Machine Learning Methods and Applications (IST438)

HOMEWORK-2

AIM: The aim of this project is train a logistic regression model that will predict either a customer with given features have high or low credit risk using the *Credit Risk Classification Dataset* from Kaggle
Link to dataset

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,f1_score,confusion_matrix,classification_report,RocCurveDisplay
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
```

DATA IMPORT

In [2]:
```
df = pd.read_csv("customer_data.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	label	id	fea_1	fea_2	fea_3	fea_4	fea_5	fea_6	fea_7	fea_8	fea_9	fea_10	fea_11
0	1	54982665	5	1245.5	3	77000.0	2	15	5	109	5	151300	244.948974
1	0	59004779	4	1277.0	1	113000.0	2	8	-1	100	3	341759	207.173840
2	0	58990862	7	1298.0	1	110000.0	2	11	-1	101	5	72001	1.000000
3	1	58995168	7	1335.5	1	151000.0	2	11	5	110	3	60084	1.000000
4	0	54987320	7	NaN	2	59000.0	2	11	5	108	4	450081	197.403141

Pre-Processing

In [4]:
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1125 entries, 0 to 1124
Data columns (total 13 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   label   1125 non-null   int64  
 1   id      1125 non-null   int64  
 2   fea_1   1125 non-null   int64  
 3   fea_2   976 non-null    float64
 4   fea_3   1125 non-null   int64  
 5   fea_4   1125 non-null   float64
 6   fea_5   1125 non-null   int64  
 7   fea_6   1125 non-null   int64  
 8   fea_7   1125 non-null   int64  
 9   fea_8   1125 non-null   int64  
 10  fea_9   1125 non-null   int64  
 11  fea_10  1125 non-null   int64  
 12  fea_11  1125 non-null   float64
dtypes: float64(3), int64(10)
memory usage: 114.4 KB
None
```

The dataset consist of 1125 rows and 13 columns in total, except the id and label, all the columns are encoded

In [5]:
```
df.describe().T
```

Out[5]:

	count	mean	std	min	25%	50%	75%	max
label	1125.0	2.000000e-01	4.001779e-01	0.0	0.0	0.000000e+00	0.000000e+00	1.000000e+00

id	1125.0	5.783677e+07	1.817150e+06	54982353.0	54990497.0	5.898975e+07	5.899799e+07	5.900624e+07
fea_1	1125.0	5.482667e+00	1.383338e+00	1.0	4.0	5.000000e+00	7.000000e+00	7.000000e+00
fea_2	976.0	1.283911e+03	5.176402e+01	1116.5	1244.0	1.281500e+03	1.314500e+03	1.481000e+03
fea_3	1125.0	2.333333e+00	8.787730e-01	1.0	1.0	3.000000e+00	3.000000e+00	3.000000e+00
fea_4	1125.0	1.208836e+05	8.844523e+04	15000.0	72000.0	1.020000e+05	1.390000e+05	1.200000e+06
fea_5	1125.0	1.928889e+00	2.571247e-01	1.0	2.0	2.000000e+00	2.000000e+00	2.000000e+00
fea_6	1125.0	1.087200e+01	2.676437e+00	3.0	8.0	1.100000e+01	1.100000e+01	1.600000e+01
fea_7	1125.0	4.832889e+00	2.971182e+00	-1.0	5.0	5.000000e+00	5.000000e+00	1.000000e+01
fea_8	1125.0	1.008027e+02	1.198896e+01	64.0	90.0	1.050000e+02	1.110000e+02	1.150000e+02
fea_9	1125.0	4.195556e+00	8.556790e-01	1.0	3.0	4.000000e+00	5.000000e+00	5.000000e+00
fea_10	1125.0	1.646185e+05	1.525205e+05	60000.0	60044.0	7.200000e+04	1.513070e+05	6.500700e+05
fea_11	1125.0	1.349990e+02	1.126168e+02	1.0	1.0	1.732051e+02	2.024846e+02	7.071068e+02

Since all the columns were encoded without any information about them, its not possible for me to understand what features they represent and manipulate them.

However, the column 'fea_2' has missing values, and I can't decide if I should drop the column, or the rows that contain missing values with that much of information.

In [6]:
```
df[df["fea_2"].isna()]["label"].mean()
```

Out[6]:
```
0.2483221476510067
```

In [7]:
```
df.dropna()["label"].mean()
```

Out[7]:
```
0.19262295081967212
```

Since there might be an statistical significance of 'fea_2' being nan, dropping the rows with missing values would create a bias. However, its not possible to impute the observations since the column is completely encoded and I don't know anything about what it represents. For example, the column might be representing the monthly income of the customer, and the customers with no income might be represented with nulls. Henceforward, it wouldn't be a good decision to fill the missing values with mean or median of the column.

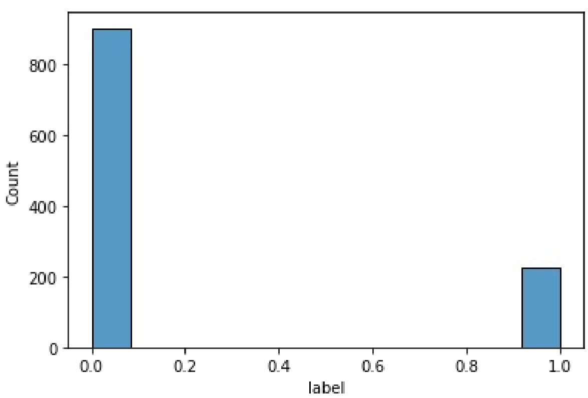
Instead of creating a biased model, or making false assumptions and building the model on top of them, while dropping the entire column might lower the accuracy of the model, its the best solution in that problem that I could find.

In [8]:
```
df = df.drop(["fea_2"],axis=1)
```

Examining the distribution of target column

In [9]:
```
sns.histplot(data = df, x= "label")
```

Out[9]:
```
<AxesSubplot:xlabel='label', ylabel='Count'>
```



In [10]:
```
print(f'Number of positive observations: {sum(df["label"] == 1)}\nNumber of negative observations: {sum(df["label
```

```
Number of positive observations: 225
Number of negative observations: 900
```

Its clearly visible that the target column is not balanced.

So any model that would predict only 0's will have an accuracy score of nearly 80 percent. This is a huge handicap for training a model.

Because of this situation, I decided to create two models, one that directly trained on the given data, and another one trained on an oversampled training data.

Model 1 - Initial Model

In [11]:
```
X_train, X_test,y_train,y_test = train_test_split(df.drop(["label"],axis=1),df["label"],test_size = 0.2, random_s
```

In [12]:
```
print(f'Number of training observations: {len(X_train)}\nNumber of test observations: {len(X_test)}')
```

```
Number of training observations: 900
Number of test observations: 225
```

Model Build

In [13]:
```
Model = LogisticRegression(random_state = 42)
Model.fit(X_train,y_train)
```

Out[13]:
```
LogisticRegression(random_state=42)
```

In [14]:
```
y_pred = Model.predict(X_test)
```

In [15]:
```
print(accuracy_score(y_test,y_pred))
```

```
0.7644444444444445
```

In [16]:
```
print(confusion_matrix(y_test,y_pred))
```

```
[[172   0]
 [ 53   0]]
```

In [17]:
```
print(classification_report(y_test,y_pred,zero_division=0))
```

```
              precision    recall  f1-score   support

           0       0.76      1.00      0.87       172
           1       0.00      0.00      0.00        53

    accuracy                           0.76       225
   macro avg       0.38      0.50      0.43       225
weighted avg       0.58      0.76      0.66       225
```

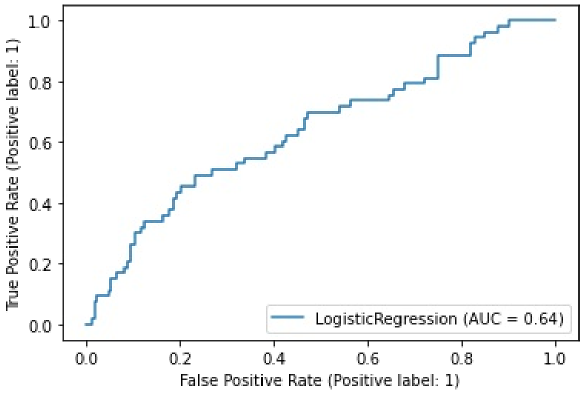
We have an accuracy score more than .8, this means that our model predicted more than 80 percent of the values correctly, however, from the confusion matrix, it can be easily seen that that model only predicted 1's and no 0's. This is just like I have expected since the data is ibmalanced

In [18]:
```
print(f1_score(y_test,y_pred))
```

```
0.0
```

In [19]:
```
RocCurveDisplay.from_estimator(Model,X_test,y_test)
```

Out[19]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x269447eb588>
```



Model 2 - Oversampled Model

In [20]:
```
smote = SMOTE(random_state = 42)
```

In [21]:
```
X_smote,y_smote = smote.fit_resample(X_train, y_train)
```

In [22]:
```
print(f'Number of training observation: {len(X_train)}')
```

```
Number of training observation: 900
```

Model Build

In [23]:
```
Model_2 = LogisticRegression(random_state=42)
Model_2.fit(X_smote,y_smote)
```

Out[23]:
```
LogisticRegression(random_state=42)
```

In [24]:
```
y_pred_2 = Model_2.predict(X_test)
```

In [25]:
```
print(accuracy_score(y_test,y_pred_2))
```

```
0.5066666666666667
```

In [26]:
```
print(confusion_matrix(y_test,y_pred_2))
```

```
[[77 95]
 [16 37]]
```

In [27]:
```
print(f1_score(y_test,y_pred_2))
```

```
0.39999999999999997
```

In [28]:
```
print(classification_report(y_test,y_pred_2))
```

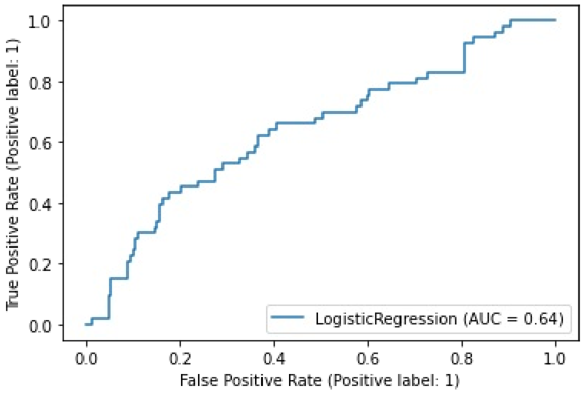
```
              precision    recall  f1-score   support

           0       0.83      0.45      0.58       172
           1       0.28      0.70      0.40        53
```

```
    accuracy                           0.51       225
   macro avg       0.55      0.57      0.49       225
weighted avg       0.70      0.51      0.54       225
```

While the accuracy score of the model is lower than the initial one, It can be seen that f1 score has slightly increased and confusion matrix looks a lot better now.

In [29]:
```
RocCurveDisplay.from_estimator(Model_2,X_test,y_test)
```

Out[29]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x269449a29c8>



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js