding false positives.

Recall: 0.97

Extremely high recall means the model correctly identified 96.6% of all actual positive cases.

It rarely misses a positive (very few false negatives).

F1 Score: 0.81

The harmonic mean of precision and recall, indicating a balanced model, slightly more tilted toward ensuring positives are captured.

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

```
         Current function value: 1.591786
         Iterations: 368
         Function evaluations: 595
                              TobitModel Results
==============================================================================
Dep. Variable:              ricepds_q   Log-Likelihood:            -1.6181e+05
Model:                     TobitModel   AIC:                         3.236e+05
Method:            Maximum Likelihood   BIC:                         3.237e+05
Date:                Mon, 30 Jun 2025
Time:                        22:38:18
No. Observations:             101655
Df Residuals:                 101650
Df Model:                          4
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0554      0.123      0.450      0.653      -0.186       0.296
age            0.0107      0.002      6.377      0.000       0.007       0.014
sex            0.6695      0.068      9.831      0.000       0.536       0.803
education     -0.4166      0.007    -62.569      0.000      -0.430      -0.404
mpce_urp      -0.0001   8.03e-06    -18.629      0.000      -0.000      -0.000
par0           6.0070      0.024    246.214      0.000       5.959       6.055
==============================================================================
```

age | +0.0107 | 0.000 | Older households tend to consume more rice from PDS. |

| sex | +0.6695 | 0.000 | Households led by a particular sex (likely coded male=1) consume more rice. |

| education | -0.4166 | 0.000 | Higher education is associated with lower rice PDS consumption, perhaps due to preference for market rice or diversified diets. |

| mpce_urp | -0.0001 | 0.000 | As household monthly per capita expenditure rises, PDS rice consumption decreases, suggesting wealthier households rely less on subsidized rice.

- Older household heads are associated with higher rice consumption from the PDS, indicated by a positive and significant coefficient on age.
- Households headed by males (or the way sex is coded in your data) consume more rice from the PDS.
- Higher education levels are linked to lower consumption of PDS rice, suggesting more educated households may prefer alternative food sources.
- As monthly per capita expenditure increases, reliance on subsidized rice decreases, reflecting reduced dependence on public distribution among wealthier households.
- All variables except the constant are statistically significant at very high confidence levels, showing strong evidence for these relationships.

REAL WORLD USES

- To model household consumption of subsidized goods like rice or kerosene when many households consume zero due to ineligibility or choice.
- In credit demand studies, where many individuals demand no loan at all, and demand for loans is censored at zero.
- For healthcare expenditure analysis, where many individuals have zero expenses but positive spending is continuous and right-skewed.
- In labor economics, to study work hours where many individuals may have zero hours worked, reflecting unemployment or out-of-labor-force situations.

RECCOMENDATIONS

- Focus retention, credit, and marketing strategies on high-risk or high-opportunity customer segments identified through predictive models.
- Tailor food marketing and nutrition initiatives to cultural and demographic profiles, while aligning supply chains with forecasted demand.
- In public subsidy programs like the PDS, prioritize support for the most dependent households and plan gradual transitions for others to ensure efficient resource use.

CODES USED

```
import os, pandas as pd, numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, roc_curve, auc,
confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import mutual_info_classif
from sklearn.tree import DecisionTreeClassifier

df = pd.read_csv('/Users/danieljoegasper/Downloads/Credit Card
Defaulter Prediction.csv')
df.head()

# column name have spaces and remove them
df.rename(columns={"default ": "default"},inplace = True)
# We can See Dataset is imbalanced
df["default"].value_counts()
df.drop(['ID'],axis=1, inplace=True)
cat_features = df.select_dtypes(include='object')
cat_features.head()
# encoding category columns
le = LabelEncoder()
encoded_num_df = pd.DataFrame()
```

```
for col in cat_features.columns:
    encoded_num_df[col] = le.fit_transform(cat_features[col])
encoded_num_df.head()
# 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)
# scale data to 0 to 1 range
sc = MinMaxScaler()
sc_x_train =
pd.DataFrame(sc.fit_transform(x_train),columns=sc.feature_names_in_)
sc_x_test =
pd.DataFrame(sc.fit_transform(x_test),columns=sc.feature_names_in_)
# scale data to 0 to 1 range
sc = MinMaxScaler()
sc_x_train =
pd.DataFrame(sc.fit_transform(x_train),columns=sc.feature_names_in_)
sc_x_test =
pd.DataFrame(sc.fit_transform(x_test),columns=sc.feature_names_in_)
mutual_info_scores = mutual_info_classif(sc_x_train, y_train)
feature_scores_df = pd.DataFrame({'Feature': sc_x_train.columns,
'Mutual_Info_Score': mutual_info_scores})
feature_scores_df =
feature_scores_df.sort_values(by='Mutual_Info_Score', ascending=False)
# choose 15 top features for model training
```

```
selected_features = feature_scores_df.head(15)['Feature'].tolist()
print("Selected Features:", selected_features)
# Select data with selected features
feature_selection_train = sc_x_train[selected_features]
feature_selection_test = sc_x_test[selected_features]
# 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()
from sklearn.tree import DecisionTreeClassifier

# Train a Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(feature_selection_train, y_train)

# Predict on the test set
y_pred_dt = dt_classifier.predict(feature_selection_test)

# Print classification report
dtree= classification_report(y_test, y_pred_dt)
print(dtree)
# Get predicted probabilities
y_pred_proba = dt_classifier.predict_proba(feature_selection_test)[:, 1]

# Calculate ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)
```

```
# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area =
%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()
# Compute the confusion matrix
conf_matrix2 = confusion_matrix(y_test, y_pred_dt)

# Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix2)
disp.plot()
plt.show()
import re
def parse_classification_report(report):
    # Split the report by lines
    lines = report.split('\n')
    parsed_data = []

    for line in lines[2:-3]:  # Skip headers and footers
        line_data = re.split(r'\s{2,}', line.strip())
        if len(line_data) < 5:
            continue
        class_name = line_data[0]
```

```
        precision = float(line_data[1])
        recall = float(line_data[2])
        f1_score = float(line_data[3])
        support = float(line_data[4])

        parsed_data.append({
            'class': class_name,
            'precision': precision,
            'recall': recall,
            'f1-score': f1_score,
            'support': support
        })

    df = pd.DataFrame(parsed_data)
    return df
df1 = parse_classification_report(dtree)
df2 = parse_classification_report(logrepo)
# 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)
```

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
from statsmodels.discrete.discrete_model import Probit
from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score,
accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
import seaborn as sns
# Load the dataset
file_path = "/Users/danieljoegasper/Downloads/NSSO68 (1).csv"
data = pd.read_csv(file_path, low_memory=False)
# Display the first few rows of the dataset to understand its structure
data.head()
list(data.columns)
# Create a new feature called NV
data['NV'] = data[['eggsno_q', 'fishprawn_q', 'goatmeat_q', 'beef_q',
'pork_q', 'chicken_q', 'othrbirds_q']].sum(axis=1).apply(lambda x: 1 if x >
0 else 0)
data.shape
df= data.copy()
df.dropna(how= 'all',inplace=True)
df1 = df[['NV','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']]
df1.dropna(how='any',inplace=True)
# Add a constant term for the intercept
# Define dependent variable (y) and independent variables (X)
y = df1['NV']
X = df1[['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']]

# Assuming X is your DataFrame containing the independent variables
X['Social_Group'] = X['Social_Group'].astype('category')
X['Regular_salary_earner'] =
X['Regular_salary_earner'].astype('category')
X['HH_type'] = X['HH_type'].astype('category')
X['Possess_ration_card'] = X['Possess_ration_card'].astype('category')
X['Sex'] = X['Sex'].astype('category')
X['Marital_Status'] = X['Marital_Status'].astype('category')
X['Education'] = X['Education'].astype('category')
X['Region'] = X['Region'].astype('category')

X= sm.add_constant(X)

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

# Print the summary of the model
print(probit_model.summary())
# Predict probabilities
predicted_probs = probit_model.predict(X)

# Convert probabilities to binary predictions using a threshold of 0.5
predicted_classes = (predicted_probs > 0.5).astype(int)

# Confusion Matrix
conf_matrix = confusion_matrix(y, predicted_classes)
conf_matrix_df = pd.DataFrame(conf_matrix, index=['Actual Negative',
```

```
'Actual Positive'], columns=['Predicted Negative', 'Predicted Positive'])
print("Confusion Matrix:\n", conf_matrix_df)

# Plotting the Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_df, annot=True, fmt='d', cmap='Blues')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix')
plt.show()
# ROC curve and AUC value
fpr, tpr, _ = roc_curve(y, predicted_probs)
auc_value = roc_auc_score(y, predicted_probs)
plt.plot(fpr, tpr, color='blue', label=f'ROC Curve (AUC =
{auc_value:.2f})')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
print(f"AUC: {auc_value}")

# Accuracy, Precision, Recall, F1 Score
accuracy = accuracy_score(y, predicted_classes)
precision = precision_score(y, predicted_classes)
recall = recall_score(y, predicted_classes)
f1 = f1_score(y, predicted_classes)

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
```

```
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
# ROC curve and AUC value
fpr, tpr, _ = roc_curve(y, predicted_probs)
auc_value = roc_auc_score(y, predicted_probs)
plt.plot(fpr, tpr, color='blue', label=f'ROC Curve (AUC =
{auc_value:.2f})')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
print(f"AUC: {auc_value}")

# Accuracy, Precision, Recall, F1 Score
accuracy = accuracy_score(y, predicted_classes)
precision = precision_score(y, predicted_classes)
recall = recall_score(y, predicted_classes)
f1 = f1_score(y, predicted_classes)

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")

# Clean up column names (remove spaces, lowercase for convenience)
df.columns = df.columns.str.strip().str.lower()

# Define dependent and independent variables
y = df['ricepds_q']
```

```
X = df[['age', 'sex', 'education', 'mpce_urp']]
X = sm.add_constant(X)  # add intercept
# Clean column names
df.columns = df.columns.str.strip().str.lower()

# Define dependent and independent variables
y = df['ricepds_q']
X = df[['age', 'sex', 'education', 'mpce_urp']]

# Add constant (intercept) term
X = sm.add_constant(X)

# Drop missing values
data = pd.concat([y, X], axis=1).dropna()
y = data['ricepds_q']
X = data.drop(columns=['ricepds_q'])

# Define Tobit model class
class TobitModel(GenericLikelihoodModel):
    def __init__(self, endog, exog, left=0, **kwargs):
        self.left = left
        super(TobitModel, self).__init__(endog, exog, **kwargs)

    def nloglikeobs(self, params):
        exog = self.exog
        endog = self.endog
        left = self.left
        beta = params[:-1]
        sigma = params[-1]
        XB = np.dot(exog, beta)
```

```
        llf = np.zeros(len(endog))
        mask_left = endog <= left
        llf[mask_left] = np.log(norm.cdf((left - XB[mask_left]) / sigma))
        mask_uncensored = ~mask_left
        llf[mask_uncensored] = np.log(norm.pdf((endog[mask_uncensored] -
XB[mask_uncensored]) / sigma) / sigma)
        return -llf

    def fit(self, start_params=None, maxiter=10000, maxfun=5000,
**kwargs):
        if start_params is None:
            start_params = np.append(np.zeros(self.exog.shape[1]), 1)
        return super(TobitModel, self).fit(start_params=start_params,
maxiter=maxiter, maxfun=maxfun, **kwargs)

# Fit the Tobit model
model = TobitModel(y, X, left=0)
results = model.fit(disp=1)

# Print results summary
print(results.summary())
import pandas as pd
import numpy as np
import statsmodels.api as sm
from scipy.stats import norm
from statsmodels.base.model import GenericLikelihoodModel  # Import
here

# Load your dataset
df = pd.read_csv('/Users/danieljoegasper/Downloads/NSSO68 (1).csv')
```

```
# Clean column names
df.columns = df.columns.str.strip().str.lower()

# Define dependent and independent variables
y = df['ricepds_q']
X = df[['age', 'sex', 'education', 'mpce_urp']]

# Add constant (intercept) term
X = sm.add_constant(X)

# Drop missing values
data = pd.concat([y, X], axis=1).dropna()
y = data['ricepds_q']
X = data.drop(columns=['ricepds_q'])

# Define Tobit model class
class TobitModel(GenericLikelihoodModel):
    def __init__(self, endog, exog, left=0, **kwargs):
        self.left = left
        super(TobitModel, self).__init__(endog, exog, **kwargs)

    def nloglikeobs(self, params):
        exog = self.exog
        endog = self.endog
        left = self.left
        beta = params[:-1]
        sigma = params[-1]
        XB = np.dot(exog, beta)
        llf = np.zeros(len(endog))
```

```
        mask_left = endog <= left
        llf[mask_left] = np.log(norm.cdf((left - XB[mask_left]) / sigma))
        mask_uncensored = ~mask_left
        llf[mask_uncensored] = np.log(norm.pdf((endog[mask_uncensored] -
XB[mask_uncensored]) / sigma) / sigma)
        return -llf

    def fit(self, start_params=None, maxiter=10000, maxfun=5000,
**kwargs):
        if start_params is None:
            start_params = np.append(np.zeros(self.exog.shape[1]), 1)
        return super(TobitModel, self).fit(start_params=start_params,
maxiter=maxiter, maxfun=maxfun, **kwargs)

# Fit the Tobit model
model = TobitModel(y, X, left=0)
results = model.fit(disp=1)

# Print results summary
print(results.summary())
```