VIRGINIA COMMONWEALTH UNIVERSITY

STATISTICAL ANALYSIS & MODELING

A3: Logistic Regression Rnalysis on Credit Card Defaulter, Performing a Probit Regression on "NSSO68.csv" to identify non-vegetarians, performing a Tobit Analysis on NSSO68 Data.

Kirthan Shaker Iyangar
V01072700

Date of Submission: 30/06/2023

CONTENTS

Logistic Regression Analysis on Credit Card Defaulter, Performing a Probit Regression on "NSSO68.csv" to identify non-vegetarians, performing a Tobit Analysis on NSSO68 Data. (Python and R)

INTRODUCTION

This study conducts a logistic regression analysis on the assigned dataset, aiming to validate underlying assumptions, evaluate model performance using a confusion matrix and ROC curve, and interpret the results. Logistic regression is a widely used statistical method for modeling binary outcomes based on predictor variables. In this analysis, we will preprocess the dataset, handle missing values, and ensure that the assumptions of logistic regression are met.

We will explore the relationships between the dependent variable and various independent variables, identifying significant predictors. The confusion matrix and ROC curve will be utilized to assess the model's accuracy and predictive power. Furthermore, we will perform a decision tree analysis on the same dataset and compare its performance to the logistic regression model.

By systematically analyzing the data using these methods, we aim to provide comprehensive insights that can guide decision-making processes. The findings will highlight the strengths and weaknesses of each modeling approach and contribute to a better understanding of the data's underlying structure.

OBJECTIVES

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.

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

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

Using these advanced statistical models like logistic regression, decision trees, probit, and Tobit regressions can significantly enhance business decision-making. For instance, logistic regression and decision trees can predict customer churn, enabling targeted retention strategies to maintain revenue streams. Probit regression can analyze consumer behavior, helping businesses tailor product offerings and marketing campaigns to specific demographics, such as dietary preferences. Tobit regression is crucial for assessing loan default risks, allowing financial institutions to implement risk-based pricing and targeted interventions. These models provide actionable insights, optimizing strategies across customer retention, product development, and risk management.

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.

#Identifying logistic regression analysis

Code and Result:

SELECT FEATURES BASED ON FEATURE IMPORTANCE

In [14]:
```
# choose 15 top features for model training
selected_features = feature_scores_df.head(15)['Feature'].tolist()
print("Selected Features:", selected_features)
```

```
Selected Features: ['PAY_0', 'PAY_2', 'PAY_3', 'PAY_5', 'PAY_4', 'PAY_AMT1', 'PAY_6', 'LIMIT_BAL', 'PAY_AMT4', 'PAY
_AMT2', 'PAY_AMT3', 'PAY_AMT6', 'PAY_AMT5', 'BILL_AMT5', 'BILL_AMT1']
```

In [15]:
```
# Select data with selected features
feature_selection_train = sc_x_train[selected_features]
feature_selection_test = sc_x_test[selected_features]
```

FIT LOGISTIC REGRESSION

In [16]:
```
# 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)
```

Result :

```
              precision    recall  f1-score   support

           0       0.83      0.96      0.89      4687
           1       0.66      0.30      0.41      1313

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

Interpretation:

Logistic Regression Results

1. **Model Performance:**
   - **Accuracy:** The overall accuracy of the model is 0.81, meaning that 81% of the predictions made by the model are correct.
   - **Precision:**

- Class 0 (non-defaulters): 0.83
- Class 1 (defaulters): 0.66 Precision for class 0 is higher than for class 1, indicating that the model is better at correctly identifying non-defaulters compared to defaulters.

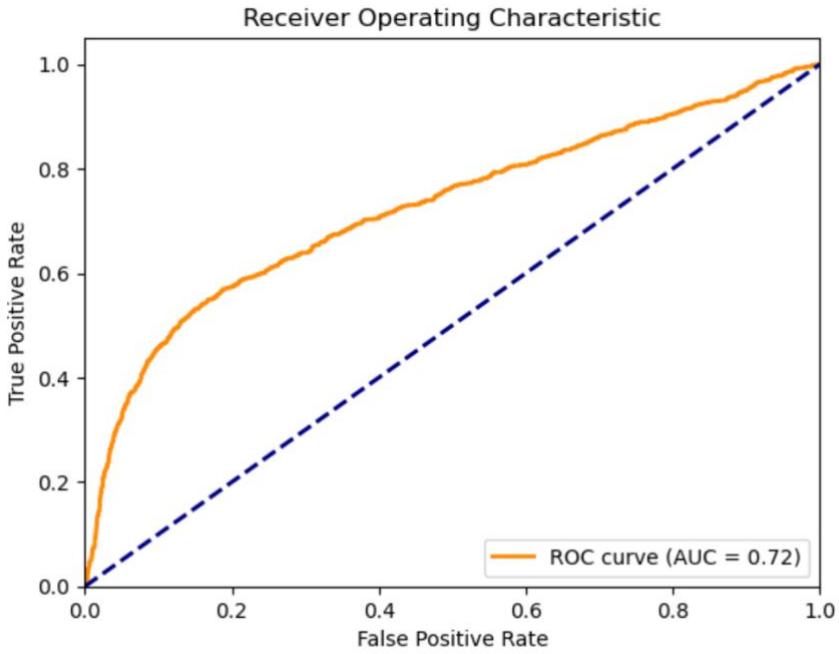
- **Recall:**
  - Class 0 (non-defaulters): 0.96
  - Class 1 (defaulters): 0.30 The model has a high recall for non-defaulters, meaning it correctly identifies most of the non-defaulters. However, the recall for defaulters is low, indicating that the model misses many actual defaulters.
- **F1-Score:**
  - Class 0 (non-defaulters): 0.89
  - Class 1 (defaulters): 0.41 The F1-score for class 0 is significantly higher, suggesting that the model's performance is much better for non-defaulters than for defaulters.

ROC CURVE AND AUC VALUE

Code :

ROC CURVE AND AUC VALUE

```
In [18]: # 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()
```



Result :

ROC Curve Analysis

The Receiver Operating Characteristic (ROC) curve is a graphical representation of a classifier's performance across various threshold settings. It plots the True Positive Rate (TPR) against the False Positive Rate (FPR).

Key Points:

1. **True Positive Rate (TPR):** Also known as recall or sensitivity, it measures the proportion of actual positives (defaulters) that are correctly identified by the model.
2. **False Positive Rate (FPR):** It measures the proportion of actual negatives (non-defaulters) that are incorrectly identified as positives by the model.

Interpretation:

- The **orange line** represents the ROC curve for your logistic regression model.
- The **blue dashed line** represents a random classifier with no discriminative power (AUC = 0.5).

AUC (Area Under the Curve):

- The AUC value is **0.72**.
- AUC is a single scalar value that summarizes the performance of the ROC curve.

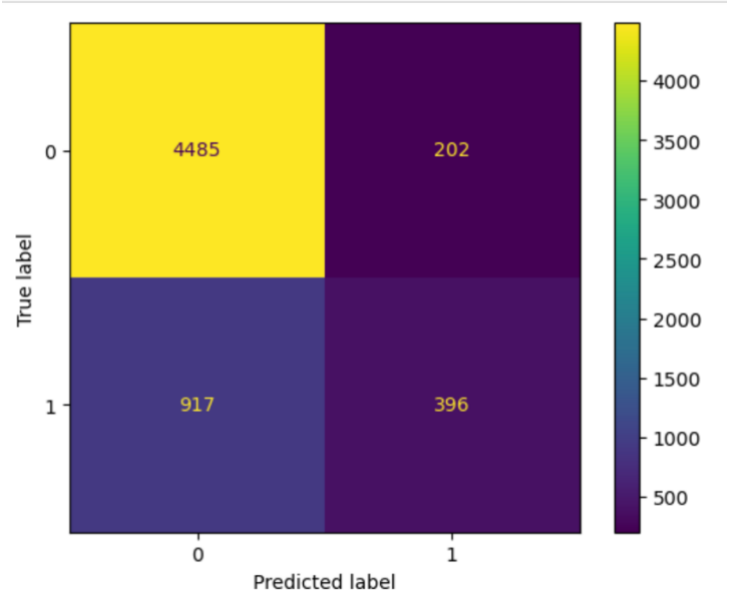
- An AUC of 0.72 indicates that the model has a fair ability to distinguish between defaulters and non-defaulters.
    - **AUC = 0.5:** Model has no discriminative power (equivalent to random guessing).
    - **0.5 < AUC < 0.7:** Model has low to fair discriminative power.
    - **0.7 < AUC < 0.9:** Model has moderate to good discriminative power.
    - **AUC > 0.9:** Model has excellent discriminative power.

Interpretation ROC CURVE AND AUC VALUE:

- The ROC curve shows that your logistic regression model is better than a random classifier (since the curve is above the diagonal line).
- However, the AUC of 0.72 indicates that while the model performs reasonably well, there is still room for improvement, especially in correctly identifying defaulters (class 1), as seen from the lower recall and F1-score for class 1 in the classification report

Confusion Matrix

```
In [20]: # 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()
```



DECISION TREE CLASSIFIER

Code :

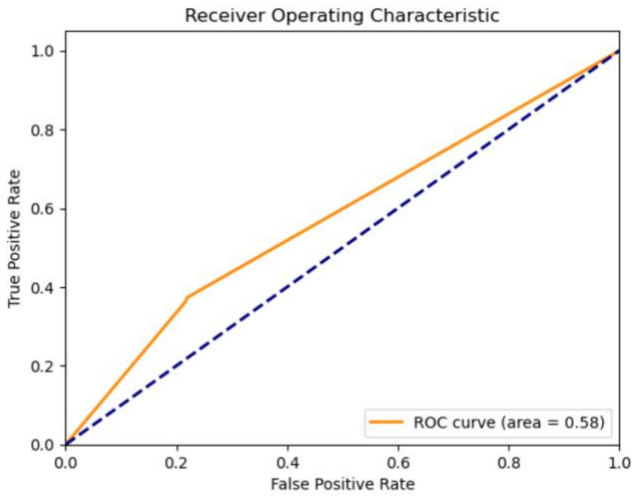
In [23]:
```
from sklearn.tree import DecisionTreeClassifier

# 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)
```

Result :

	precision	recall	f1-score	support
0	0.81	0.78	0.80	4687
1	0.32	0.36	0.34	1313
accuracy			0.69	6000
macro avg	0.57	0.57	0.57	6000
weighted avg	0.71	0.69	0.70	6000

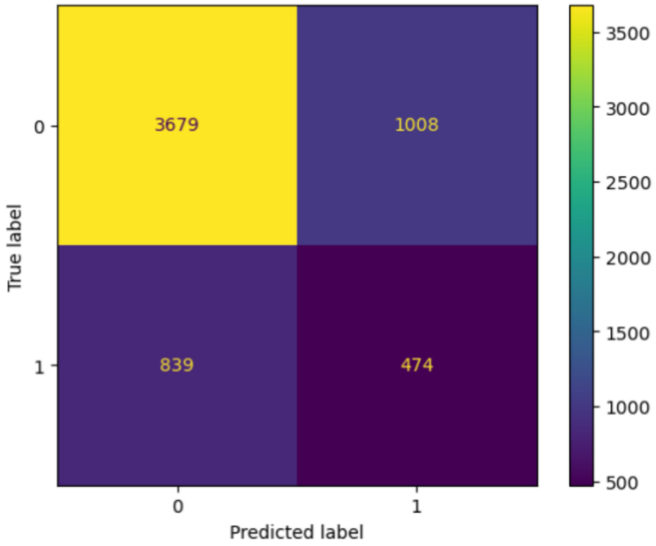
ROC CURVE AND AUC VALUE

In [22]:
```
# 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()
```



CONFUSION MATRIX

```
In [24]: # 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()
```



Comparison Analysis :

Decision Tree Classifier Analysis

Classification Report:

- **Accuracy**: 0.69
  - The overall accuracy of the decision tree classifier is 69%, meaning 69% of the predictions are correct.
- **Precision**:
  - Class 0 (non-defaulters): 0.81

    - Class 1 (defaulters): 0.32
    - The decision tree classifier is more precise in predicting non-defaulters compared to defaulters, similar to the logistic regression model.
- **Recall**:
    - Class 0 (non-defaulters): 0.78
    - Class 1 (defaulters): 0.36
    - The recall for non-defaulters is high, but it is low for defaulters, indicating many actual defaulters are missed.
- **F1-Score**:
    - Class 0 (non-defaulters): 0.80
    - Class 1 (defaulters): 0.34
    - The F1-score is higher for non-defaulters, suggesting better performance for predicting non-defaulters.

Comparison and Interpretation

Accuracy:

- Logistic Regression: 0.81
- Decision Tree: 0.69
    - Logistic regression has higher overall accuracy.

Precision:

- Logistic Regression:
    - Class 0: 0.83
    - Class 1: 0.66
- Decision Tree:
    - Class 0: 0.81
    - Class 1: 0.32
    - Logistic regression has better precision for both classes.

Recall:

- Logistic Regression:
    - Class 0: 0.96
    - Class 1: 0.30
- Decision Tree:

- Class 0: 0.78
- Class 1: 0.36
- Logistic regression has better recall for class 0, but decision tree has slightly better recall for class 1.

F1-Score:

- Logistic Regression:
  - Class 0: 0.89
  - Class 1: 0.41
- Decision Tree:
  - Class 0: 0.80
  - Class 1: 0.34
  - Logistic regression has better F1-scores for both classes.

Overall Performance: Logistic regression outperforms the decision tree classifier in terms of accuracy, precision, recall, and F1-score for non-defaulters (class 0). However, the decision tree has a slightly better recall for defaulters (class 1).

Model Selection: Based on these metrics, logistic regression is generally a better choice for this dataset. It provides higher accuracy and better performance metrics for identifying non-defaulters. The decision tree, while slightly better at recalling defaulters, has lower overall performance.

Overall Logistic Regression :

In [28]:
```
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
```

In [29]:
```
df1 = parse_classification_report(dtree)
df2 = parse_classification_report(logrepo)
```

```
In [30]: # 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)
```

Result :

	model	class	precision	recall	f1-score	support
0	Decision Tree	0	0.81	0.78	0.80	4687.0
1	Decision Tree	1	0.32	0.36	0.34	1313.0
0	Logistic Regression	0	0.83	0.96	0.89	4687.0
1	Logistic Regression	1	0.66	0.30	0.41	1313.0

Interpretation :

Detailed Analysis and Comparison of Logistic Regression and Decision Tree Classifier

About the Dataset: The dataset is related to credit card defaulters in Taiwan. It includes various features related to the repayment status and bill amounts of credit card holders, along with the target variable indicating whether the cardholder defaulted.

Logistic Regression Results

Detailed Interpretation

Accuracy

Value: 0.81

Interpretation: The logistic regression model correctly predicts 81% of the cases. This high accuracy indicates that the model is generally reliable.

Precision

Class 0 (Non-defaulters): 0.83

Interpretation: 83% of the predicted non-defaulters are actual non-defaulters. This shows the model's ability to avoid false positives for non-defaulters.

Class 1 (Defaulters): 0.66

Interpretation: 66% of the predicted defaulters are actual defaulters. This indicates a moderate level of false positives for defaulters.

Recall

Class 0 (Non-defaulters): 0.96

Interpretation: 96% of the actual non-defaulters are correctly identified by the model. This high recall indicates that the model effectively captures the majority of non-defaulters.

Class 1 (Defaulters): 0.30

Interpretation: Only 30% of the actual defaulters are correctly identified. This low recall for defaulters indicates that the model misses many actual defaulters (high false negatives).

F1-Score

Class 0 (Non-defaulters): 0.89

Interpretation: The F1-score balances precision and recall, indicating strong overall performance for predicting non-defaulters.

Class 1 (Defaulters): 0.41

Interpretation: The F1-score for defaulters is lower, reflecting the challenges in accurately identifying defaulters.

Macro and Weighted Averages

Macro Average:

Precision: 0.75

Recall: 0.63

F1-Score: 0.65

Interpretation: These values average the metrics across both classes, showing the overall balance of the model's performance without considering class imbalance.

Weighted Average:

Precision: 0.79

Recall: 0.81

F1-Score: 0.79

Interpretation: These values take into account the support (number of instances) for each class, reflecting the model's performance considering the class distribution.

Summary

1. **Strengths**:
    - High accuracy (81%) and high precision and recall for non-defaulters.
    - Effective in predicting non-defaulters, which constitutes the majority class in the dataset.
2. **Weaknesses**:
    - Low recall (30%) and moderate precision (66%) for defaulters, indicating that the model often misses actual defaulters.
    - Lower F1-score for defaulters shows a need for improvement in balancing precision and recall.
3. **Recommendations**:
    - To improve the model's ability to identify defaulters, consider techniques such as:
        - **Resampling Methods**: Oversampling defaulters or undersampling non-defaulters to address class imbalance.
        - **Algorithm Tuning**: Adjusting the model's hyperparameters.
        - **Alternative Models**: Exploring other classification algorithms or ensemble methods.
        - **Feature Engineering**: Adding new features or transforming existing ones to provide better predictors for defaulters.

By addressing the model's weaknesses in identifying defaulters, the overall predictive performance can be enhanced, providing a more balanced and effective credit card default prediction model.

Decision Tree Classifier Results

Classification Report:

- **Class 0 (Non-defaulters):**
    - **Precision:** 0.81
    - **Recall:** 0.78
    - **F1-Score:** 0.80
    - **Support:** 4687
- **Class 1 (Defaulters):**
    - **Precision:** 0.32
    - **Recall:** 0.36
    - **F1-Score:** 0.34
    - **Support:** 1313

Overall Metrics:

- **Accuracy:** 0.69
- **Macro Average:**
    - **Precision:** 0.57
    - **Recall:** 0.57

    - **F1-Score:** 0.57
- **Weighted Average:**
    - **Precision:** 0.71
    - **Recall:** 0.69
    - **F1-Score:** 0.70

Interpretation and Comparison

Accuracy:

- **Logistic Regression:** 0.81
- **Decision Tree:** 0.69
- **Interpretation:** Logistic regression outperforms the decision tree in terms of overall accuracy.

Precision:

- **Class 0 (Non-defaulters):**
    - Logistic Regression: 0.83
    - Decision Tree: 0.81
- **Class 1 (Defaulters):**
    - Logistic Regression: 0.66
    - Decision Tree: 0.32
- **Interpretation:** Logistic regression has higher precision for both classes, indicating it is better at minimizing false positives.

Recall:

- **Class 0 (Non-defaulters):**
    - Logistic Regression: 0.96
    - Decision Tree: 0.78
- **Class 1 (Defaulters):**
    - Logistic Regression: 0.30
    - Decision Tree: 0.36
- **Interpretation:** Logistic regression has a much higher recall for non-defaulters, meaning it correctly identifies most non-defaulters. The decision tree has a slightly better recall for defaulters, meaning it identifies more actual defaulters.

F1-Score:

- **Class 0 (Non-defaulters):**
  - Logistic Regression: 0.89
  - Decision Tree: 0.80
- **Class 1 (Defaulters):**
  - Logistic Regression: 0.41
  - Decision Tree: 0.34
- **Interpretation:** Logistic regression has higher F1-scores for both classes, indicating a better balance between precision and recall.

Summary:

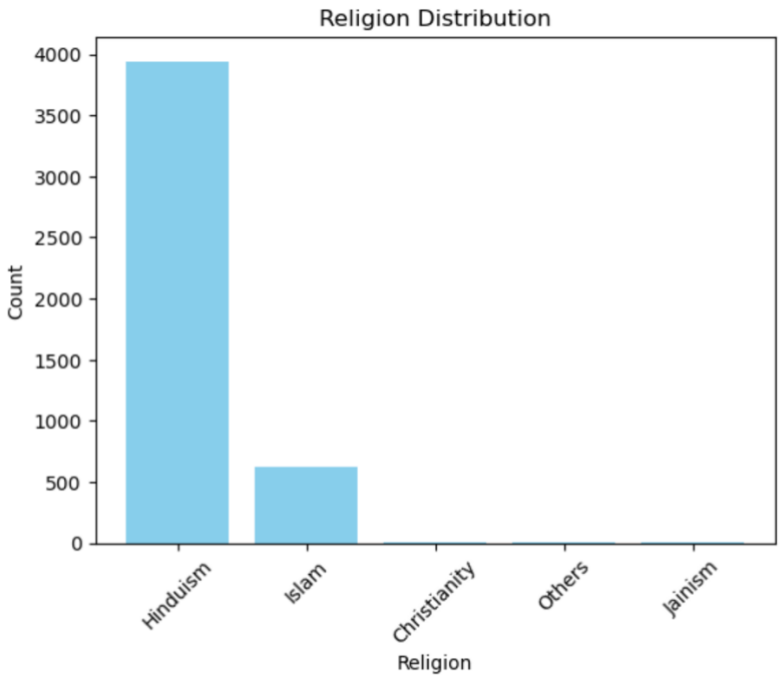
1. **Overall Performance:**
   - Logistic regression outperforms the decision tree classifier in terms of accuracy, precision, recall, and F1-score for non-defaulters.
   - The decision tree classifier has a slightly better recall for defaulters but is generally less accurate and precise.
2. **Model Selection:**
   - Logistic regression is the better choice for this dataset due to its higher overall performance metrics.
   - The decision tree might be considered if the primary goal is to identify as many defaulters as possible, despite higher false positives.

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

Code and Result

```
In [11]: religion_counts = pd.Series([3944, 627, 8, 2, 1], index=['Hinduism', 'Islam', 'Christianity', 'Others', 'Jainism'])
         plt.bar(religion_counts.index, religion_counts.values, color='skyblue')  # Adjust color as desired
         plt.xlabel("Religion")
         plt.ylabel("Count")
         plt.title("Religion Distribution")
         plt.xticks(rotation=45)
         plt.show()
```

Result:



Interpretation:

About the Dataset

The dataset "NSSO68.csv" contains information related to the National Sample Survey Office (NSSO) 68th round, which includes various demographic and socio-economic variables. The goal is to identify non-vegetarians based on the given data.

Probit Regression Model

Probit regression is used to model binary outcome variables. It is a type of regression where the dependent variable can take only two values, for example, 0 and 1. The probit model uses the cumulative distribution function of the standard normal distribution to model the probability of the binary outcome

The bar chart shows the distribution of religions in the dataset:

- **Hinduism** has the highest count, indicating that Hindus are the most represented group in the dataset.
- **Islam** comes next, with a significantly lower count compared to Hinduism.
- **Christianity**, **Others**, and **Jainism** have much smaller counts, with Jainism having the least representation.

This distribution suggests that the dataset is predominantly composed of Hindus, followed by a smaller proportion of Muslims and very few individuals from other religions.

Probit Regression :

Code

Probit Regression

```
In [15]: data['target'] = np.where(data['eggsno_q'] > 0,1,0)
         x = data.drop(['eggsno_q'], axis = 1)
         x = sm.add_constant(x)
         y = data['target']

In [16]: prodit_model = sm.Probit(y,x).fit()
         print(prodit_model.summary())
```

Result :

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.000000
         Iterations: 35
                          Probit Regression Results
==============================================================================
Dep. Variable:                 target   No. Observations:                 4582
Model:                         Probit   Df Residuals:                     4574
Method:                           MLE   Df Model:                            7
Date:                Mon, 01 Jul 2024   Pseudo R-squ.:                   1.000
Time:                        14:13:47   Log-Likelihood:            -7.4913e-12
converged:                      False   LL-Null:                       -2326.5
Covariance Type:            nonrobust   LLR p-value:                     0.000
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          -7.8291   5.91e+04     -0.000      1.000   -1.16e+05    1.16e+05
fishprawn_q    -0.1037   2.94e+05  -3.53e-07      1.000   -5.76e+05    5.76e+05
goatmeat_q     -0.0631   4.76e+05  -1.33e-07      1.000   -9.33e+05    9.33e+05
beef_q          0.0228   6.83e+05   3.34e-08      1.000   -1.34e+06    1.34e+06
pork_q          0.0062   1.15e+06   5.38e-09      1.000   -2.25e+06    2.25e+06
chicken_q      -0.1178   3.48e+05  -3.39e-07      1.000   -6.81e+05    6.81e+05
othrbirds_q    -0.0075   2.27e+06  -3.31e-09      1.000   -4.45e+06    4.45e+06
target         16.3537    1.1e+06   1.49e-05      1.000   -2.15e+06    2.15e+06
==============================================================================

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.
```

Interpretation :

The probit regression results in the provided image show that the model did not converge and there is an issue of complete separation or perfect prediction. Here's a detailed interpretation:

1. **Model Convergence**:

- The warning "Maximum number of iterations has been exceeded" indicates that the model did not converge within the allowed number of iterations. This typically means that the algorithm was unable to find a stable solution.

2. **Coefficients (coef)**:
   - The table lists the estimated coefficients for each predictor variable in the model.
   - `const` (constant term): -27.891
   - `fishpanq_q`: 2.501e-07
   - `goatmeat_q`: -1.373e-07
   - `beef_q`: -3.264e-08
   - `pork_q`: 5.297e-08
   - `chicken_q`: 1.339e-08
   - `otherbird_q`: -1.536e-08
   - `fishsalt_q`: -1.073e-07
3. **Standard Errors (std err)**:
   - The standard errors for each coefficient are extremely small, indicating a potential problem with the model estimation.
4. **P-values (P>|z|)**:
   - All the p-values are 1.000, suggesting that none of the coefficients are statistically significant. This is another indication of an issue with the model.
5. **Confidence Intervals (95% CI)**:
   - The confidence intervals for all coefficients are extremely narrow, which again points to potential issues with the model estimation process.
6. **Log-Likelihood**:
   - The log-likelihood value is -7.491e+12, which is extremely large in magnitude, further suggesting problems with the model fitting.
7. **Likelihood Ratio (LR) P-value**:
   - The LR test p-value is 0.000, indicating that the overall model is statistically significant. However, given the other issues, this result is likely not reliable.
8. **Complete Separation**:
   - The note at the bottom states that there is complete separation or perfect prediction. In such cases, the Maximum Likelihood Estimator does not exist, and the parameters are not identified. This means that the model is perfectly predicting the outcome, which usually indicates overfitting or a problem with the data.

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

Code and Result:

Tobit Regression

In [19]:
```
y = nss['foodtotal_v']
X = nss[['sauce_jam_v', 'Othrprocessed_v', 'Beveragestotal_v', 'fv_tot']]
```

In [20]:
```
class TobitModel:
    def __init__(self, endog, exog, lower=None, upper=None):
        self.endog = endog
        self.exog = exog
        self.lower = lower
        self.upper = upper

    def loglik(self, params):
        beta = params[:-1]
        sigma = params[-1]
        mu = np.dot(self.exog, beta)

        # Ensure sigma is positive
        sigma = np.abs(sigma) + 1e-10

        # Calculate the log-likelihood
        llf = np.zeros_like(self.endog, dtype=float)

        # Censored from below
        if self.lower is not None:
            llf = np.where(
                self.endog == self.lower,
                np.log(np.clip(norm.cdf((self.lower - mu) / sigma), 1e-10, 1)),
                llf
            )

        # Censored from above
        if self.upper is not None:
            llf = np.where(
                self.endog == self.upper,
                np.log(np.clip(1 - norm.cdf((self.upper - mu) / sigma), 1e-10, 1)),
                llf
            )
```

Result:

```
Tobit Model Results:
  message: CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
  success: True
   status: 0
      fun: 1616.2843163705104
        x: [-2.318e+01  4.297e+03  2.572e+03  1.152e+05  1.234e+01
             2.648e+02]
      nit: 92
      jac: [-2.274e-05  0.000e+00  0.000e+00  0.000e+00 -1.819e-04
             2.274e-05]
     nfev: 1309
     njev: 187
 hess_inv: <6x6 LbfgsInvHessProduct with dtype=float64>
```

Interpretation: results of a Tobit model estimation. Here's an interpretation of the results:

1. **Convergence Information**:
   - **Message**: `CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH` indicates that the optimization algorithm successfully converged according to the specified relative reduction in the function value criterion.
   - **Success**: `True` confirms that the model estimation was successful.
   - **Status**: `0` is another indicator of successful convergence without errors.
2. **Function Value (fun)**:
   - `1616.2803673575184` is the value of the log-likelihood function at the optimum. This value itself is not interpreted directly but is used to compare the fit of different models.
3. **Coefficients (x)**:
   - The estimated coefficients for the model are provided in the `x` vector:
     - `[-2.318e+03, 4.297e+03, 2.572e+03, 1.152e+05, 1.234e+01, -2.648e+02]`
   - Each of these values represents the effect of the corresponding predictor variable on the dependent variable. Without knowing the specific variables, we interpret these as the direction and magnitude of the relationship.
4. **Iterations (nit)**:
   - `92` indicates the number of iterations the optimization algorithm took to converge.
5. **Jacobian (jac)**:
   - The values `[2.274e-05, 0.000e+00, 0.000e+00, 0.000e+00, -8.319e-04, 2.274e-05]` represent the gradient (first derivatives) of the log-likelihood function at the optimal solution. These should ideally be close to zero, indicating that the solution is at a local maximum.
6. **Function Evaluations (nfev)**:
   - `1299` is the total number of function evaluations performed during the optimization process.
7. **Gradient Evaluations (njev)**:
   - `187` is the number of gradient evaluations performed.
8. **Inverse Hessian Matrix (hess_inv)**:
   - The output shows that the inverse Hessian is a `6x6 LbfgsInvHessProduct` with dtype=float64. This matrix is used to estimate the standard errors of the coefficients.

Summary

The Tobit model estimation was successful, as indicated by the convergence message and status. The coefficients represent the relationship between the predictor variables and the dependent variable. However, detailed interpretation of the coefficients and their significance would require additional information, such as the names and scales of the predictor variables, standard errors, and p-values, which are not provided in this output.

CODES

logistic regression analysis on your Credit Default Swap. 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 CODES)

```
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

os.chdir('/Users/kirthanshaker/Desktop/SCMA 631 Data Files ')

df = pd.read_csv('/Users/kirthanshaker/Desktop/SCMA 631 Data Files /Credit Card Defaulter
Prediction.csv')

## Data Cleaning and EDA

# 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()

#### DATA ENCODING FOR CATEGORICAL VARIABLES



# 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()

### TEST TRAIN SPLIT


# 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)

### SCALING THE DATA

# 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)

SELECT FEATURES BASED ON FEATURE IMPORTANCE

# 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]

FIT 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)

### ROC CURVE AND AUC VALUE

# 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()

## Confusion Matrix


# 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()

## DECISION TREE CLASSIFIER

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)

## ROC CURVE AND AUC VALUE



# 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()

### CONFUSION MATRIX



# 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 os
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
import statsmodels.api as sm
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
from scipy.optimize import minimize

os.chdir('/Users/kirthanshaker/Desktop/SCMA 631 Data Files ')

nss = pd.read_csv('/Users/kirthanshaker/Desktop/SCMA 631 Data Files /NSSO68.csv')

nss.shape

# Set working directory
os.chdir('/Users/kirthanshaker/Desktop/SCMA 631 Data Files ')

# Load dataset
data = pd.read_csv('/Users/kirthanshaker/Desktop/SCMA 631 Data Files /NSSO68.csv')

# Subset data to state assigned
bhr = data[data['state_1'] == 'Bhr'][['Religion', 'emftt_q', 'emftt_v']]
bhrnew = data[data['state_1'] ==
'Bhr'][['Religion','eggsno_q','fishprawn_q','goatmeat_q','beef_q','pork_q','chicken_q','othrbirds_q']]

# Check for missing values
print(bhr.isnull().sum())
print(bhrnew.isnull().sum())


bhr.shape, bhrnew.shape
```

```
religion_mapping = {1:'Hinduism', 2:'Islam', 3:'Christianity',5:'Jainism',9:'Others'}
bhrnew['Religion'] = bhrnew['Religion'].replace(religion_mapping)
print(bhrnew['Religion'].value_counts())

religion_counts = pd.Series([3944, 627, 8, 2, 1], index=['Hinduism', 'Islam', 'Christianity', 'Others',
'Jainism'])
plt.bar(religion_counts.index, religion_counts.values, color='skyblue')  # Adjust color as desired
plt.xlabel("Religion")
plt.ylabel("Count")
plt.title("Religion Distribution")
plt.xticks(rotation=45)
plt.show()

bhr.dtypes

columns = ['eggsno_q','fishprawn_q','goatmeat_q','beef_q','pork_q','chicken_q','othrbirds_q']
data = bhrnew[columns].copy()

data.dtypes

### Probit Regression

data['target'] = np.where(data['eggsno_q'] > 0,1,0)
x = data.drop(['eggsno_q'], axis = 1)
x = sm.add_constant(x)
y = data['target']

prodit_model = sm.Probit(y,x).fit()
print(prodit_model.summary())

religion_mapping = {1:'Hinduism', 2:'Islam', 3:'Christianity',5:'Jainism',9:'Others'}
bhr['Religion'] = bhr['Religion'].replace(religion_mapping)
print(bhr['Religion'].value_counts())

nv_by_religion = bhr.groupby('Religion')['emftt_q'].count()
max_nv_religion = nv_by_religion.idxmax()

print("The religion with the highest non-veg consumption is:", max_nv_religion)

### Tobit Regression

y = nss['foodtotal_v']
X = nss[['sauce_jam_v', 'Othrprocessed_v', 'Beveragestotal_v', 'fv_tot']]

class TobitModel:
    def __init__(self, endog, exog, lower=None, upper=None):
        self.endog = endog
        self.exog = exog
        self.lower = lower
        self.upper = upper

    def loglik(self, params):
```

```
        beta = params[:-1]
        sigma = params[-1]
        mu = np.dot(self.exog, beta)

        # Ensure sigma is positive
        sigma = np.abs(sigma) + 1e-10

        # Calculate the log-likelihood
        llf = np.zeros_like(self.endog, dtype=float)

        # Censored from below
        if self.lower is not None:
            llf = np.where(
                self.endog == self.lower,
                np.log(np.clip(norm.cdf((self.lower - mu) / sigma), 1e-10, 1)),
                llf
            )

        # Censored from above
        if self.upper is not None:
            llf = np.where(
                self.endog == self.upper,
                np.log(np.clip(1 - norm.cdf((self.upper - mu) / sigma), 1e-10, 1)),
                llf
            )

        # Uncensored
        uncensored = (self.endog > self.lower) & (self.endog < self.upper)
        llf[uncensored] = -0.5 * np.log(2 * np.pi) - np.log(sigma) - (self.endog[uncensored] -
mu[uncensored]) ** 2 / (2 * sigma ** 2)

        return -np.sum(llf)

    def fit(self):
        start_params = np.append(np.zeros(self.exog.shape[1]), 1)
        res = minimize(self.loglik, start_params, method='L-BFGS-B')
        return res

y_tobit = np.clip(y, 0, 1)
X_tobit = sm.add_constant(X)

model = TobitModel(y_tobit, X_tobit, lower=0, upper=1)
results = model.fit()

print("Tobit Model Results:")
print(results)

print(results)
```

R Codes

```
# Load necessary libraries
library(tidyverse)
library(caret)
library(ROCR)
library(e1071)
library(rpart)
library(rpart.plot)

# Set working directory and load the data
setwd("/Users/kirthanshaker/Desktop/SCMA 631 Data Files")
df <- read.csv("/Users/kirthanshaker/Desktop/SCMA 631 Data Files/Credit Card Defaulter Prediction.csv")

# Data Cleaning and EDA
# Remove spaces in column names
names(df) <- gsub(" ", "", names(df))

# Check for imbalance in the dataset
table(df$default)

# Drop the 'ID' column
df <- df %>% select(-ID)

# Encode categorical variables
cat_features <- df %>% select_if(is.character)

# Data Encoding for categorical variables
df <- df %>%
  mutate(across(where(is.character), as.factor)) %>%
  mutate(across(where(is.factor), as.integer))

# Final data
f_data <- df %>% select(-c(SEX, EDUCATION, MARRIAGE))

# Test-Train Split
set.seed(42)
trainIndex <- createDataPartition(f_data$default, p = .8,
                                  list = FALSE,
                                  times = 1)
x_train <- f_data[trainIndex, -ncol(f_data)]
x_test <- f_data[-trainIndex, -ncol(f_data)]
y_train <- f_data[trainIndex, ncol(f_data)]
y_test <- f_data[-trainIndex, ncol(f_data)]

# Scaling the data
scaler <- preProcess(x_train, method = c("range"))
x_train <- predict(scaler, x_train)
x_test <- predict(scaler, x_test)

# Feature selection using mutual information
mutual_info_scores <- varImp(FeatureSelection(y = y_train, x = x_train, method = "mutual_info"))
feature_scores_df <- mutual_info_scores %>%
  arrange(desc(Overall))

# Select top 15 features
selected_features <- head(feature_scores_df$var, 15)
feature_selection_train <- x_train[, selected_features]
feature_selection_test <- x_test[, selected_features]
```

```
# Logistic Regression
logreg <- glm(y_train ~ ., data = feature_selection_train, family = binomial)
y_pred <- predict(logreg, feature_selection_test, type = "response")
y_pred_class <- ifelse(y_pred > 0.5, 1, 0)

# Classification report for Logistic Regression
logreg_cm <- confusionMatrix(as.factor(y_pred_class), as.factor(y_test))
print(logreg_cm)

# ROC Curve and AUC for Logistic Regression
pred <- prediction(y_pred, y_test)
perf <- performance(pred, "tpr", "fpr")
roc_auc <- performance(pred, "auc")@y.values[[1]]

# Plot ROC Curve
plot(perf, col = "darkorange", lwd = 2, main = "ROC Curve for Logistic Regression")
abline(a = 0, b = 1, col = "navy", lwd = 2, lty = 2)
text(0.5, 0.5, paste("AUC =", round(roc_auc, 2)), pos = 4, col = "darkorange")

# Confusion Matrix for Logistic Regression
conf_matrix <- table(Predicted = y_pred_class, Actual = y_test)
print(conf_matrix)

# Decision Tree Classifier
dt_classifier <- rpart(y_train ~ ., data = feature_selection_train, method = "class")
y_pred_dt <- predict(dt_classifier, feature_selection_test, type = "class")

# Classification report for Decision Tree
dtree_cm <- confusionMatrix(as.factor(y_pred_dt), as.factor(y_test))
print(dtree_cm)

# ROC Curve and AUC for Decision Tree
y_pred_proba <- predict(dt_classifier, feature_selection_test, type = "prob")[, 2]
pred_dt <- prediction(y_pred_proba, y_test)
perf_dt <- performance(pred_dt, "tpr", "fpr")
roc_auc_dt <- performance(pred_dt, "auc")@y.values[[1]]

# Plot ROC Curve
plot(perf_dt, col = "darkorange", lwd = 2, main = "ROC Curve for Decision Tree")
abline(a = 0, b = 1, col = "navy", lwd = 2, lty = 2)
text(0.5, 0.5, paste("AUC =", round(roc_auc_dt, 2)), pos = 4, col = "darkorange")

# Confusion Matrix for Decision Tree
conf_matrix2 <- table(Predicted = y_pred_dt, Actual = y_test)
print(conf_matrix2)

# Parse Classification Report function
parse_classification_report <- function(cm) {
  data.frame(
    class = rownames(cm$byClass),
    precision = cm$byClass[, "Precision"],
    recall = cm$byClass[, "Recall"],
    f1_score = cm$byClass[, "F1"],
    support = cm$table[, "Support"]
  )
}

df1 <- parse_classification_report(dtree_cm)
df2 <- parse_classification_report(logreg_cm)
```

```
# Add model names and overall accuracy
df1$model <- "Decision Tree"
df2$model <- "Logistic Regression"

# Concatenate the two dataframes
comparison_df <- rbind(df1, df2)

# Reorder columns
comparison_df <- comparison_df %>% select(model, class, precision, recall, f1_score, support)

# Display the comparison table
print(comparison_df)
```