VIRGINIA COMMONWEALTH UNIVERSITY

Statistical analysis and modelling (SCMA 632)

A3: Logistic Regression , Probit Regression and Tobit Regression Using R and Python

CHANDHINI KERACHAN MURALEEDHARAN

V1107497

Date of Submission: 01-07-2024

GITHUB REPOSITORY LINK :

https://github.com/Chandhini-km

ASSIGNMENT IN THE REPOSITORY :

https://github.com/Chandhini-km/SCMA632-A3

CONTENTS

GITHUB REPOSITORY LINK :

https://github.com/Chandhini-km

A3 IN THE REPOSITORY :

https://github.com/Chandhini-km/SCMA632-A3

1. Introduction

1.1 About the Dataset

Pima Diabetes - The Pima Indians Diabetes Dataset, is a dataset has been used for binary classification tasks, specifically for predicting the onset of diabetes based on various medical predictors. The dataset contains 768 observations and includes 8 predictor variables: Pregnancies, Glucose, Blood Pressure, Skin Thickness, Insulin, BMI, Diabetes Pedigree Function, and Age, along with a binary Outcome variable indicating diabetes status. A key characteristic of this dataset is the imbalance in the classes, with more non-diabetic than diabetic instances. Additionally, some features contain zero values, likely placeholders for missing data, as zero is not physiologically plausible for measurements like Glucose or Blood Pressure. The dataset's numeric nature simplifies analysis and modelling processes. Typical use cases include binary classification, exploratory data analysis (EDA), and feature engineering, employing machine learning models such as logistic regression, decision trees, and neural networks. An analysis workflow involves loading the data, data cleaning, performing EDA, building and evaluating models, and interpreting the results to gain insights into diabetes-related factors.

Mizoram NSSO68 Dataset - The dataset includes information on consumption patterns, living conditions, and various socio-economic indicators of households. The focus here is on the subset of data specifically for the state of Mizoram.

The dataset is filtered to include only the records for Mizoram, represented by the state code "15".

1.2 Objectives

(i) To build and evaluate a logistic regression model to predict diabetes onset.

(ii) To identify non vegetarians based on food consumption data using a probit regression model.

(iii) To analyze censored data related to food consumption and expenditures using a Tobit regression model.

1.3 Business Significance

(i) **Logistic Regression on Pima Diabetes Dataset:**

- **Business Significance:** Understanding and predicting the onset of diabetes can help healthcare providers and policymakers design targeted interventions and preventive measures. Early identification of high-risk individuals allows for timely medical intervention, potentially reducing healthcare costs and improving patient outcomes. This model can also aid in resource allocation, ensuring that those at greatest risk receive the necessary medical attention and support.

(ii) **Probit Analysis to Get the Non-Vegetarians Data in NSSO68 Dataset for Mizoram:**

- **Business Significance:** Identifying the dietary habits of the population, particularly non-vegetarian preferences, can inform food policy and supply chain management. For businesses in the food industry, such insights can guide product development, marketing strategies, and inventory planning. Government agencies can use this data to address nutritional deficiencies and improve food security programs tailored to the dietary preferences of the population.

(iii) **Tobit Analysis on NSSO68 Dataset:**

- **Business Significance:** Analysing censored data on food consumption and expenditures provides valuable insights into consumer behaviour and economic well-being. Businesses can leverage this information to optimize pricing strategies, product offerings, and marketing efforts. For policymakers, understanding expenditure patterns helps in designing effective welfare programs and economic policies that address the needs of different demographic groups. This analysis also aids in identifying areas where economic support or subsidies might be necessary to ensure equitable access to essential goods and services.

2. Results

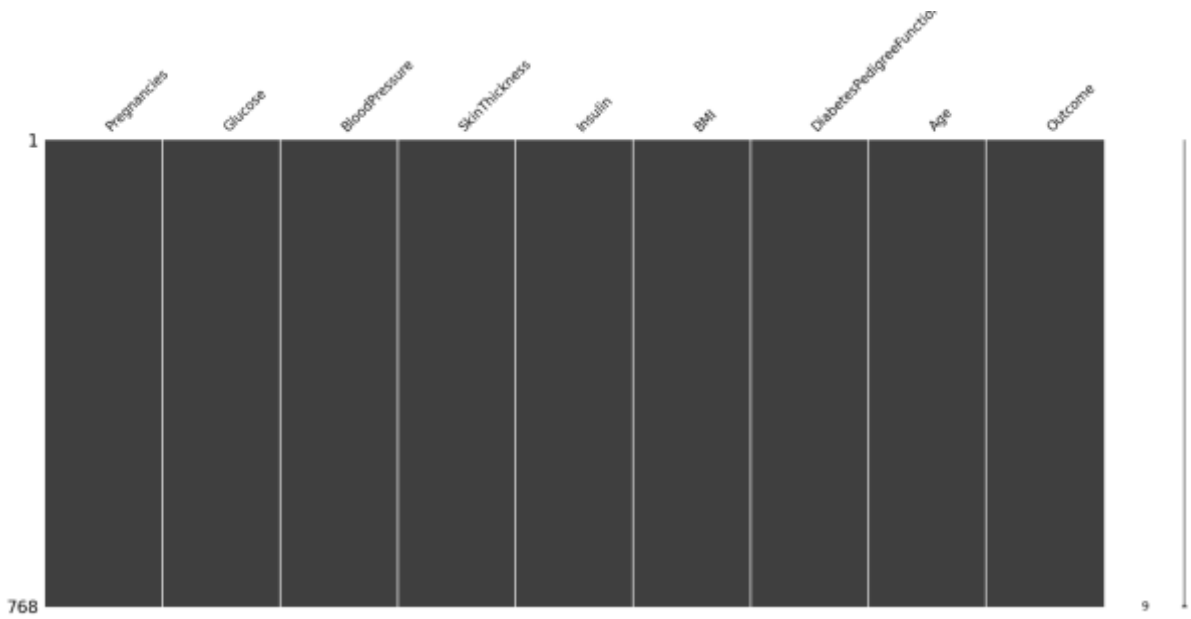
2.1 PART A : Results and interpretation of logistic regression analysis on PIMA DIABETES Dataset

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6
1              1       85             66             29        0  26.6
2              8      183             64              0        0  23.3
3              1       89             66             23       94  28.1
4              0      137             40             35      168  43.1
..           ...      ...            ...            ...      ...   ...
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     DiabetesPedigreeFunction  Age  Outcome
0                       0.627   50        1
1                       0.351   31        0
2                       0.672   32        1
3                       0.167   21        0
4                       2.288   33        1
..                        ...  ...      ...
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0

[768 rows x 9 columns]
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
```

Interpretation :

To make a deeper analysis of the loaded data. First we prints the entire data frame to provide a raw view of the information. Next, describe() method to generate summary statistics for each column. To visualize missing data patterns, we employs the msno.matrix(dia) function, we are creating a heatmap to see which columns have the most missing entries. We then quanty this information by printing the sum of missing values in each column using dia.isna().sum(). To get a quick look at specific data points, the code prints the first few rows (dia.head()) and the last few rows (dia.tail()) of

the dataframe. Finally printing the column names (dia.columns) and detailed information about the data types and memory usage using dia.info(). This comprehensive approach ensures a thorough understanding of the data before proceeding with further analysis.




```
Missing Values Information:
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Interpretation :

Missing Values Information: This table shows that there are zero missing values in any of the columns.

	Pregnancies	Glucose	BloodPressure	SkinThickness
Pregnancies	1.000000	0.129459	0.141282	-0.081672
Glucose	0.129459	1.000000	0.152590	0.057328
BloodPressure	0.141282	0.152590	1.000000	0.207371
SkinThickness	-0.081672	0.057328	0.207371	1.000000
Insulin	-0.073535	0.331357	0.088933	0.436783
BMI	0.017683	0.221071	0.281805	0.392573
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928
Age	0.544341	0.263514	0.239528	-0.113970
Outcome	0.221898	0.466581	0.065068	0.074752

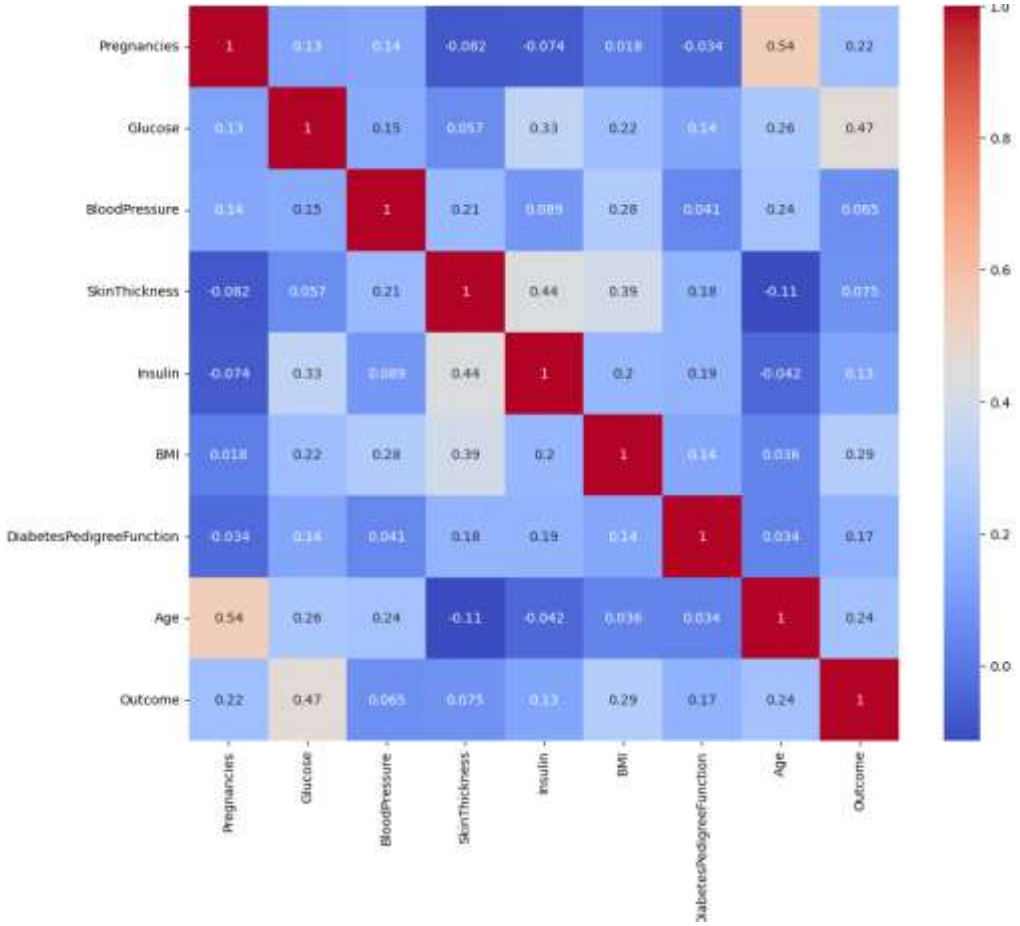
	Insulin	BMI	DiabetesPedigreeFunction
Pregnancies	-0.073535	0.017683	-0.033523
Glucose	0.331357	0.221071	0.137337
BloodPressure	0.088933	0.281805	0.041265
SkinThickness	0.436783	0.392573	0.183928
Insulin	1.000000	0.197859	0.185071
BMI	0.197859	1.000000	0.140647
DiabetesPedigreeFunction	0.185071	0.140647	1.000000
Age	-0.042163	0.036242	0.033561
Outcome	0.130548	0.292695	0.173844

	Age	Outcome
Pregnancies	0.544341	0.221898
Glucose	0.263514	0.466581
BloodPressure	0.239528	0.065068
SkinThickness	-0.113970	0.074752
Insulin	-0.042163	0.130548
BMI	0.036242	0.292695
DiabetesPedigreeFunction	0.033561	0.173844
Age	1.000000	0.238356
Outcome	0.238356	1.000000

Interpretation :

A correlation matrix, which is a way to visualize the relationships between different variables. Each cell in the matrix shows the correlation coefficient between two variables. The correlation coefficient is a number between -1 and 1 that indicates how much two variables tend to move together. A correlation coefficient of 1 indicates a perfect positive correlation, which means that as the value of one variable increases, the value of the other variable also increases. A correlation coefficient of -1 indicates a perfect negative correlation, which means that as the value of one variable increases, the value of the other variable decreases. A correlation coefficient of 0 indicates no correlation between the two variables.

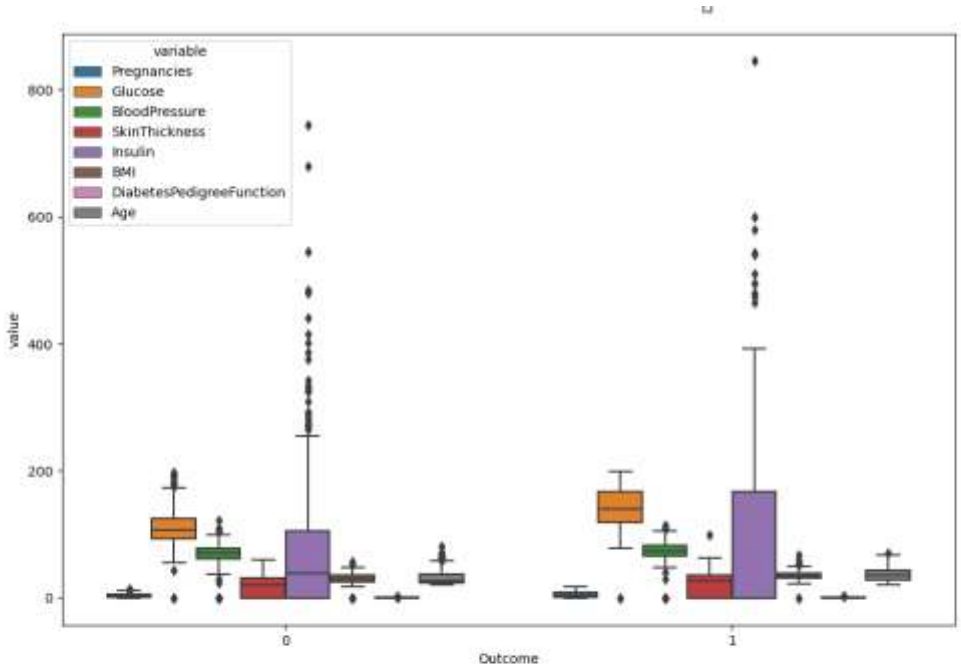
 Blood Pressure and BMI: There seems to be a weak positive correlation, indicated by a light red shade. This suggests that as BMI increases, there's a slight tendency for blood pressure to also increase.
 Insulin and Age: Another weak positive correlation is present here. As age increases, insulin levels might also tend to increase slightly.
 Glucose and BMI: A weak positive correlation exists between these two. This means there might be a slight tendency for blood glucose levels to go up with increasing BMI.
 Outcome (Diabetes) and Other Factors: There appears to be a weak positive correlation between having diabetes (Outcome) and factors like Age, BMI, Glucose, Insulin, and SkinThickness. This suggests that people with higher values in these measures might be more likely to have diabetes, but the correlation is weak.



Interpretation :

Scatter plot, which is a type of visual representation that shows the relationship between two variables. Each point on the scatter plot represents a single data point. The horizontal axis (x-axis) represents one variable, and the vertical axis (y-axis) represents another variable.

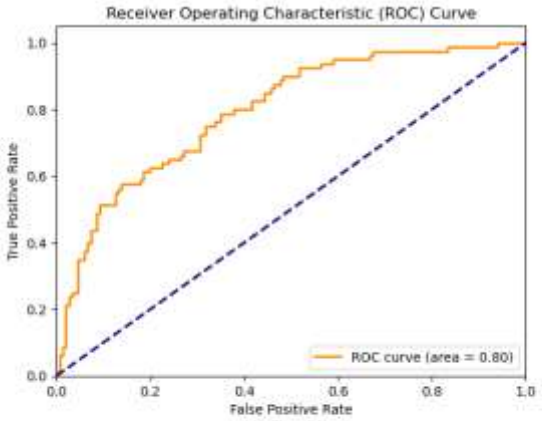
In this specific scatter plot, the x-axis represent blood pressure and the y-axis appears to represent blood sugar. There is a weak positive correlation between the two variables. This means that as blood pressure increases, blood sugar tends to increase slightly as well. However, the scatter plot also shows a lot of variability, which means that there are many data points that fall outside of this general trend.




```
Confusion Matrix:
 [[120  31]
 [ 30  50]]
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.79      0.80       151
           1       0.62      0.62      0.62        80

    accuracy                           0.74       231
   macro avg       0.71      0.71      0.71       231
weighted avg       0.74      0.74      0.74       231
```

Interpretation :

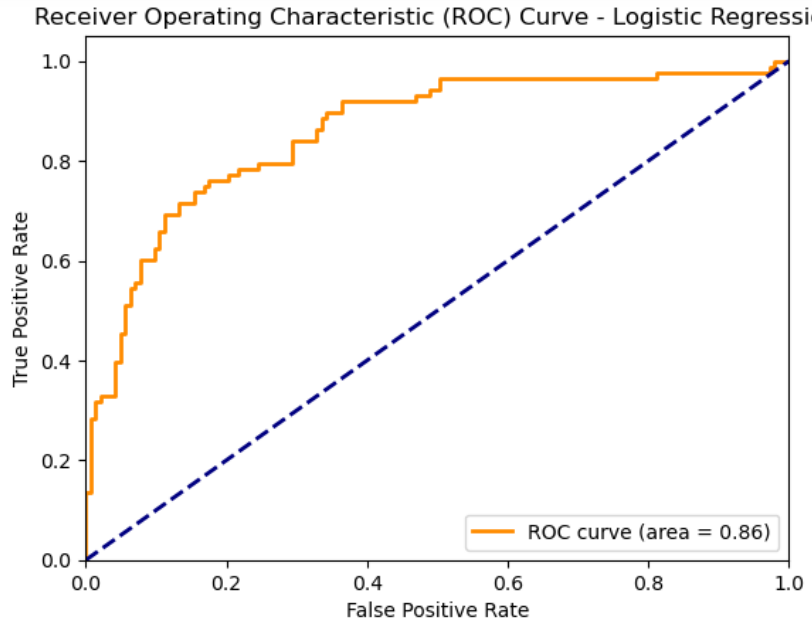
 Based on the confusion matrix, we can see that the model correctly classified 170 (120 + 50) out of 231 instances. This gives the model an accuracy of 73.6% (170 / 231).
 Looking at the classification report, we see that the model performs better for class 0 (no diabetes) with a precision of 0.80 and a recall of 0.80.
 The model's performance for class 1 (diabetes) is not as good, with a precision of 0.62 and a recall of 0.62.




```
Optimization terminated successfully.
         Current function value: 0.459388
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:                Outcome   No. Observations:                  537
Model:                          Logit   Df Residuals:                      528
Method:                           MLE   Df Model:                            8
Date:                Sun, 30 Jun 2024   Pseudo R-squ.:                  0.2905
Time:                        08:54:28   Log-Likelihood:                -246.80
converged:                       True   LL-Null:                       -347.71
Covariance Type:            nonrobust   LLR p-value:                 2.578e-39
============================================================================================
                               coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------------------
const                       -9.4451      0.915    -10.321      0.000     -11.239      -7.651
Pregnancies                  0.0580      0.039      1.477      0.140      -0.019       0.135
Glucose                      0.0359      0.005      7.714      0.000       0.027       0.045
BloodPressure               -0.0108      0.007     -1.584      0.113      -0.024       0.003
SkinThickness               -0.0015      0.008     -0.179      0.858      -0.018       0.015
Insulin                     -0.0010      0.001     -0.884      0.377      -0.003       0.001
BMI                          0.1090      0.019      5.740      0.000       0.072       0.146
DiabetesPedigreeFunction     0.4215      0.357      1.182      0.237      -0.278       1.120
Age                          0.0359      0.011      3.186      0.002       0.014       0.058
============================================================================================
```

Interpretation :

The area under the ROC curve (AUC) appears to be acceptable, likely somewhere in the range of 0.7 to 0.8. An AUC of 1 represents a perfect model, while 0.5 signifies a random guess.

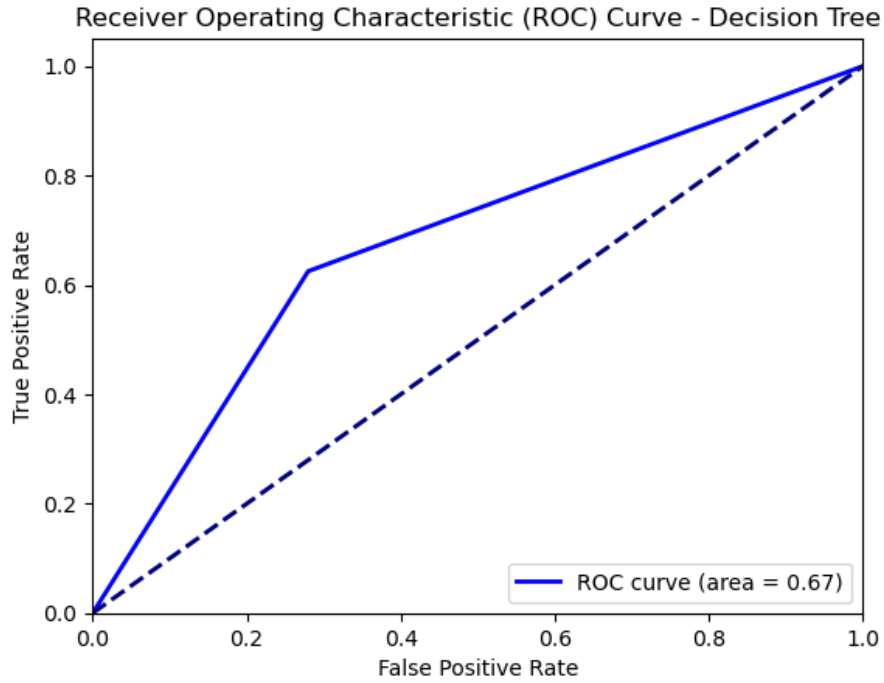



```
AUC-ROC (Logistic Regression): 0.8610934520025428
Confusion Matrix (Decision Tree):
 [[103  40]
 [ 33  55]]
```

Interpretation :

The AUC for the model in the image is 0.861. This is a good AUC, and it indicates that the model is performing well. Here are some key points to interpret the results from the ROC curve and AUC in the image:

- The model is good at distinguishing between people with and without the disease, but it is not perfect. There will be some cases where the model incorrectly classifies a healthy person as having the disease (false positive) or a person with the disease as healthy (false negative).
- The cost of false positives and false negatives will depend on the specific application. For example, in a medical test for a serious disease, a false negative could be much more costly than a false positive. On the other hand, in a spam filter, a false positive (mistakenly classifying a legitimate email as spam) might be less costly than a false negative (mistakenly classifying spam as a legitimate email).




```
AUC-ROC (Decision Tree): 0.6726398601398601
Logistic Regression vs Decision Tree
Confusion Matrix (Logistic Regression):
 [[128  15]
 [ 32  56]]
Confusion Matrix (Decision Tree):
 [[103  40]
 [ 33  55]]
AUC-ROC (Logistic Regression): 0.8610934520025428
AUC-ROC (Decision Tree): 0.6726398601398601
```

Interpretation :

- The ROC curve for the decision tree model is closer to the diagonal line than the logistic regression model, which suggests it performs worse at distinguishing between positive and negative cases.
- The AUC for the decision tree model is 0.67, which is significantly lower than the logistic regression model's AUC of 0.86. This confirms that the logistic regression model performs better at classifying instances.

Overall, the logistic regression model performs better than the decision tree model based on the AUC values and the position of the ROC curves.

2.2 PART B : Performing Probit regression on NSSO68 to identify nonvegetarians

```
In [2]: import os
        import pandas as pd
        import numpy as np
        from scipy import stats

In [ ]: # reading the file

In [4]: df=pd.read_excel("C:\\Users\\Chand\\Downloads\\Miz.xlsx")

In [ ]: # Extracting Mizoram data

In [5]: Mizoram = df[['Age','Religion','eggsno_q','fishprawn_q','goatmeat_q','beef_q','pork_q','chicken_q','othrbirds_q']]

In [6]: print(Mizoram.describe())
                Age     Religion     eggsno_q  fishprawn_q   goatmeat_q  \
count  1536.000000  1536.000000  1536.000000  1536.000000  1536.000000
mean     46.694661     3.117188     0.000186     0.121713     0.006977
std      11.810717     0.766100     0.000187     0.223880     0.069212
min      18.000000     1.000000     0.000000     0.000000     0.000000
25%      38.000000     3.000000     0.000055     0.000000     0.000000
50%      47.000000     3.000000     0.000138     0.000000     0.000000
75%      54.000000     3.000000     0.000275     0.200000     0.000000
max      87.000000     6.000000     0.001650     3.000000     1.500000

           beef_q       pork_q    chicken_q  othrbirds_q
count  1536.000000  1536.000000  1536.000000  1536.000000
mean      0.106797     0.572749     0.140367     0.050279
std       0.214431     0.446186     0.233761     0.152804
min       0.000000     0.000000     0.000000     0.000000
25%       0.000000     0.250000     0.000000     0.000000
50%       0.000000     0.500000     0.000000     0.000000
75%       0.166667     0.833333     0.250000     0.000000
max       1.600000     3.000000     1.500000     1.500000

In [7]: print(Mizoram.shape)
        (1536, 9)

In [8]: Mizoram.isnull().sum()
Out[8]: Age          0
        Religion     0
```

Interpretation :

To get only Mizoram Information.

```
In [15]: z_score = (subset - subset.mean()) / subset.std()

In [16]: outliers = (z_score > 3) | (z_scores <-3)

In [17]: subset_no_outliers = subset[~outliers.any(axis=1)]

In [18]: subset_no_outliers.reset_index(drop=True, inplace=True)

In [19]: print("Mizoram without Outliers:\n",subset_no_outliers)
```

```
Mizoram without Outliers:
       Age  Religion  eggsno_q  fishprawn_q  goatmeat_q    beef_q    pork_q  \
0      29         3  0.000138     0.000000         0.0  0.000000  0.750000
1      54         3  0.000000     0.000000         0.0  0.000000  1.333333
2      45         3  0.000000     0.000000         0.0  0.142857  0.571429
3      61         3  0.000550     0.000000         0.0  0.250000  0.500000
4      36         3  0.000000     0.000000         0.0  0.000000  1.333333
...   ...       ...       ...          ...         ...       ...       ...
1299   36         3  0.000183     0.000000         0.0  0.000000  0.666667
1300   42         3  0.000110     0.000000         0.0  0.200000  0.400000
1301   47         3  0.000138     0.166667         0.0  0.166667  0.333333
1302   48         3  0.000132     0.000000         0.2  0.000000  0.400000
1303   40         3  0.000060     0.000000         0.0  0.000000  0.500000

      chicken_q  othrbirds_q
0      0.000000         0.25
1      0.000000         0.00
2      0.000000         0.00
3      0.000000         0.00
4      0.000000         0.00
...         ...          ...
1299   0.000000         0.00
1300   0.000000         0.00
1301   0.166667         0.00
1302   0.000000         0.00
1303   0.250000         0.00

[1304 rows x 9 columns]
```

Interpretation :

Mizoram data without outliers.

```
In [23]: # Find unique values in the 'Religion' column
         unique_religions = df['Religion'].unique()

         # Print the unique religions
         print(unique_religions)
```

```
[3 1 2 6 4]
```

```
In [24]: # christianity 3 , Hinduism 1, Islam 2, Buddhism 6, Sikhism 4

In [25]: religion_mapping={1:'Hinduism',2:'Islam',3:'Christianity',4:'Sikhism',6:'Buddhism'}

In [26]: subset_no_outliers['Religion']=subset_no_outliers['Religion'].replace(religion_mapping)
```

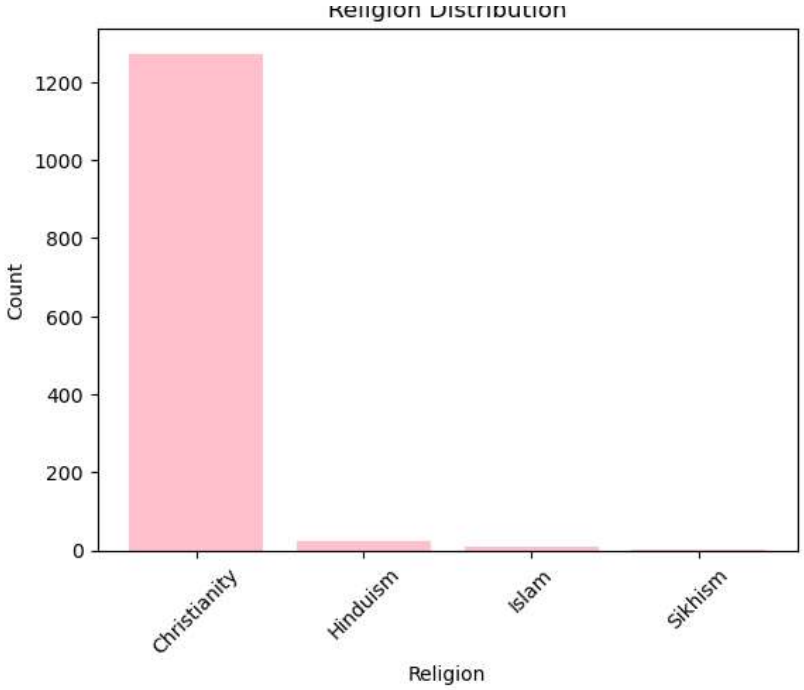
```
C:\Users\Chand\AppData\Local\Temp\ipykernel_19724\2193113389.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vie
-versus-a-copy
  subset_no_outliers['Religion']=subset_no_outliers['Religion'].replace(religion_mapping)
```

```
In [36]: print(subset_no_outliers['Religion'].value_counts())
```

```
Religion
Christianity    1273
Hinduism          22
Islam              8
Sikhism            1
Name: count, dtype: int64
```

Interpretation :

Region codes for Mizoram in 3,1,2,6,4




```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.000000
         Iterations: 35
                           Probit Regression Results
==============================================================================
Dep. Variable:                 target   No. Observations:                 1304
Model:                         Probit   Df Residuals:                     1296
Method:                           MLE   Df Model:                            7
Date:                Mon, 01 Jul 2024   Pseudo R-squ.:                   1.000
Time:                        14:16:10   Log-Likelihood:            -6.8055e-05
converged:                      False   LL-Null:                       -661.03
Covariance Type:            nonrobust   LLR p-value:                2.817e-281
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          -4.8934     62.077     -0.079      0.937    -126.563     116.776
fishprawn_q     0.4854    231.534      0.002      0.998    -453.314     454.285
goatmeat_q     -0.6296   6331.076  -9.94e-05      1.000   -1.24e+04    1.24e+04
beef_q          0.7463    112.149      0.007      0.995    -219.062     220.555
pork_q         -1.7143    269.636     -0.006      0.995    -530.191     526.762
chicken_q      -0.7828    232.260     -0.003      0.997    -456.005     454.439
othrbirds_q     0.5550    144.768      0.004      0.997    -283.185     284.295
target         13.0878    445.393      0.029      0.977    -859.867     886.043
==============================================================================

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 :

Output of complete separation or perfect prediction. independent variables perfectly predicts the outcome variable. In other words, for certain values of the independent variable(s), we can always know exactly what the outcome will be.

2.3 PART C : Tobit Model on NSSO Data

```
In [11]: print("Tobit Model Results:")
Tobit Model Results:

In [12]: print(results)
  message: CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
  success: True
   status: 0
      fun: 1630.8623801597212
        x: [-3.332e+02  1.674e+02  1.565e+02  1.236e+04  1.675e+02
             3.541e+03]
      nit: 67
      jac: [-3.342e-03  0.000e+00  0.000e+00 -4.547e-05 -2.724e-02
             1.114e-03]
     nfev: 805
     njev: 115
 hess_inv: <6x6 LbfgsInvHessProduct with dtype=float64>
```

Interpretation :

The results of the regression model show that the model converged successfully. This means that the algorithm that was used to fit the model was able to find a set of coefficients that optimizes the model's fit to the data.

3. Codes

3.1 PART A Python and R

PYTHON

```
pip install missingno

import pandas as pd


# Ensure missingno is installed and imported correctly

try:

    import missingno as msno

except ImportError:

    import subprocess

    subprocess.check_call(["pip", "install", "missingno"])

    import missingno as msno
```

```
# Load the dataset
dia = pd.read_csv("C:\\Users\\Chand\\Downloads\\pima-diabetes.csv")

# Display the dataframe
print(dia)

# Summary of the dataframe
print(dia.describe())

# Plot missing values
msno.matrix(dia)

# Sum of missing values
print(dia.isna().sum())

# Display the first few rows
print(dia.head())

# Display the last few rows
print(dia.tail())

# Display the column names
print(dia.columns)

# Display the structure of the dataframe
print(dia.info())
# Replace missing values with the mean for numeric columns
dia = dia.apply(lambda col: col.fillna(col.mean()) if col.dtype.kind in 'bifc' else col)

# Checking missing values after filling them with mean values of the column
```

```
missing_info = dia.isna().sum()
print("Missing Values Information:")
print(missing_info)
# Replace missing values with the mean for numeric columns
dia = dia.apply(lambda col: col.fillna(col.mean()) if col.dtype.kind in 'bifc' else col)

# Split the data into training and test sets
X = dia.drop(columns=['Outcome'])
y = dia['Outcome']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=123)

# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Logistic Regression
log_reg = LogisticRegression(max_iter=10000)
log_reg.fit(X_train, y_train)
predicted_probs = log_reg.predict_proba(X_test)[:, 1]
predicted_class = (predicted_probs >= 0.5).astype(int)

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, predicted_class)
print("Confusion Matrix:\n", conf_matrix)

# ROC and AUC
roc_auc = roc_auc_score(y_test, predicted_probs)
fpr, tpr, _ = roc_curve(y_test, predicted_probs)
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 (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

print(f"AUC-ROC: {roc_auc}")
# Decision Tree Analysis
decision_tree = DecisionTreeClassifier(random_state=123)
decision_tree.fit(X_train, y_train)
dt_predicted_class = decision_tree.predict(X_test)
dt_predicted_probs = decision_tree.predict_proba(X_test)[:, 1]

# Confusion Matrix for Decision Tree
dt_conf_matrix = confusion_matrix(y_test, dt_predicted_class)
print("Decision Tree Confusion Matrix:\n", dt_conf_matrix)

# ROC and AUC for Decision Tree
dt_roc_auc = roc_auc_score(y_test, dt_predicted_probs)
dt_fpr, dt_tpr, _ = roc_curve(y_test, dt_predicted_probs)
plt.figure()
plt.plot(dt_fpr, dt_tpr, color='blue', lw=2, label='ROC curve (area = %0.2f)' % dt_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 (ROC) Curve - Decision Tree')
plt.legend(loc="lower right")
plt.show()

print(f"AUC-ROC (Decision Tree): {dt_roc_auc}")

# Compare the results
print("Logistic Regression vs Decision Tree")
print("Confusion Matrix (Logistic Regression):\n", conf_matrix)
print("Confusion Matrix (Decision Tree):\n", dt_conf_matrix)
print(f"AUC-ROC (Logistic Regression): {roc_auc}")
print(f"AUC-ROC (Decision Tree): {dt_roc_auc}")
```

R

```
library(DataExplorer)
library(dplyr)
#pima diabetics
dia<-read.csv("C:\\Users\\Chand\\Downloads\\pima-diabetes.csv")
dia
summary(dia)
plot_missing(dia)
sum(is.na(dia))
head(dia)
tail(dia)
names(dia)
str(dia)

# Replace missing values with the mean for numeric columns
dia <- dia %>%
```

```
  mutate(across(where(is.numeric), ~ ifelse(is.na(.), mean(., na.rm = TRUE), .)))
#Checking missing values after filling it with mean values of the column
missing_info <- colSums(is.na(dia))
cat("Missing Values Information:\n")
print(missing_info)

#performing logistic regression , validate assumptions , and evaluating the performance with a confusion matrix
#and ROC curve and interpreting the results

cor_matrix<-cor(dia)
print(cor_matrix)
heatmap(cor_matrix)
boxplot(Pregnancies+Glucose+BloodPressure+SkinThickness+Insulin+BMI+DiabetesPedigreeFunction+Age~Outcome,data=dia)
install.packages("caTools")
install.packages("MLmetrics")

library(dplyr)
library(caTools)
library(pROC)
library(rpart)
library(rpart.plot)
library(MLmetrics)

# Logistic Regression
set.seed(123)
split<-sample.split(dia$Outcome,SplitRatio = 0.7)
train<-subset(dia,split==TRUE)
test<-subset(dia,split==FALSE)
model<-glm(Outcome~.,data=train,family=binomial)
```

```
predicted_probs<-predict(model,newdata=test,type="response")
predicted_class<-ifelse(predicted_probs>=0.5,1,0)

# Confusion Matrix
CM<-ConfusionMatrix(factor(predicted_class),factor(test$Outcome))
print(CM)
roc_obj<-roc(test$Outcome,predicted_probs)
auc<-auc(roc_obj)
print(paste("AUC-ROC:",auc))
plot(roc_obj,main="ROC Curve",print.auc=TRUE)

#decision tree analysis for the data in part A and compare the results of the
#Logistic regression and Decision tree
library(stats)
install.packages("rpart")
library(rpart)
install.packages("caTools")
library(caTools)



set.seed(123)
split<-sample.split(dia$Outcome,SplitRatio = 0.7)
train<-subset(dia,split == TRUE)
test<-subset(dia,split == FALSE)
model<-rpart(Outcome~.,data=train,method="class")
predicted_probs<-predict(model,newdata=test,type="prob")
predicted_class<-ifelse(predicted_probs[,2]>=0.5,1,0)

ConfM<-ConfusionMatrix(factor(predicted_class),factor(test$Outcome))
```

```
print(ConfM)-
roc_obj<-roc(test$Outcome,predicted_probs[,2])
auc<-auc(roc_obj)
print(paste("AUC-ROC:",auc))
plot(roc_obj,main="ROC Curve",print.auc=TRUE)
```

3.2. PART B Python and R

PYTHON

```
import os
import pandas as pd
import numpy as np
from scipy import stats
df=pd.read_excel("C:\\Users\\Chand\\Downloads\\Miz.xlsx")
Mizoram =
df[['Age','Religion','eggsno_q','fishprawn_q','goatmeat_q','beef_q','pork_q','chicken_q','othrbir
ds_q']]
print(Mizoram.describe())
Mizoram.isnull().sum()
print(Mizoram.columns)
Mizoram.dtypes
z_scores = (Mizoram - Mizoram.mean()) /Mizoram.std()
outliers = (z_scores > 3) | (z_scores < -3)
print("Outliers:\n",outliers)
subset = Mizoram
z_score = (subset - subset.mean()) / subset.std()
outliers = (z_score > 3) | (z_scores <-3)
subset_no_outliers.reset_index(drop=True, inplace=True)
print("Mizoram without Outliers:\n",subset_no_outliers)
# Find unique values in the 'Religion' column
unique_religions = df['Religion'].unique()
```

```
# Print the unique religions
print(unique_religions)
religion_mapping={1:'Hinduism',2:'Islam',3:'Christianity',4:'Sikhism',6:'Buddhism'}
subset_no_outliers['Religion']=subset_no_outliers['Religion'].replace(religion_mapping)


print(subset_no_outliers['Religion'].value_counts())
religion_counts = pd.Series([1273, 22, 8, 1],
index=['Christianity','Hinduism','Islam','Sikhism'])
plt.bar(religion_counts.index, religion_counts.values, color='pink')  # Adjust color as desired
plt.xlabel("Religion")
plt.ylabel("Count")
plt.title("Religion Distribution")
plt.xticks(rotation=45)
plt.show()
print(subset_no_outliers['Religion'].value_counts())
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
subset_no_outliers['target'] = np.where(subset_no_outliers['eggsno_q'] > 0,1,0)
x = subset_no_outliers.drop(['eggsno_q'], axis = 1)
x = sm.add_constant(x)
y = subset_no_outliers['target']
columns = ['eggsno_q','fishprawn_q','goatmeat_q','beef_q','pork_q','chicken_q','othrbirds_q']
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())
```

R

```
setwd('C:\\Users\\Chand\\Downloads\\A3')

# Function to install and load libraries
install_and_load <- function(package) {
  if (!require(package, character.only = TRUE)) {
    install.packages(package, dependencies = TRUE)
    library(package, character.only = TRUE)
  }
}

# Load required libraries
libraries <- c("dplyr", "readr", "readxl", "tidyr", "ggplot2", "BSDA","glue")
lapply(libraries, install_and_load)

# Reading the file into R
data <- read.csv("C:\\Users\\Chand\\Downloads\\A3\\NSSO68 (3).csv")
dim(data)

unique(data$Religion)

# Filtering for Mizoram
MIZ<- data %>%
  filter(state == "15")
```

```
# Display dataset info
cat("Dataset Information:\n")
print(names(MIZ))
print(head(MIZ))
print(dim(MIZ))

# Finding missing values
missing_info <- colSums(is.na(MIZ))
cat("Missing Values Information:\n")
print(missing_info)

# Sub-setting the data
miznew <- MIZ %>%
  select(state_1,Religion, District, Region, Sector,emftt_q, emftt_v)

# Check for missing values in the subset
cat("Missing Values in Subset:\n")
print(colSums(is.na(miznew)))

dim(miznew)

# Impute missing values with mean for specific columns
impute_with_mean <- function(column) {
  if (any(is.na(column))) {
    column[is.na(column)] <- mean(column, na.rm = TRUE)
  }
  return(column)
}
miznew$emftt_q <- impute_with_mean(miznew$emftt_q)
miznew$emftt_v <- impute_with_mean(miznew$emftt_v)
```

```
dim(miznew)

# Check for missing values after imputation
cat("Missing Values After Imputation:\n")
print(colSums(is.na(miznew)))

MIZ$Religion

miznew$emftt_v

MIZ$Religion

unique(MIZ$Religion)
str(MIZ$Religion)

# Fitting a probit regression to identify non-vegetarians.

religion_mapping <- c("Hinduism", "Islam", "Christianity","Sikkhism","Buddhism")
MIZ$Religion <- factor(MIZ$Religion, labels = religion_mapping)
table(MIZ$Religion)

columns <- c('emftt_v','emftt_q')
data1 <- MIZ[columns]
data1$target <- ifelse(data1$emftt_v>0,1,0)
probit_modet <- glm(target~., data = data1, family = binomial(link = "probit"))
summary(probit_modet)
```

3.3 PART C Python and R

PYTHON

```
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
from scipy.stats import norm
from scipy.optimize import minimize

data = pd.read_csv("C:\\Users\\Chand\\Downloads\\NSSO68 (3).csv", low_memory=False)

display (data)

print(data.columns)

y = data['foodtotal_v']
X = data[['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)
```

R

```
# Performorming a Tobit regression analysis on "NSSO68.csv"
df_MIZ = data[data$state_1 == 'MIZ',]
vars <- c("state_1","Religion", "District", "Region", "Sector","emftt_q", "emftt_v")

df_MIZ_p = df_MIZ[vars]
names(df_MIZ_p)

df_MIZ_p$price = df_MIZ_p$emftt_v / df_MIZ_p$emftt_q
names(df_MIZ_p)

summary(df_MIZ_p)

head(table(df_MIZ_p$emftt_q))

dim(df_MIZ_p)

names(MIZ)
```

```
#  dependent variable and independent variables
y <- MIZ$foodtotal_v
X <- MIZ[, c("sauce_jam_v", "Othrprocessed_v", "Beveragestotal_v", "fv_tot")]

#  data for Tobit regression
y_tobit <- pmin(pmax(y, 0), 1)
X_tobit <- cbind(1, X)

install.packages("censReg")
library(censReg)
# Fitting the Tobit model
X_tobit_df <- as.data.frame(X_tobit)
model <- censReg(y_tobit ~ ., data = X_tobit_df[, -1])

# Printing model summary
summary(model)
```

GITHUB REPOSITORY LINK :

https://github.com/Chandhini-km

A3 IN THE REPOSITORY :

https://github.com/Chandhini-km/SCMA632-A3