|  |  |
| --- | --- |
| نام و نام خانوادگی | **محمد پویا افشاری – علیرضا اسمعیل زاده** |
| شماره دانشجویی | **810198351-810198577** |
| تاریخ ارسال گزارش | **1402.09.17** |

|  |  |  |
| --- | --- | --- |
|  | **به نام خدا**  **دانشگاه تهران**  **دانشکده‌ مهندسی برق و کامپیوتر** |  |
| **درس شبکه‌های عصبی و یادگیری عمیق**  **تمرین امتیازی شماره 1** | | |

**فهرست**

[**پاسخ 1** – 3](#_Toc155681867)

[۱-۱ 3](#_Toc155681868)

[**پاسخ** ۳ - تشخیص تقلب 3](#_Toc155681869)

[۳-۱. 3](#_Toc155681870)

[۳-۱-۱. کتابخانه 3](#_Toc155681871)

[۳-۱-۲. نمودار هیستوگرام کلاس‌ها 3](#_Toc155681872)

[۳-۱-۳. چرا نمی توانیم این کلاس ها را اموزش بدهیم؟ 4](#_Toc155681873)

[۳-۲. پياده سازي مدل مقاله 4](#_Toc155681874)

[۳-۲-۱. پيش پردازش 4](#_Toc155681875)

[۳-۲-۲. آموزش با داده هاي unbalanced 5](#_Toc155681876)

[۳-۳. نمونه برداري – به كمك Adaptive Synthetic Sampling (ADASYN) 8](#_Toc155681877)

[۳-۳-۳. توزيح دهيد نمونه برداري بايد قبل از تقسيم داده ها به اموزش و تست انجام بشود يا بعد از ان؟ 12](#_Toc155681878)

[۳-۴. آموزش مدل 14](#_Toc155681879)

**شکل‌ها**

شکل 1. عنوان تصویر نمونه 4

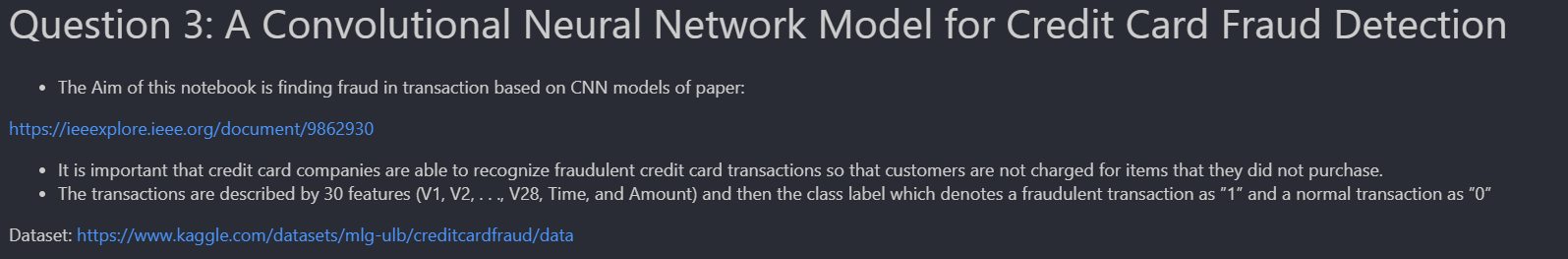
# **پاسخ 1** –

## ۱-۱

# **پاسخ** ۳ - تشخیص تقلب

## ۳-۱.

### ۳-۱-۱. کتابخانه



import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import cv2

from sklearn.preprocessing import RobustScaler

import os

import seaborn as sns

import shutil

from google.colab import drive

from google.colab import files

### ۳-۱-۲. نمودار هیستوگرام کلاس‌ها

uploaded = files.upload()

!mkdir -p ~/.kaggle

!mv kaggle.json ~/.kaggle/

!chmod 600 ~/.kaggle/kaggle.json

!kaggle datasets download -d mlg-ulb/creditcardfraud

!unzip -q creditcardfraud.zip

df = pd.read\_csv("./creditcard.csv")

df

df['Class'].value\_counts()

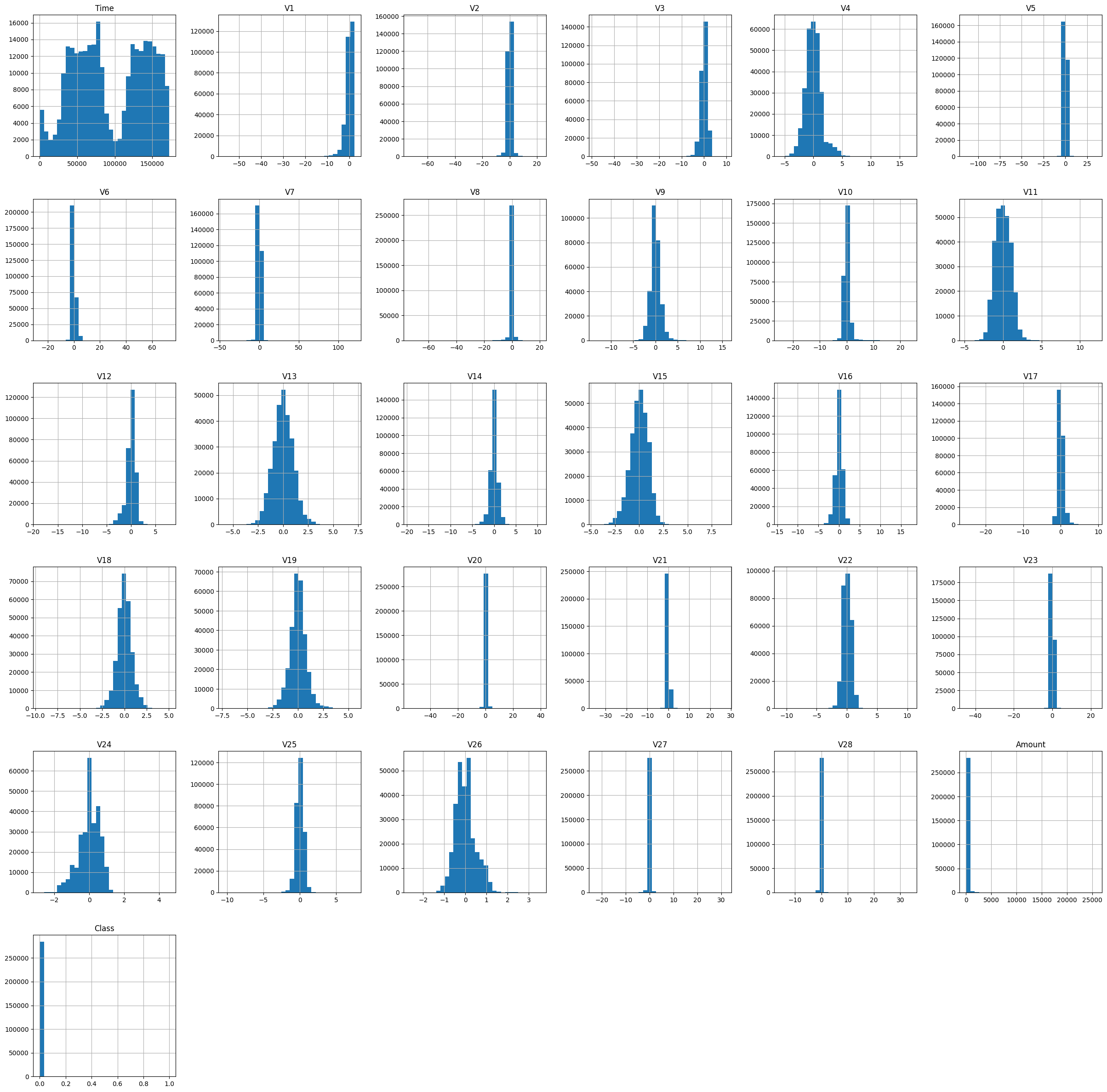
0 284315

1 492

Name: Class, dtype: int64

خب در اینجا متوجه نابرابری توزیع کلاس ها می‌شویم

df.hist(bins=30,figsize=(30,30))



df.describe()

### ۳-۱-۳. چرا نمی توانیم این کلاس ها را اموزش بدهیم؟

دیتاست داده شده از تراکنش های حاصله دو روز کاری 492 تا entry fraud و 284.807 تا تراکنش عادی دارد. خب این دیتاست بسیار unbalanced هست برای کلاس fraud و 0.172% کل تراکنش را شامل می‌شود. براي همين هنگام اموزش خيلي روي كلاس negative fragility دارد.

چون 99% تراكنش ها شيادي حساب نمي‌شود الگوريتم ها با احتمال زياد تري هميشه non-fraud براورد مي‌كنند. با اين حالت با درصد بالايي دقت 99% بدست مي‌آيد در داده train. ما دنبال اين كار نيستيم و دنبال generalize بودن هستيم بنابر اين نياز داريم كه new labeling داشته باشيم كه توزيع درست براورد كنيم.

## ۳-۲. پياده سازي مدل مقاله

### ۳-۲-۱. پيش پردازش

new\_df = df.copy()

time = new\_df['Time']

new\_df['Amount'] = RobustScaler().fit\_transform(new\_df['Amount'].to\_numpy().reshape(-1,1))

new\_df['Time'] = (time - time.min()) / (time.max() - time.min())

### ۳-۲-۲. آموزش با داده هاي unbalanced

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv1D, BatchNormalization, MaxPool1D, Dropout, Flatten, Dense

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.losses import BinaryCrossentropy

from tensorflow.keras.metrics import Accuracy

X = df.drop('Class', axis=1)

y = df['Class']

X\_unbalanced = X.values

y\_unbalanced = y.values

X\_unbalanced\_reshaped = X\_unbalanced.reshape(X\_unbalanced.shape[0], X\_unbalanced.shape[1], 1)

X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X\_unbalanced\_reshaped, y\_unbalanced, test\_size=0.4, random\_state=42)

X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42)

print("Training set shape:", X\_train.shape)

print("Validation set shape:", X\_val.shape)

print("Testing set shape:", X\_test.shape)

Training set shape: (170884, 30, 1)

Validation set shape: (56961, 30, 1)

Testing set shape: (56962, 30, 1)

model = Sequential()

model.add(Conv1D(filters=32, kernel\_size=2, activation='relu', input\_shape=(X\_train.shape[1], 1)))

model.add(BatchNormalization())

model.add(MaxPool1D(pool\_size=2))

model.add(Dropout(rate=0.2))

model.add(Conv1D(filters=64, kernel\_size=2, activation='relu'))

model.add(BatchNormalization())

model.add(MaxPool1D(pool\_size=2))

model.add(Dropout(rate=0.5))

model.add(Flatten())

model.add(Dense(units=64, activation='relu'))

model.add(Dropout(rate=0.5))

model.add(Dense(units=64, activation='relu'))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(optimizer=Adam(learning\_rate=0.0001),

              loss=BinaryCrossentropy(),

              metrics=[Accuracy()])

model.summary()

Model: "sequential\_4"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv1d\_7 (Conv1D) (None, 29, 32) 96

batch\_normalization\_6 (Bat (None, 29, 32) 128

chNormalization)

max\_pooling1d\_6 (MaxPoolin (None, 14, 32) 0

g1D)

dropout\_9 (Dropout) (None, 14, 32) 0

conv1d\_8 (Conv1D) (None, 13, 64) 4160

batch\_normalization\_7 (Bat (None, 13, 64) 256

chNormalization)

max\_pooling1d\_7 (MaxPoolin (None, 6, 64) 0

g1D)

dropout\_10 (Dropout) (None, 6, 64) 0

flatten\_3 (Flatten) (None, 384) 0

...

Total params: 33505 (130.88 KB)

Trainable params: 33313 (130.13 KB)

Non-trainable params: 192 (768.00 Byte)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

history = model.fit(X\_train, y\_train, epochs=6, batch\_size=32, validation\_data=(X\_val, y\_val))

Epoch 1/6

5341/5341 [==============================] - 45s 8ms/step - loss: 0.0183 - accuracy: 0.0000e+00 - val\_loss: 0.0064 - val\_accuracy: 0.0000e+00

Epoch 2/6

5341/5341 [==============================] - 38s 7ms/step - loss: 0.0078 - accuracy: 0.0000e+00 - val\_loss: 0.0058 - val\_accuracy: 1.7556e-05

Epoch 3/6

5341/5341 [==============================] - 38s 7ms/step - loss: 0.0066 - accuracy: 5.8519e-06 - val\_loss: 0.0048 - val\_accuracy: 0.0000e+00

Epoch 4/6

5341/5341 [==============================] - 38s 7ms/step - loss: 0.0059 - accuracy: 0.0000e+00 - val\_loss: 0.0044 - val\_accuracy: 0.0000e+00

Epoch 5/6

5341/5341 [==============================] - 38s 7ms/step - loss: 0.0052 - accuracy: 0.0000e+00 - val\_loss: 0.0039 - val\_accuracy: 0.0000e+00

Epoch 6/6

5341/5341 [==============================] - 38s 7ms/step - loss: 0.0053 - accuracy: 0.0000e+00 - val\_loss: 0.0043 - val\_accuracy: 0.0000e+00

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Training and Validation Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Training and Validation Accuracy')

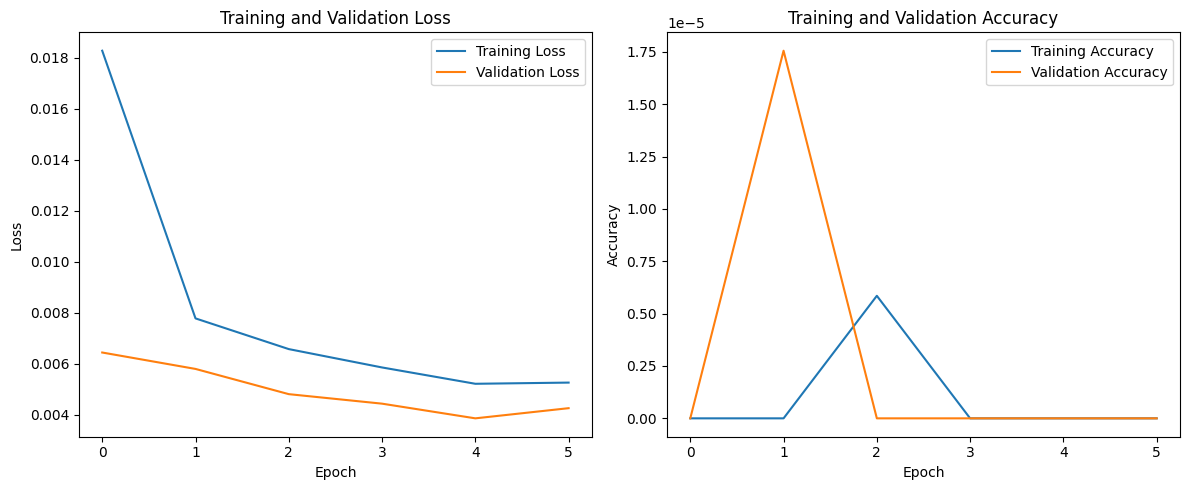
plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.tight\_layout()

plt.show()



from sklearn.metrics import confusion\_matrix, classification\_report

import seaborn as sns

import matplotlib.pyplot as plt

*# Obtain model predictions for the test data*

y\_pred = model.predict(X\_test)

y\_pred\_binary = (y\_pred > 0.5).astype(int)  *# Convert to binary predictions (0 or 1)*

*# Calculate and plot the confusion matrix heatmap*

cm = confusion\_matrix(y\_test, y\_pred\_binary)

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,

            xticklabels=['Legitimate', 'Fraud'],

            yticklabels=['Legitimate', 'Fraud'])

plt.title('Confusion Matrix')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

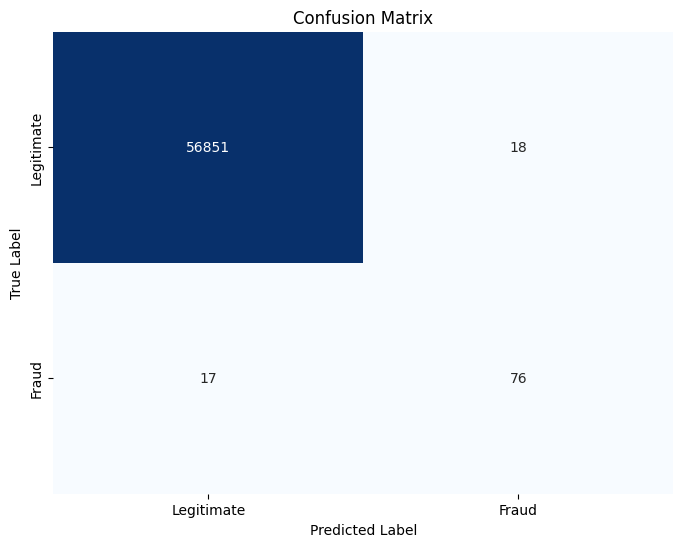
plt.show()

*# Report precision, recall, and F1 score*

report = classification\_report(y\_test, y\_pred\_binary, target\_names=['Legitimate', 'Fraud'])

print("Classification Report:")

print(report)



Classification Report:

precision recall f1-score support

Legitimate 1.00 1.00 1.00 56869

Fraud 0.81 0.82 0.81 93

accuracy 1.00 56962

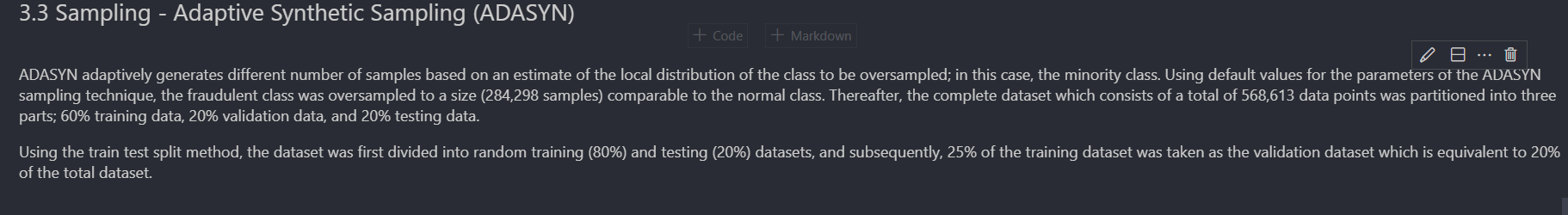
macro avg 0.90 0.91 0.91 56962

weighted avg 1.00 1.00 1.00 56962

خب همانطور كه اشاره كرديم احتمال مي‌رفت كه گزارش نتايج بالا قابل اتكا كردن نباشد.

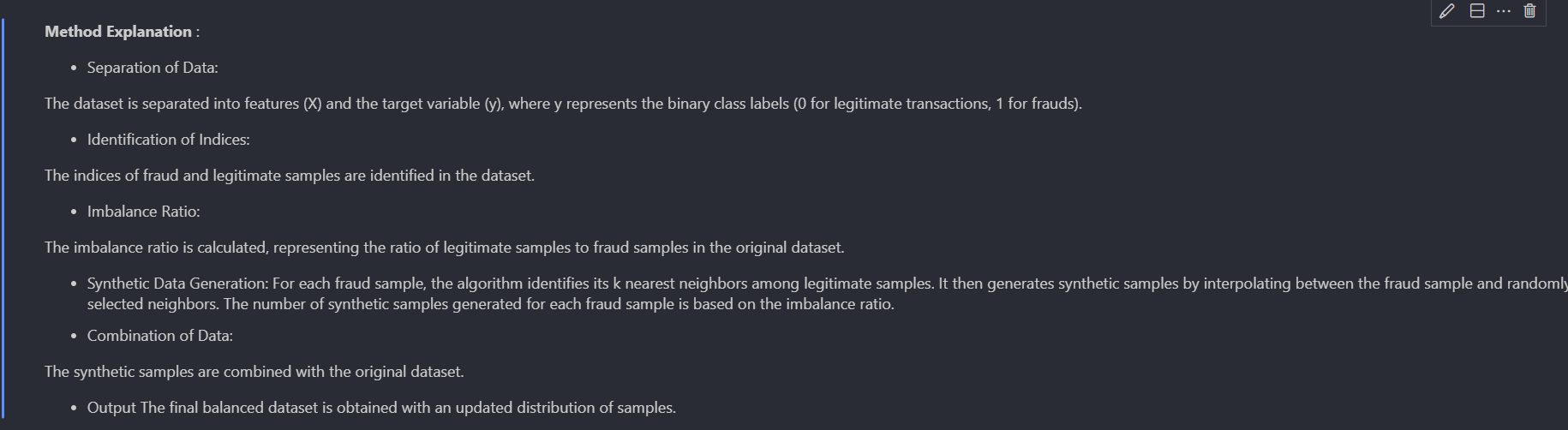
## ۳-۳. نمونه برداري – به كمك Adaptive Synthetic Sampling (ADASYN)

ADASYN



به طور تطبیقی تعداد متفاوتی از نمونه ها را بر اساس تخمینی از توزیع محلی کلاسی که قرار است بیش از حد نمونه برداری شود تولید می کند. در این مورد، طبقه اقلیت. با استفاده از مقادیر پیش‌فرض برای پارامترهای تکنیک نمونه‌گیری ADASYN، کلاس تقلبی به اندازه (284298 نمونه) بیش از حد نمونه‌گیری شد که با کلاس معمولی قابل مقایسه بود. پس از آن، مجموعه داده کامل که از مجموع 568613 نقطه داده تشکیل شده است به سه بخش تقسیم شد. 60 درصد داده های آموزشی، 20 درصد داده های اعتبار سنجی و 20 درصد داده های آزمایشی.

با استفاده از روش train test split دادگان به صورت رندم به 80 20 براي train و test و 25 درصد داده هاي باقي مانده training كه متعاقبا معادل 20 درصد كل ديتاست است برايvalidation در نظر گرفته شد.



توزيح متد:

* جداسازي ديتا

در اين بخش feature (x) و target (y) جداسازي مي‌شودكه y يك باينري دوتايي 0 1 هست.

* مشخص سازي indices

The indices از fraud و legitimate مشخص مي‌شود.

* مشخص كردن Imbalance ratio

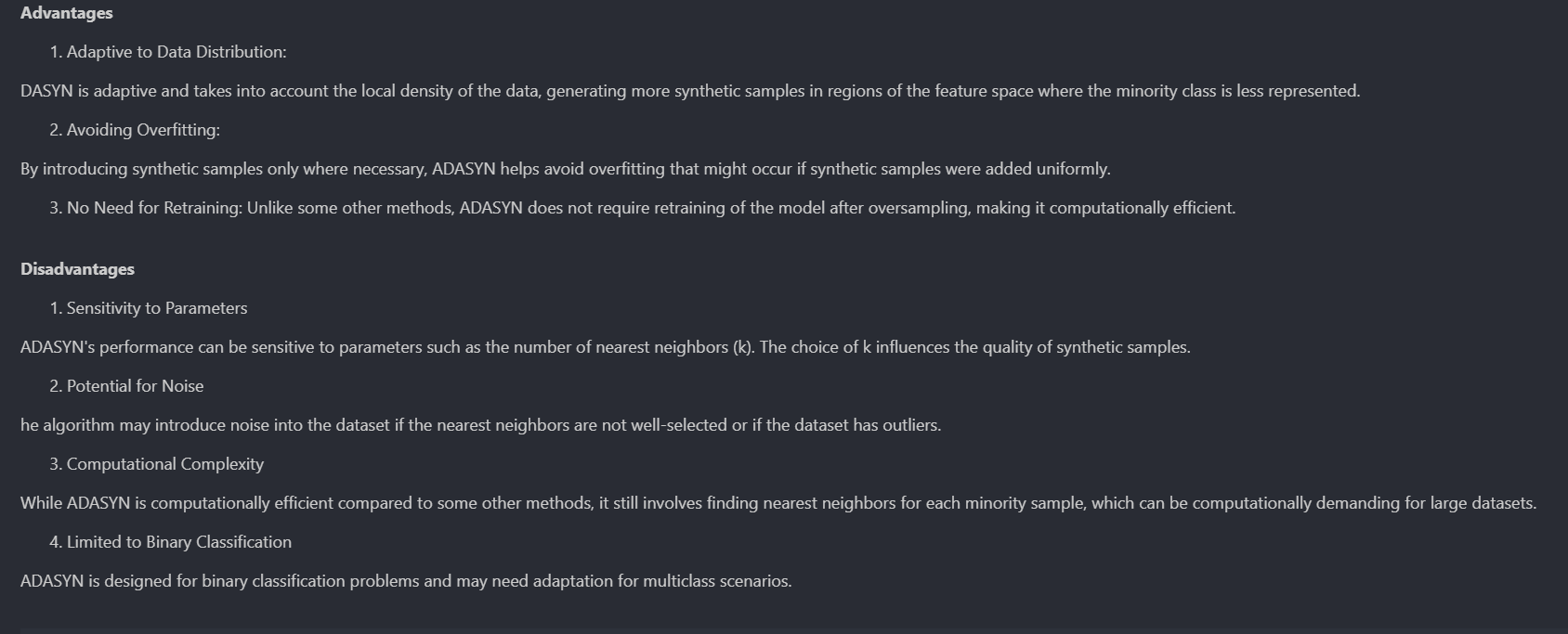
Ratio ناوزني بين دادگان مشخص و حساب مي‌شود.

* ساخت ديتاي مصنوعي

1. برای هر نمونه تقلب، الگوریتم k نزدیکترین همسایه خود را در بین نمونه های قانونی شناسایی می کند.
2. سپس با درون یابی بین نمونه تقلب و همسایگان به طور تصادفی انتخاب شده، نمونه های مصنوعی تولید می کند.

تعداد نمونه های مصنوعی تولید شده برای هر نمونه تقلب بر اساس نسبت عدم تعادل است.

* تركيب ديتا



مزايا و معايب:

مزايا:

1. سازگار با توزیع داده:

DASYN تطبیقی است و چگالی محلی داده ها را در نظر می گیرد و نمونه های مصنوعی بیشتری را در مناطقی از فضای ویژگی ایجاد می کند که در آن کلاس اقلیت کمتر نمایش داده می شود.

2. اجتناب از Overfit:

ADASYN با معرفی نمونه‌های مصنوعی فقط در صورت لزوم، به جلوگیری از برازش بیش از حد که ممکن است در صورت اضافه شدن یکنواخت نمونه‌های مصنوعی رخ دهد، کمک می‌کند.

3. عدم نیاز به آموزش مجدد:

برخلاف برخی روش‌های دیگر، ADASYN نیازی به آموزش مجدد مدل پس از نمونه‌برداری بیش‌ازحد ندارد و آن را از نظر محاسباتی کارآمد می‌کند.

معايب:

\*\*معایب\*\*

1. حساسیت به پارامترها

عملکرد ADASYN می تواند به پارامترهایی مانند تعداد نزدیکترین همسایگان (k) حساس باشد. انتخاب k بر کیفیت نمونه های مصنوعی تأثیر می گذارد.

2. پتانسیل برای نویز

اگر نزدیکترین همسایگان به خوبی انتخاب نشده باشند یا اگر مجموعه داده دارای مقادیر پرت باشد، الگوریتم ممکن است نویز را به مجموعه داده وارد کند.

3. پیچیدگی محاسباتی

در حالی که ADASYN از نظر محاسباتی در مقایسه با برخی روش‌های دیگر کارآمد است، اما همچنان شامل یافتن نزدیک‌ترین همسایگان برای هر نمونه اقلیت است، که می‌تواند از نظر محاسباتی برای مجموعه‌های داده بزرگ نیاز باشد.

4. محدود به طبقه بندی باینری

ADASYN برای مسائل طبقه بندی باینری طراحی شده است و ممکن است برای سناریوهای چند کلاسه نیاز به تطبیق داشته باشد.

*# Separate the dataset into features (X) and the target variable (y)*

X = df.drop('Class', axis=1)

y = df['Class']

*# Identify the indices of fraud and legitimate samples*

fraud\_indices = np.where(y == 1)[0]

legitimate\_indices = np.where(y == 0)[0]

*# Calculate the imbalance ratio*

imbalance\_ratio = len(legitimate\_indices) / len(fraud\_indices)

*# Initialize variables for the synthetic data*

synthetic\_features = []

synthetic\_labels = []

*# Loop over each fraud sample and generate synthetic samples*

for fraud\_index in fraud\_indices:

*# Find the k nearest neighbors of the fraud sample*

    k\_neighbors = np.argsort(np.linalg.norm(X.values[legitimate\_indices] - X.values[fraud\_index], axis=1))[:5]

*# Calculate the number of synthetic samples to generate for the current fraud sample*

    num\_synthetic\_samples = int(imbalance\_ratio) - 1

*#  Generate synthetic samples*

    for \_ in range(num\_synthetic\_samples):

        random\_neighbor\_index = np.random.choice(k\_neighbors)

        synthetic\_sample = X.values[fraud\_index] + np.random.rand() \* (X.values[random\_neighbor\_index] - X.values[fraud\_index])

        synthetic\_features.append(synthetic\_sample)

        synthetic\_labels.append(1)  *# Label for fraud sample*

*#  Combine the synthetic samples with the original dataset*

X\_balanced = np.vstack((X.values, np.array(synthetic\_features)))

y\_balanced = np.concatenate((y.values, np.array(synthetic\_labels)))

*# Print the dataset distribution before and after balancing*

print("TABLE I. DATASET DISTRIBUTION")

print("Dataset \t Legitimate \t Fraud \t\t Total")

print("Before Balancing \t {} \t\t {} \t\t {}".format(len(legitimate\_indices), len(fraud\_indices), len(y)))

print("After Balancing \t {} \t\t {} \t\t {}".format(len(np.where(y\_balanced == 0)[0]), len(np.where(y\_balanced == 1)[0]), len(y\_balanced)))

TABLE I. DATASET DISTRIBUTION

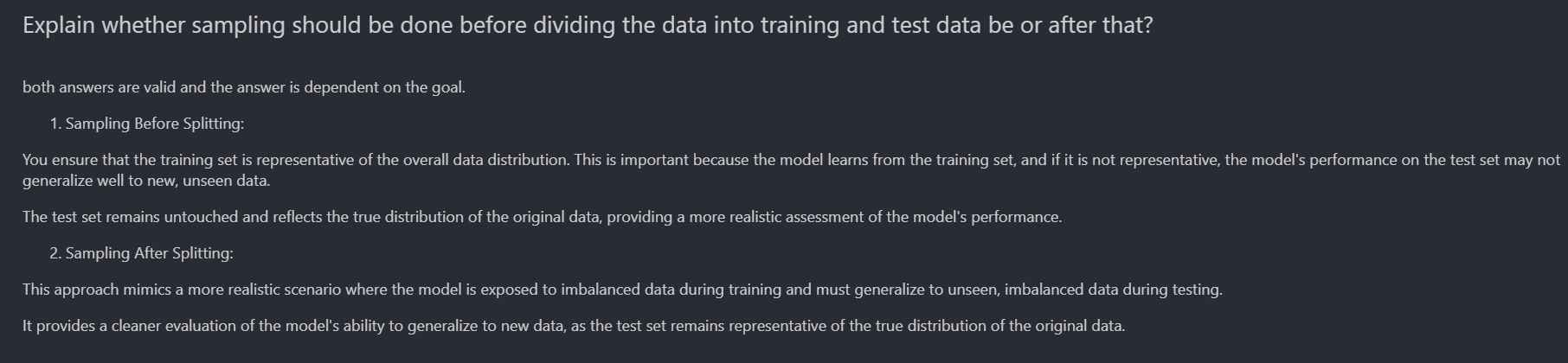
Dataset Legitimate Fraud Total

Before Balancing 284315 492 284807

After Balancing 284315 283884 568199

همان طور كه بالا محاسبه كرديم توزيع داده ها برابر شده است.

### ۳-۳-۳. توزيح دهيد نمونه برداري بايد قبل از تقسيم داده ها به اموزش و تست انجام بشود يا بعد از ان؟



هر دو پاسخ معتبر است و پاسخ به هدف بستگی دارد.

1. نمونه برداری قبل از تقسیم:

شما مطمئن می شوید که مجموعه آموزشی نماینده توزیع کلی داده است. این مهم است زیرا مدل از مجموعه آموزش یاد می گیرد و اگر نماینده نباشد ، عملکرد مدل در مجموعه آزمون ممکن است به خوبی به داده های جدید و غیب تعمیم ندهد.

مجموعه آزمون دست نخورده باقی مانده و توزیع واقعی داده های اصلی را منعکس می کند و ارزیابی واقعی تری از عملکرد مدل ارائه می دهد.

2. نمونه برداری پس از تقسیم:

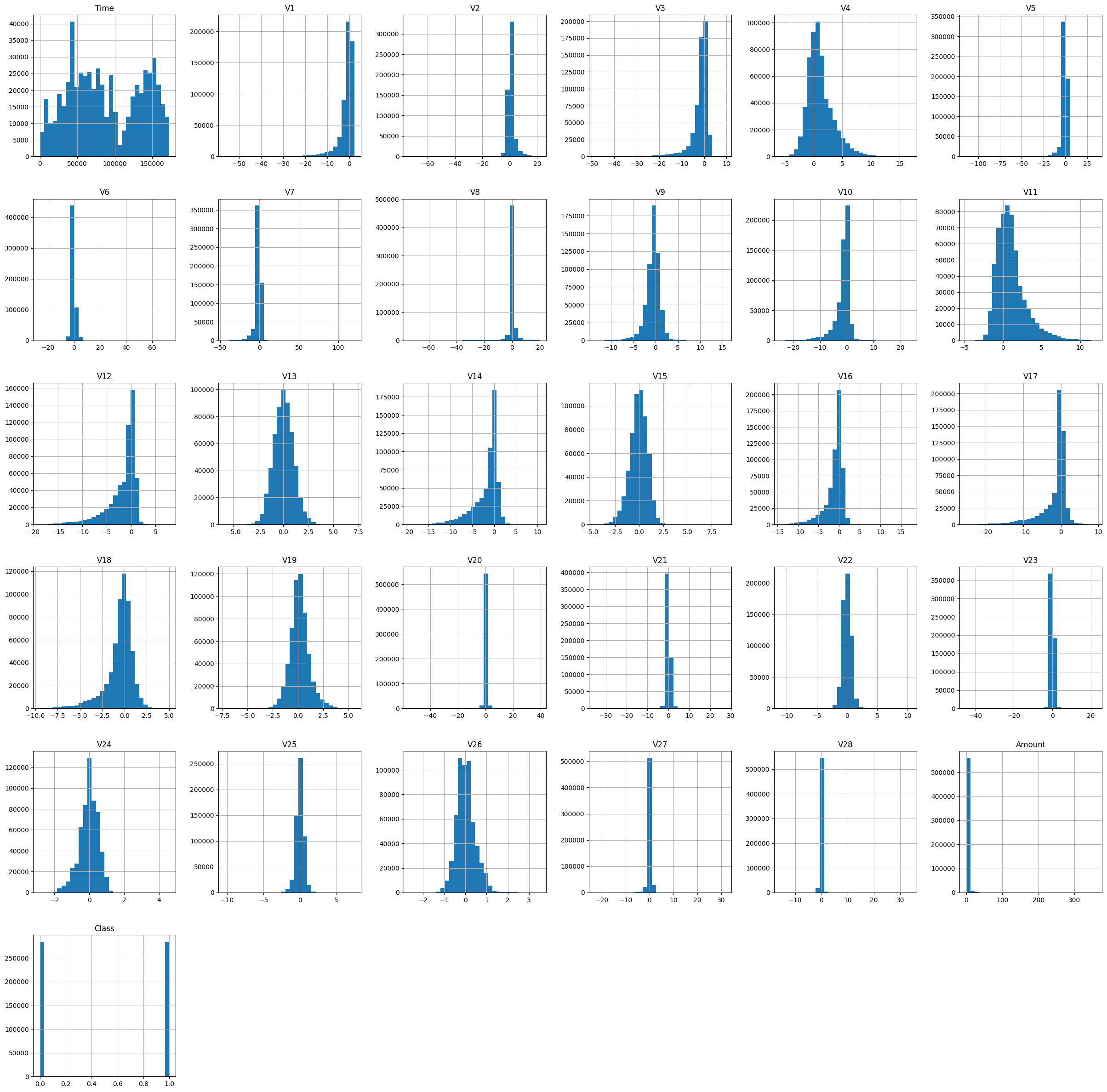
این رویکرد سناریوی واقع بینانه تری را تقلید می کند که در آن مدل در طول آموزش در معرض داده های نامتعادل قرار می گیرد و باید در طول آزمایش به داده های ناشناخته و نابرابر تعمیم یابد.

این یک ارزیابی تمیزتر از توانایی مدل برای تعمیم داده های جدید را ارائه می دهد ، زیرا مجموعه آزمون نماینده توزیع واقعی داده های اصلی است.

df\_balanced = pd.DataFrame(data=np.column\_stack((X\_balanced, y\_balanced)), columns=df.columns)

df\_balanced

df\_balanced.hist(bins=30,figsize=(30,30))



در اينجا اخرين كلاس ميبينيم كه برچسب target توازن دارند.

نمونه برداري و تقسيم بندي:

from sklearn.model\_selection import train\_test\_split

X\_balanced\_reshaped = X\_balanced.reshape(X\_balanced.shape[0], X\_balanced.shape[1], 1)

X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X\_balanced\_reshaped, y\_balanced, test\_size=0.4, random\_state=42)

X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42)

print("Training set shape:", X\_train.shape)

print("Validation set shape:", X\_val.shape)

print("Testing set shape:", X\_test.shape)

Training set shape: (340919, 30, 1)

Validation set shape: (113640, 30, 1)

Testing set shape: (113640, 30, 1)

## ۳-۴. آموزش مدل

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv1D, BatchNormalization, MaxPool1D, Dropout, Flatten, Dense

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.losses import BinaryCrossentropy

from tensorflow.keras.metrics import Accuracy

model = Sequential()

model.add(Conv1D(filters=32, kernel\_size=2, activation='relu', input\_shape=(X\_train.shape[1], 1)))

model.add(BatchNormalization())

model.add(MaxPool1D(pool\_size=2))

model.add(Dropout(rate=0.2))

model.add(Conv1D(filters=64, kernel\_size=2, activation='relu'))

model.add(BatchNormalization())

model.add(MaxPool1D(pool\_size=2))

model.add(Dropout(rate=0.5))

model.add(Flatten())

model.add(Dense(units=64, activation='relu'))

model.add(Dropout(rate=0.5))

model.add(Dense(units=64, activation='relu'))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(optimizer=Adam(learning\_rate=0.0001),

              loss=BinaryCrossentropy(),

              metrics=[Accuracy()])

model.summary()

Model: "sequential\_3"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv1d\_5 (Conv1D) (None, 29, 32) 96

batch\_normalization\_4 (Bat (None, 29, 32) 128

chNormalization)

max\_pooling1d\_4 (MaxPoolin (None, 14, 32) 0

g1D)

dropout\_6 (Dropout) (None, 14, 32) 0

conv1d\_6 (Conv1D) (None, 13, 64) 4160

batch\_normalization\_5 (Bat (None, 13, 64) 256

chNormalization)

max\_pooling1d\_5 (MaxPoolin (None, 6, 64) 0

g1D)

dropout\_7 (Dropout) (None, 6, 64) 0

flatten\_2 (Flatten) (None, 384) 0

...

Total params: 33505 (130.88 KB)

Trainable params: 33313 (130.13 KB)

Non-trainable params: 192 (768.00 Byte)

history = model.fit(X\_train, y\_train, epochs=6, batch\_size=32, validation\_data=(X\_val, y\_val))

Epoch 1/6

10654/10654 [==============================] - 81s 8ms/step - loss: 0.3358 - accuracy: 0.0755 - val\_loss: 0.2760 - val\_accuracy: 0.1125

Epoch 2/6

10654/10654 [==============================] - 80s 7ms/step - loss: 0.2994 - accuracy: 0.1120 - val\_loss: 0.2601 - val\_accuracy: 0.1262

Epoch 3/6

10654/10654 [==============================] - 78s 7ms/step - loss: 0.2838 - accuracy: 0.1293 - val\_loss: 0.2501 - val\_accuracy: 0.1406

Epoch 4/6

10654/10654 [==============================] - 79s 7ms/step - loss: 0.2754 - accuracy: 0.1300 - val\_loss: 0.2392 - val\_accuracy: 0.1291

Epoch 5/6

10654/10654 [==============================] - 76s 7ms/step - loss: 0.2655 - accuracy: 0.1377 - val\_loss: 0.2376 - val\_accuracy: 0.1198

Epoch 6/6

10654/10654 [==============================] - 80s 7ms/step - loss: 0.2613 - accuracy: 0.1306 - val\_loss: 0.2397 - val\_accuracy: 0.0905

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Training and Validation Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Training and Validation Accuracy')

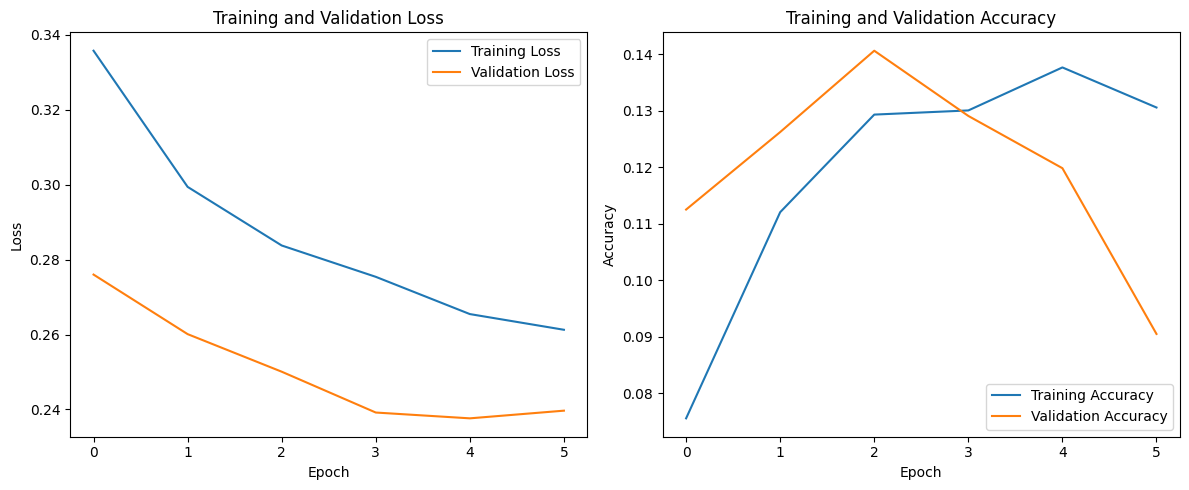
plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.tight\_layout()

plt.show()



from sklearn.metrics import confusion\_matrix, classification\_report

import seaborn as sns

import matplotlib.pyplot as plt

*# Obtain model predictions for the test data*

y\_pred = model.predict(X\_test)

y\_pred\_binary = (y\_pred > 0.5).astype(int)  *# Convert to binary predictions (0 or 1)*

*# Calculate and plot the confusion matrix heatmap*

cm = confusion\_matrix(y\_test, y\_pred\_binary)

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,

            xticklabels=['Legitimate', 'Fraud'],

            yticklabels=['Legitimate', 'Fraud'])

plt.title('Confusion Matrix')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

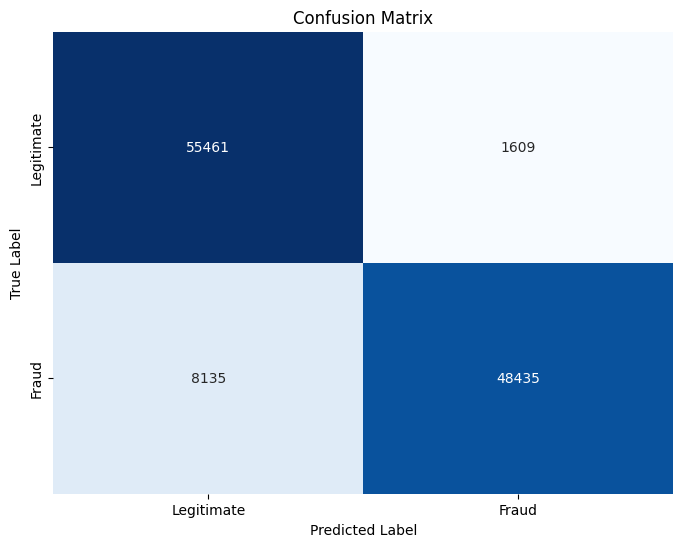
plt.show()

*# Report precision, recall, and F1 score*

report = classification\_report(y\_test, y\_pred\_binary, target\_names=['Legitimate', 'Fraud'])

print("Classification Report:")

print(report)



Classification Report:

precision recall f1-score support

Legitimate 0.87 0.97 0.92 57070

Fraud 0.97 0.86 0.91 56570

accuracy 0.91 113640

macro avg 0.92 0.91 0.91 113640

weighted avg 0.92 0.91 0.91 113640

در اين حالت نتايج دريافت شده قابل اتكا بوده و به دقت 91 درصد مي‌رسيم.