In [23]:
```
import pandas as pd
import numpy as np
import matplotlib as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
```

In [24]:
```
#Reading data from the Credit Score Classification dataset
creditScore = pd.read_csv("Credit Score Classification Dataset.csv")
```

In [25]:
```
#Displaying the first seven rows
creditScore.head(7)
```

Out[25]:

	Age	Gender	Income	Education	Marital Status	Number of Children	Home Ownership	Credit Score
0	25	Female	50000	Bachelor's Degree	Single	0	Rented	High
1	30	Male	100000	Master's Degree	Married	2	Owned	High
2	35	Female	75000	Doctorate	Married	1	Owned	High
3	40	Male	125000	High School Diploma	Single	0	Owned	High
4	45	Female	100000	Bachelor's Degree	Married	3	Owned	High
5	50	Male	150000	Master's Degree	Married	0	Owned	High
6	26	Female	40000	Associate's Degree	Single	0	Rented	Average

In [26]:
```
#Displaying detailed information
creditScore.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 164 entries, 0 to 163
Data columns (total 8 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Age                     164 non-null    int64
 1   Gender                  164 non-null    object
 2   Income                  164 non-null    int64
 3   Education               164 non-null    object
 4   Marital Status          164 non-null    object
 5   Number of Children      164 non-null    int64
 6   Home Ownership          164 non-null    object
 7   Credit Score            164 non-null    object
dtypes: int64(3), object(5)
memory usage: 10.4+ KB
```

This displays more information about the dataset in detail. That is, the data types of each column, the number of non-null values and the memory usage.

In [27]:
```
#Performing statistical operations on the dataset
creditScore.describe()
```

Out[27]:

	Age	Income	Number of Children
count	164.000000	164.000000	164.000000
mean	37.976513	93755.243902	0.652439
std	8.477208	32457.664728	0.882046
min	25.000000	25000.000000	0.000000
25%	30.750000	57500.000000	0.000000
50%	37.000000	83750.000000	0.000000
75%	45.000000	105000.000000	1.000000
max	63.000000	162500.000000	3.000000

The output shows the results of calculations based on the statistics of numerical columns

In [28]:
```
#Checking for missing values
print(creditScore.isnull().sum())
```

```
Age                     0
Gender                  0
Income                  0
Education               0
Marital Status          0
Number of Children      0
Home Ownership          0
Credit Score            0
dtype: int64
```

The `isnull()` function is used to determine whether each element in a DataFrame or Series is null(missing) or not. When you chain the `sum()` function with `isnull()`, it calculates the number of missing values in each column of the DataFrame. For example, if you have a DataFrame named `footballPerformance` and you want to know how many missing values are there in each column, you can use `footballPerformance.isnull().sum()`. This will return a Series where the index represents column names, and the values represent the count of missing values in each respective column. Judging from the output, it appears that there are no null values.

In [29]:
```
#Checking for the duplicate rows and displaying them
duplicateRows = creditScore.duplicated(keep=False)
print(duplicateRows)
```

```
0      False
1      False
2      False
3      False
4      False
       ...
159    True
160    True
161    True
162    True
163    True
Length: 164, dtype: bool
```

In [30]:
```
creditScore_no_duplicates = creditScore.drop_duplicates(keep=False)
print(creditScore_no_duplicates)
```

```
     Age  Gender  Income                Education Marital Status  \
0     25  Female   50000          Bachelor's Degree       Single
1     30    Male  100000          Master's Degree      Married
2     35  Female   75000                 Doctorate      Married
3     40    Male  125000  High School Diploma          Single
4     45  Female  100000          Bachelor's Degree      Married
..    ...     ...     ...                        ...          ...
159   31    Male   85000                 Doctorate       Single
161   26  Female   55000          Bachelor's Degree      Married
162   32    Male   85000          Master's Degree       Single
163   29  Female   60000                 Doctorate      Married
164   34    Male  105000          Bachelor's Degree      Married

     Number of Children  Home Ownership Credit Score
0                     0          Rented         High
1                     2           Owned         High
2                     1           Owned         High
3                     0           Owned         High
4                     3           Owned         High
..                  ...             ...          ...
159                   0          Rented         High
161                   1           Owned      Average
162                   0          Rented         High
163                   2           Owned      Average
164                   1          Rented         High

[96 rows x 8 columns]
```

In [31]:
```
#Displaying the data types of the dataset
creditScore.dtypes
```

Out[31]:
```
Age                     int64
Gender                 object
Income                  int64
Education              object
Marital Status         object
Number of Children      int64
Home Ownership         object
Credit Score           object
dtype: object
```

In [32]:
```
#Transforming the datatypes of some columns from categorical to numeric data types using label encoding which assigns
#a unique numerical value to each category
labelEncoder = LabelEncoder()
creditScore['Education'] = labelEncoder.fit_transform(creditScore['Education'])
```

In [33]:
```
#Transforming the datatypes of some columns from categorical to numeric data types using label encoding which assigns
#a unique numerical value to each category
labelEncoder = LabelEncoder()
creditScore['Marital Status'] = labelEncoder.fit_transform(creditScore['Marital Status'])
```

In [34]:
```
#Transforming the datatypes of some columns from categorical to numeric data types using label encoding which assigns
#a unique numerical value to each category
labelEncoder = LabelEncoder()
creditScore['Home Ownership'] = labelEncoder.fit_transform(creditScore['Home Ownership'])
```

In [35]:
```
#Transforming the datatypes of some columns from categorical to numeric data types using label encoding which assigns
#a unique numerical value to each category
labelEncoder = LabelEncoder()
creditScore['Credit Score'] = labelEncoder.fit_transform(creditScore['Credit Score'])
```

In [36]:
```
creditScore.tail()
```

Out[36]:

	Age	Gender	Income	Education	Marital Status	Number of Children	Home Ownership	Credit Score
159	29	0	27500	3	1	0	1	2
160	34	1	47500	0	1	0	1	0
161	39	0	62500	1	0	2	0	1
162	44	1	87500	4	1	0	0	1
163	49	0	77500	2	0	1	0	1

In [37]:
```
creditScore["Credit Score"].value_counts()
```

Out[37]:
```
Credit Score
1    113
0     36
2     15
Name: count, dtype: int64
```

The **`value_counts()`** function is a Pandas method that, when applied to a Series (like a DataFrame column), counts the occurrences of unique values in that Series. It returns a new Series where the unique values in the original Series are the index labels, and the corresponding values are the counts of each unique value. Thus, When you run `creditScore["Credit Score"].value_counts()`, it will output a Series that shows the count of each unique value in the Credit Score' column.

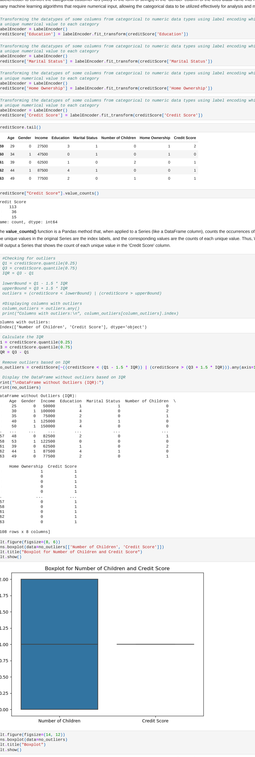
In [38]:
```
#Checking for outliers
# Q1 = creditScore.quantile(0.25)
# Q3 = creditScore.quantile(0.75)
# IQR = Q3 - Q1

# lowerBound = Q1 - 1.5 * IQR
# upperBound = Q3 + 1.5 * IQR
# outliers = (creditScore < lowerBound) | (creditScore > upperBound)

# #Displaying columns with outliers
# column_outliers = outliers.any()
# print("Columns with outliers:", column_outliers[column_outliers].index)
```

```
Columns with outliers:
Index(['Number of Children', 'Credit Score'], dtype='object')
```

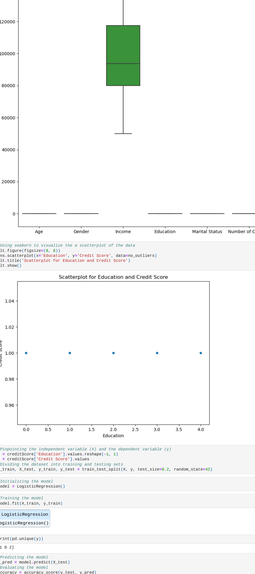
In [39]:
```
# Calculate the IQR for the specified columns
Q1 = creditScore.quantile(0.25)
Q3 = creditScore.quantile(0.75)
IQR = Q3 - Q1

# Remove outliers based on IQR
no_outliers = creditScore[~((creditScore < (Q1 - 1.5 * IQR)) | (creditScore > (Q3 + 1.5 * IQR)).any(axis=1))]

# Display the DataFrame without outliers based on IQR
print("\nDataFrame without Outliers (IQR):")
print(no_outliers)
```

```
DataFrame without Outliers (IQR):
     Age  Gender  Income  Education  Marital Status  Number of Children  \
0     25       0   50000          1               1                   0
1     30       1  100000          3               0                   2
2     35       0   75000          2               1                   1
3     40       1  125000          3               1                   0
5     50       1  150000          4               1                   0
..    ...    ...     ...        ...             ...                 ...
157   48       0   82500          2               0                   1
158   53       1  112500          4               1                   0
161   39       0   62500          1               0                   2
162   44       1   87500          4               1                   0
163   49       0   77500          2               0                   1

     Home Ownership  Credit Score
0                 1             1
1                 0             1
2                 0             1
3                 0             1
5                 0             1
..              ...           ...
157               0             1
158               0             1
161               0             1
162               0             1
163               0             1

[108 rows x 8 columns]
```

In [40]:
```
plt.figure(figsize=(4, 6))
sns.boxplot(dataset[['Number of Children', 'Credit Score']])
plt.title('Boxplot for Number of Children and Credit Score')
plt.show()
```



In [41]:
```
plt.figure(figsize=(4, 12))
sns.boxplot(dataset[outliers])
plt.title("Boxplot")
plt.show()
```



In [42]:
```
#Using seaborn to visualize the a scatterplot of the data
plt.figure(figsize=(4, 6))
sns.scatterplot(x="Education", y="Credit Score", dataset=no_outliers)
plt.title('Scatterplot for Education and Credit Score')
plt.show()
```

In [43]:
```
#Dispatching the independent variable (X) and the dependent variable (y)
X = creditScore['Education'].values.reshape(-1, 1)
y = creditScore['Credit Score'].values
#Dividing the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [44]:
```
#Initializing the model
model = LogisticRegression()
```

In [45]:
```
#Training the model
model.fit(X_train, y_train)
```

Out[45]:
```
▾ LogisticRegression
LogisticRegression()
```

In [46]:
```
print(np.unique(y))
```

```
[1 0 2]
```

In [47]:
```
#Predicting the model
y_pred = model.predict(X_test)
#Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred, zero_division=1)
#Printing results
print("Accuracy: {accuracy}")
print("Confusion Matrix:\n{conf_matrix}")
print("Classification Report:\n{classification_report}")
```

```
Accuracy: 0.7272727272727273
Confusion Matrix:
[[ 1  4  0]
 [ 0 23  0]
 [ 1  4  0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.20      0.29         5
           1       0.74      1.00      0.85        23
           2       1.00      0.00      0.00         5

    accuracy                           0.73        33
   macro avg       0.75      0.40      0.38        33
weighted avg       0.74      0.73      0.59        33
```

In [48]:
```
#Visualizing the confusion matrix using a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', annot_kws={"size": 16}, cbar=False,
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```