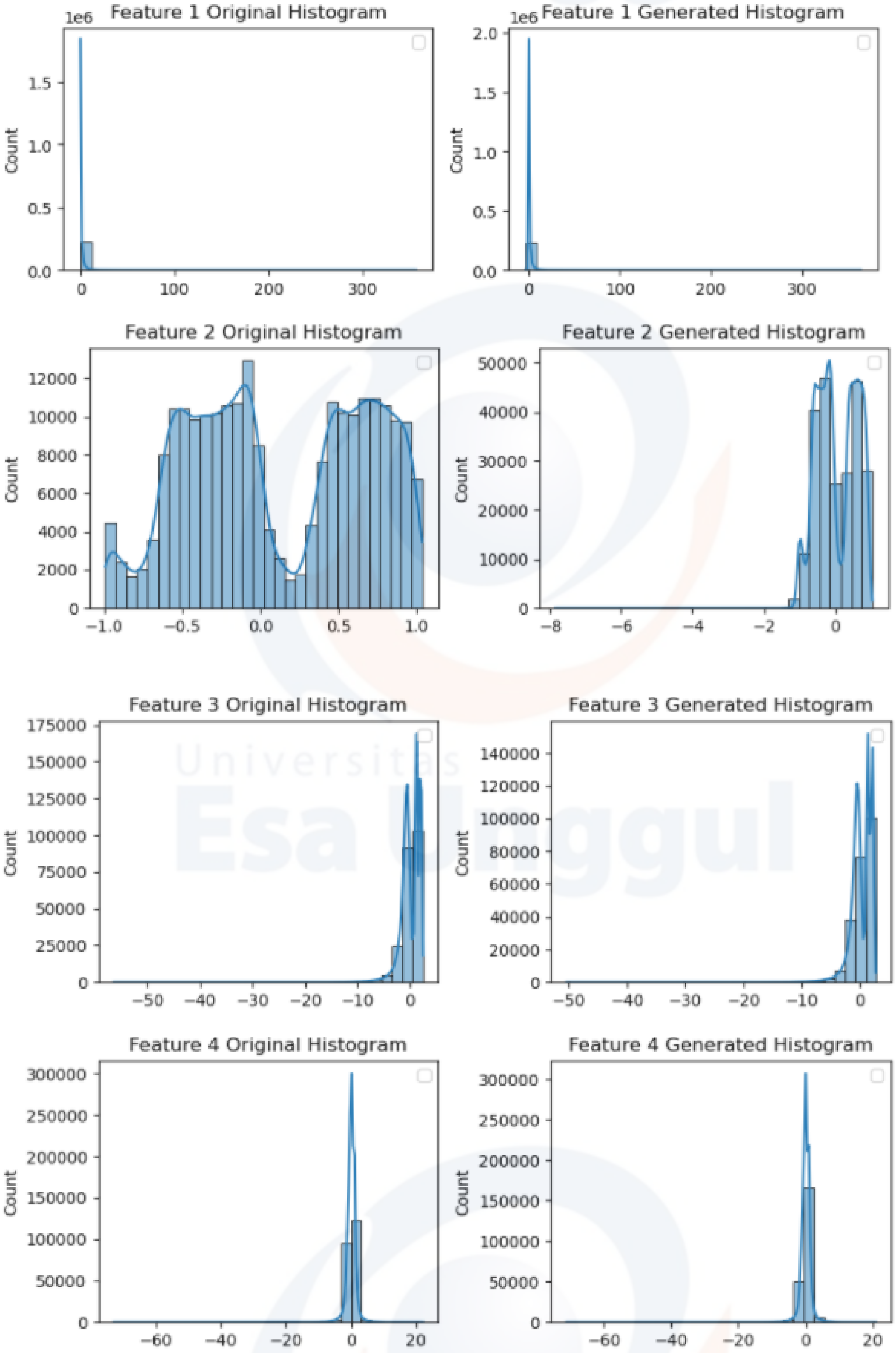
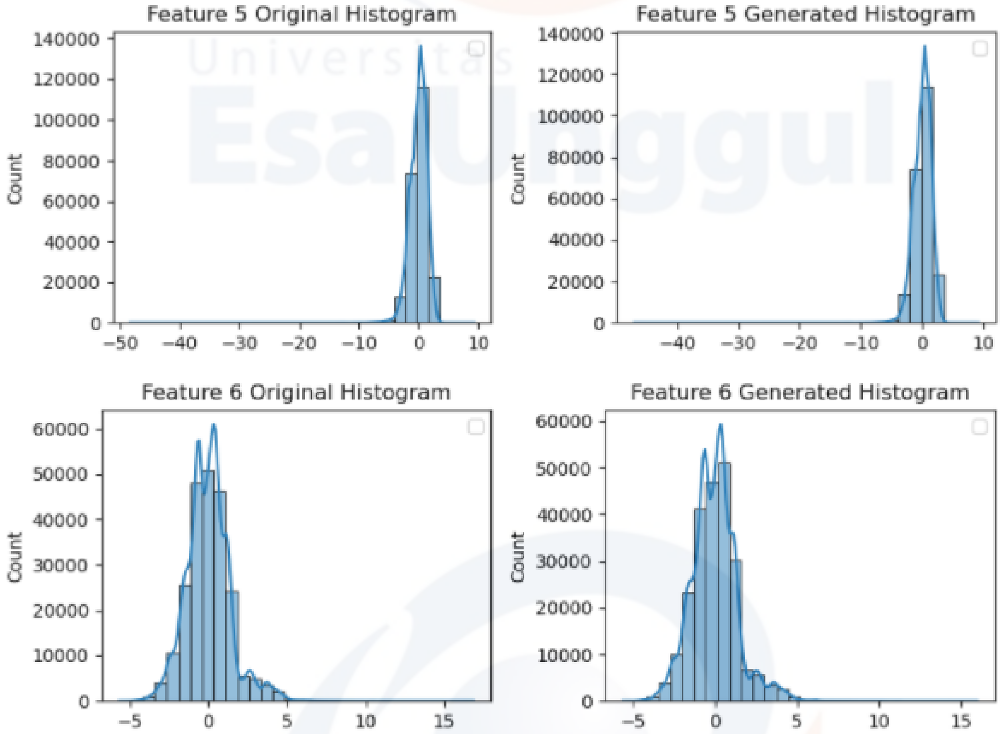
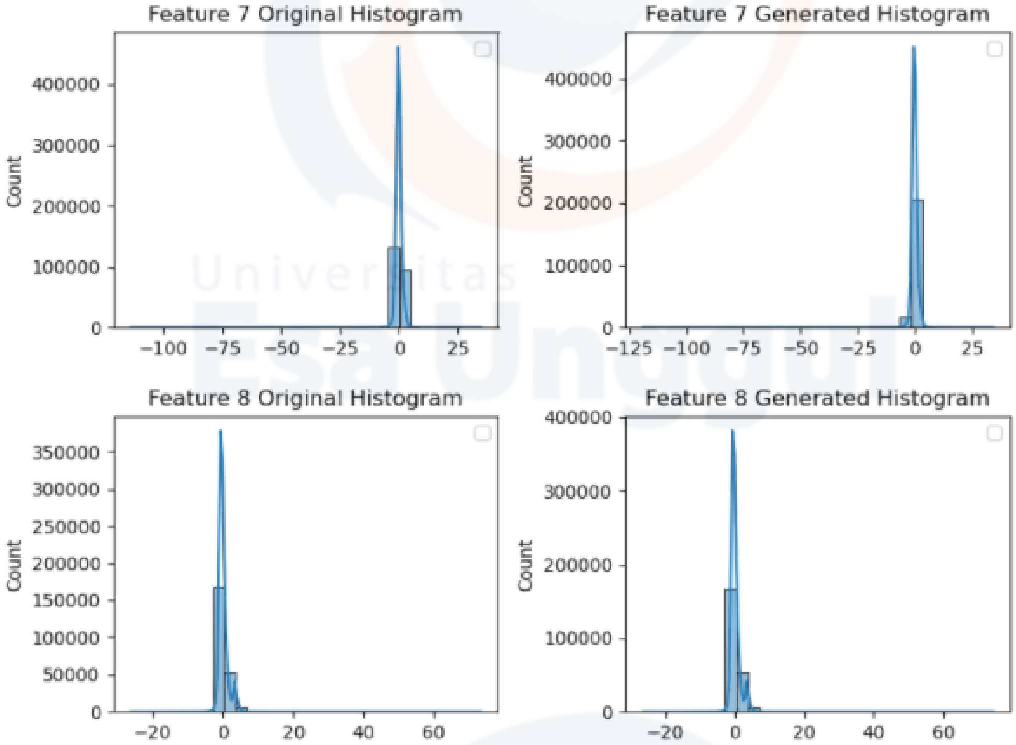
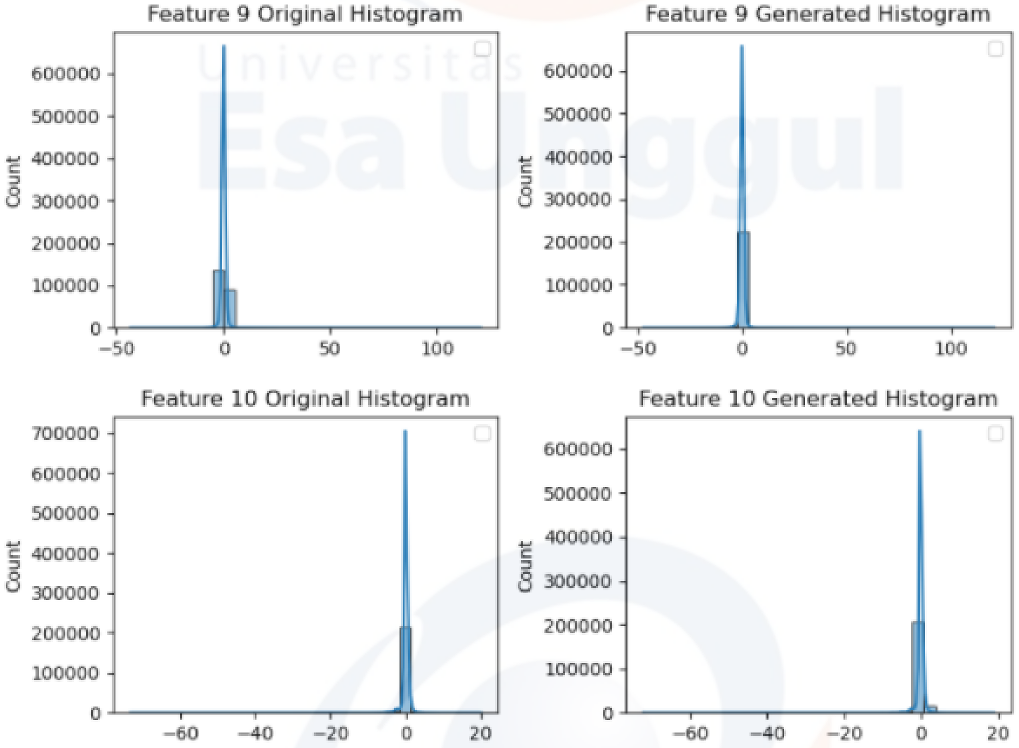
LAMPIRAN A

PERBANDINGAN HISTOGRAM

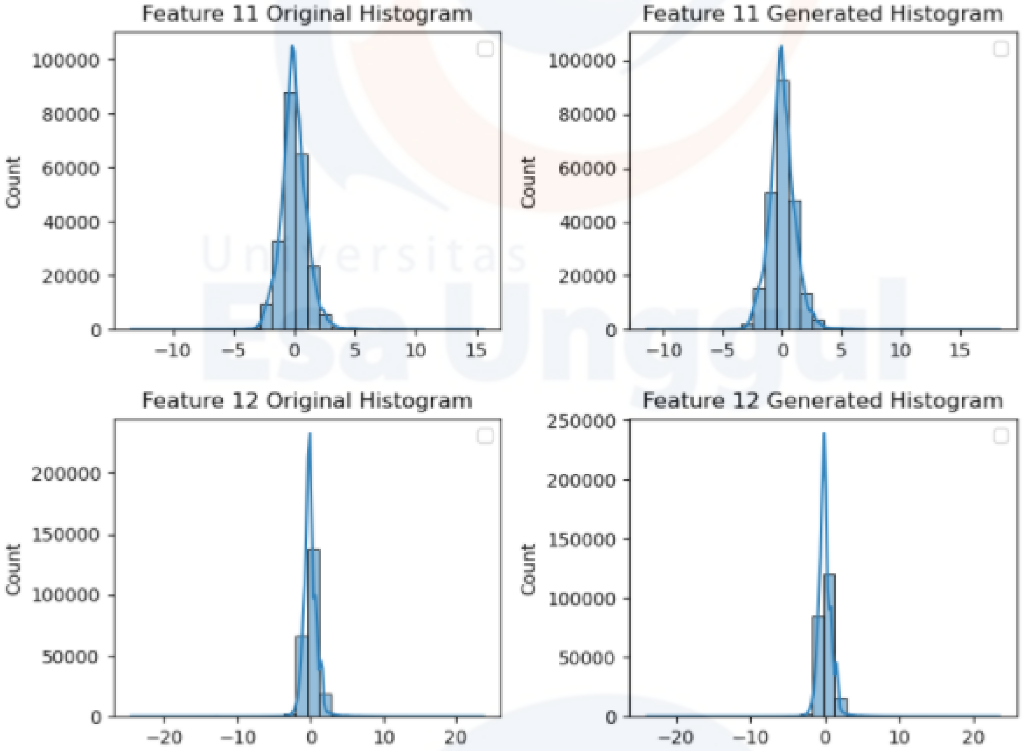



Feature 5 Original Histogram
Count
0
20000
40000
60000
80000
100000
120000
140000
−50
−40
−30
−20
−10
0
10
Feature 5 Generated Histogram
Count
0
20000
40000
60000
80000
100000
120000
140000
−40
−30
−20
−10
0
10
Feature 6 Original Histogram
Count
0
10000
20000
30000
40000
50000
60000
−5
0
5
10
15
Feature 6 Generated Histogram
Count
0
10000
20000
30000
40000
50000
60000
−5
0
5
10
15

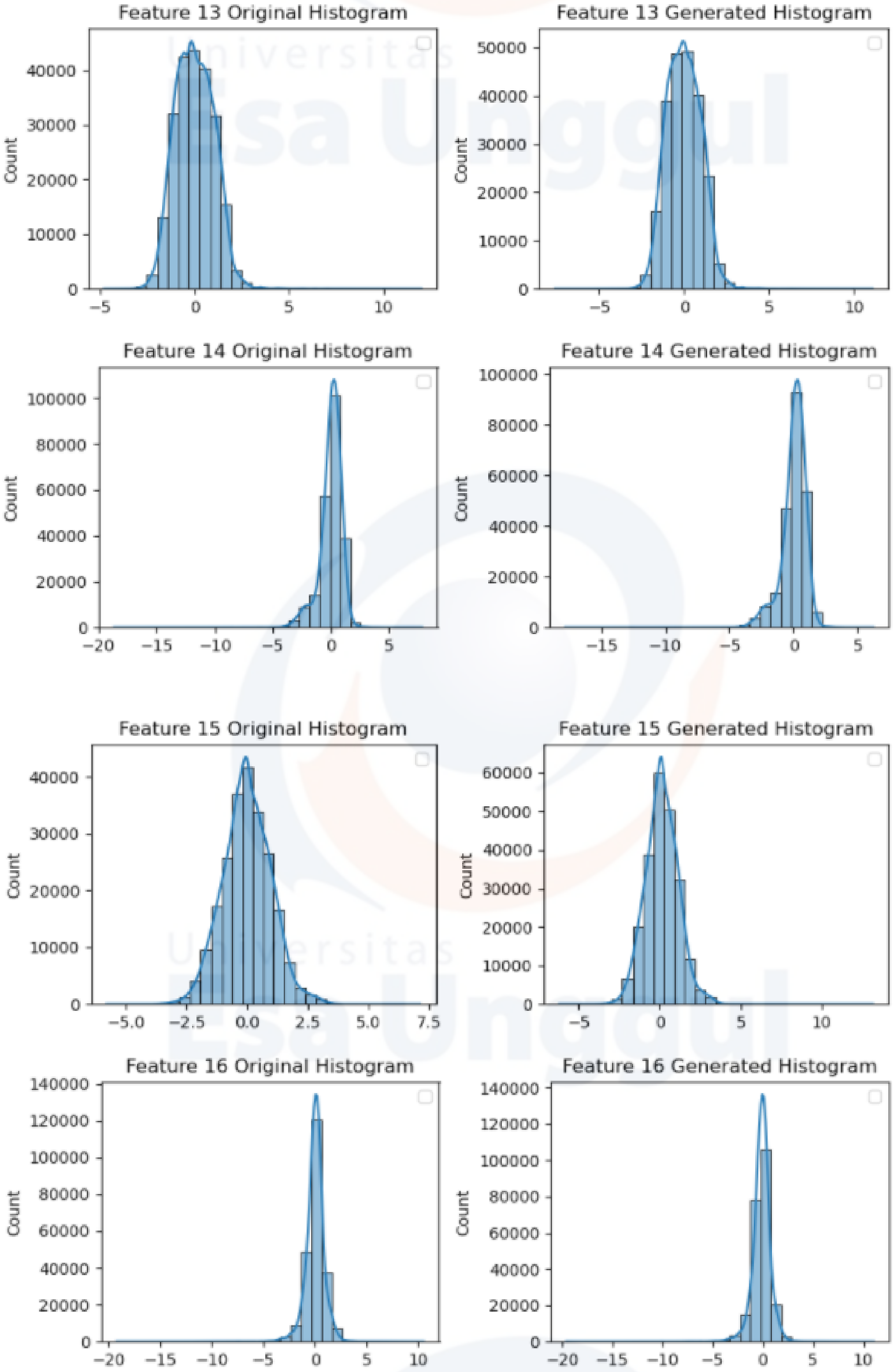

Feature 7 Original Histogram
Count
0
100000
200000
300000
400000
−100
−75
−50
−25
0
25
Feature 7 Generated Histogram
Count
0
100000
200000
300000
400000
−125
−100
−75
−50
−25
0
25
Feature 8 Original Histogram
Count
0
50000
100000
150000
200000
250000
300000
350000
−20
0
20
40
60
Feature 8 Generated Histogram
Count
0
100000
200000
300000
400000
−20
0
20
40
60

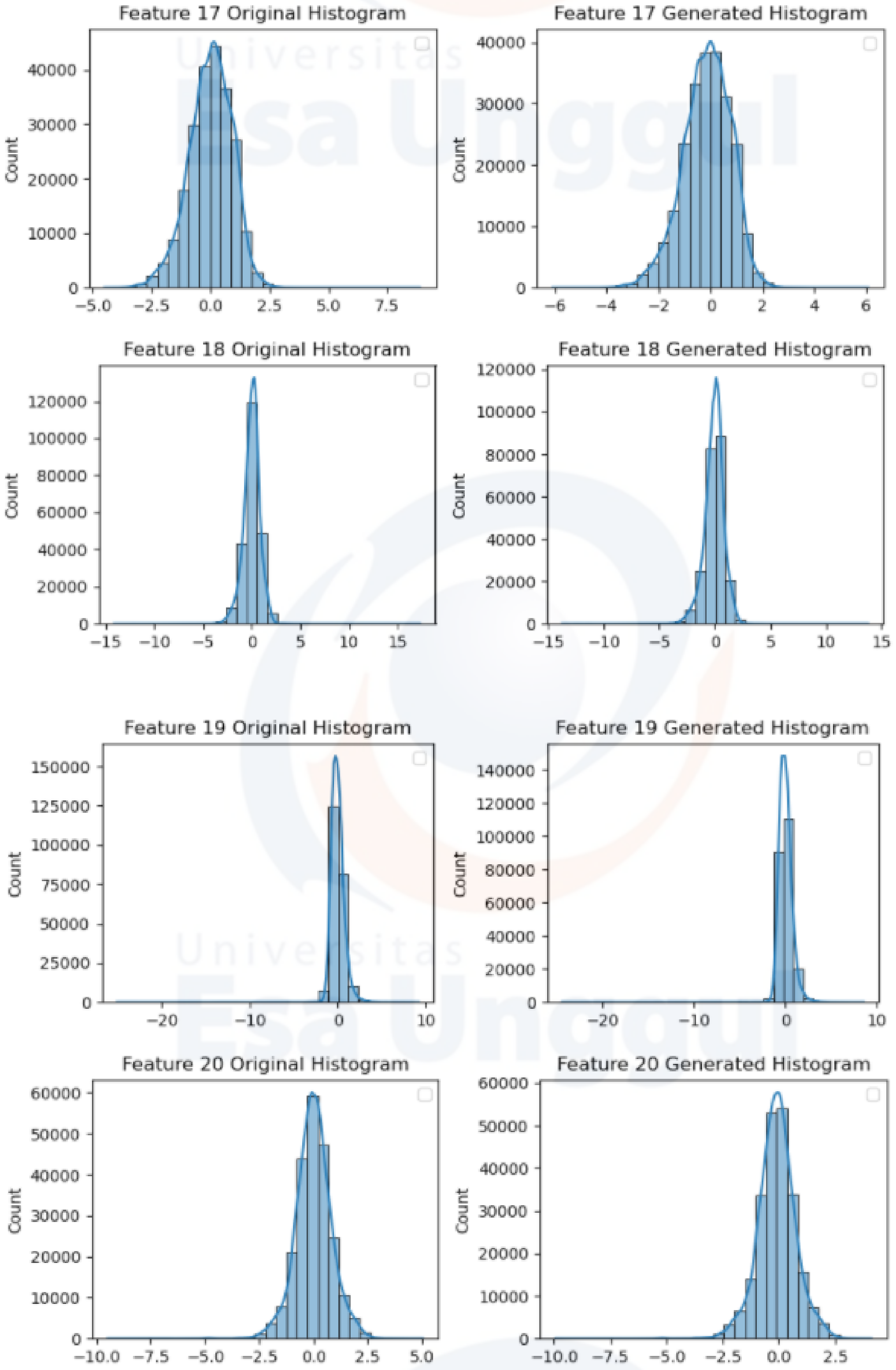


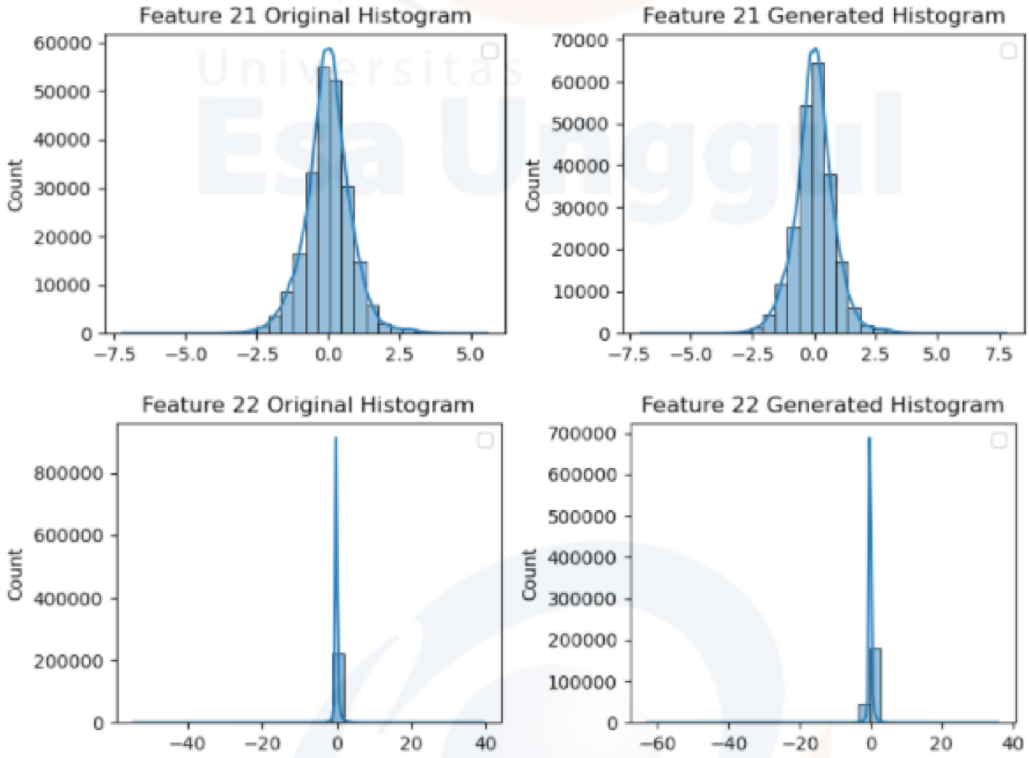
Feature 9 Original Histogram
Feature 9 Generated Histogram
Count
600000
500000
400000
300000
200000
100000
0
−50
0
50
100
Feature 10 Original Histogram
Feature 10 Generated Histogram
700000
−60
−40
−20
20

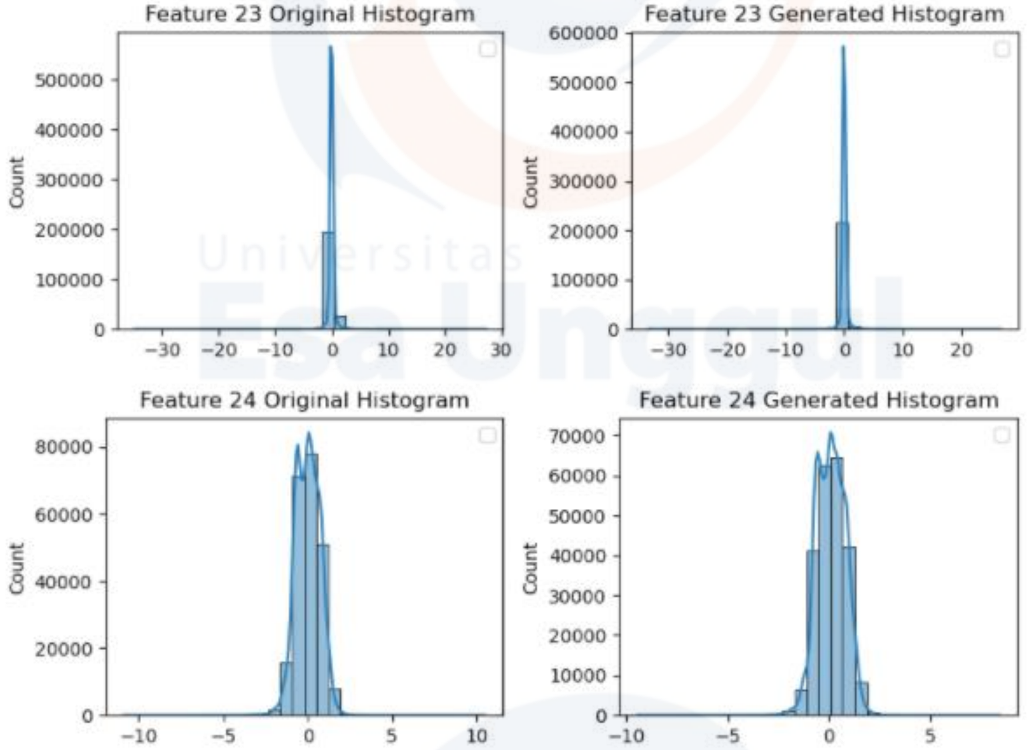


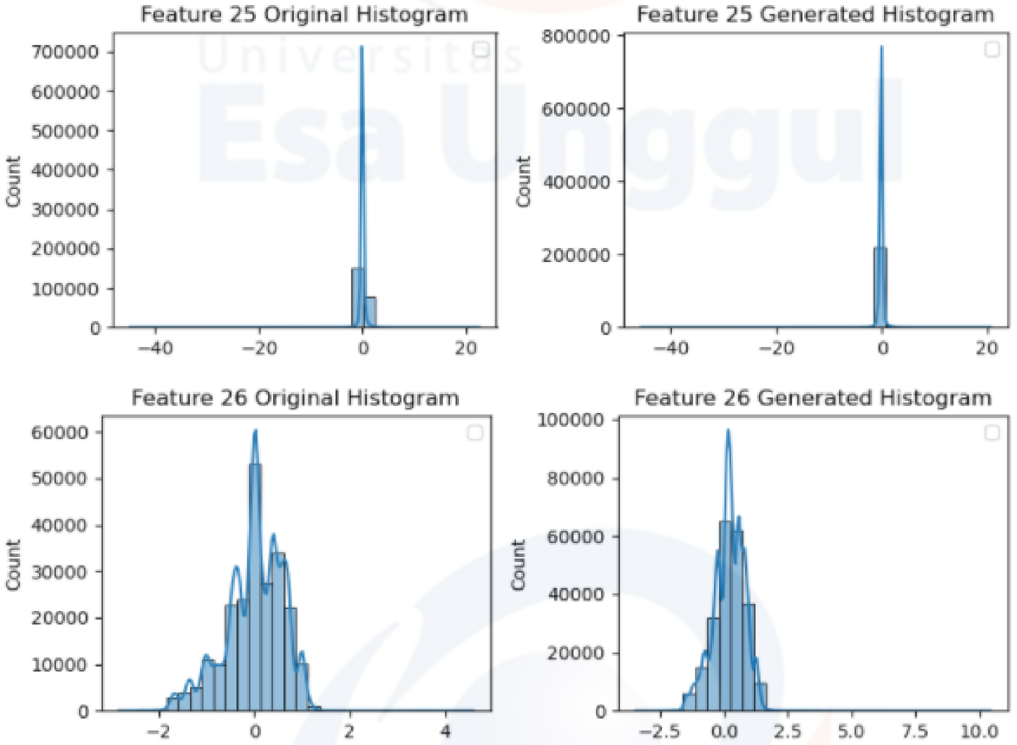
Feature 11 Original Histogram
Feature 11 Generated Histogram
Count
100000
80000
60000
40000
20000
0
−10
−5
0
5
10
15
Feature 12 Original Histogram
Feature 12 Generated Histogram
250000
200000
150000
100000
50000
0
−20
−10
0
10
20

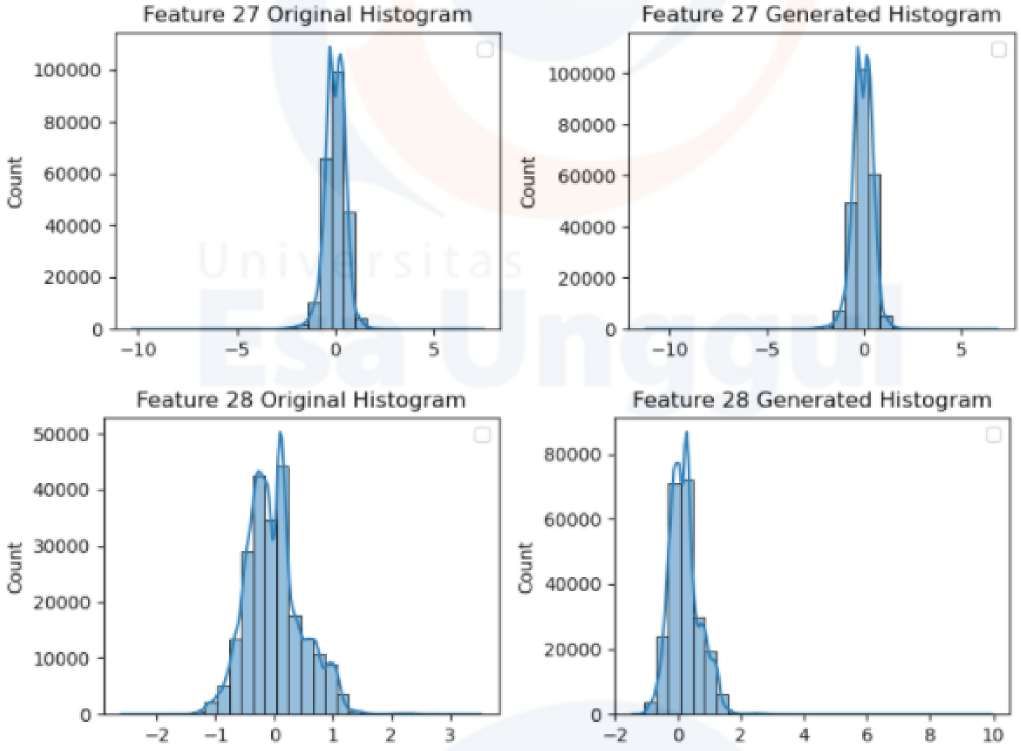

Feature 13 Original Histogram
Count
0
10000
20000
30000
40000
−5
0
5
10
Feature 13 Generated Histogram
Count
0
10000
20000
30000
40000
50000
−5
0
5
10
Feature 14 Original Histogram
Count
0
20000
40000
60000
80000
100000
−20
−15
−10
−5
0
5
Feature 14 Generated Histogram
Count
0
20000
40000
60000
80000
100000
−15
−10
−5
0
5
Feature 15 Original Histogram
Count
0
10000
20000
30000
40000
−5.0
−2.5
0.0
2.5
5.0
7.5
Feature 15 Generated Histogram
Count
0
10000
20000
30000
40000
50000
60000
−5
0
5
10
Feature 16 Original Histogram
Count
0
20000
40000
60000
80000
100000
120000
140000
−20
−15
−10
−5
0
5
10
Feature 16 Generated Histogram
Count
0
20000
40000
60000
80000
100000
120000
140000
−20
−15
−10
−5
0
5
10

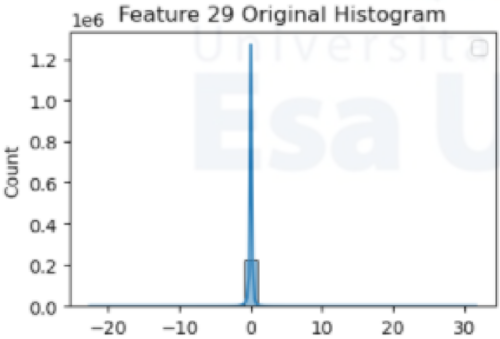

Feature 17 Original Histogram
Count
0
10000
20000
30000
40000
−5.0
−2.5
0.0
2.5
5.0
7.5
Feature 17 Generated Histogram
Count
0
10000
20000
30000
40000
−6
−4
−2
0
2
4
6
Feature 18 Original Histogram
Count
0
20000
40000
60000
80000
100000
120000
−15
−10
−5
0
5
10
15
Feature 18 Generated Histogram
Count
0
20000
40000
60000
80000
100000
120000
−15
−10
−5
0
5
10
15
Feature 19 Original Histogram
Count
0
25000
50000
75000
100000
125000
150000
−20
−10
0
10
Feature 19 Generated Histogram
Count
0
20000
40000
60000
80000
100000
120000
140000
−20
−10
0
10
Feature 20 Original Histogram
Count
0
10000
20000
30000
40000
50000
60000
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
Feature 20 Generated Histogram
Count
0
10000
20000
30000
40000
50000
60000
−10.0
−7.5
−5.0
−2.5
0.0
2.5

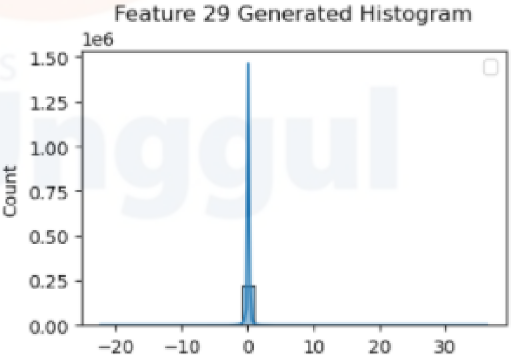

Feature 21 Original Histogram
Count
60000
50000
40000
30000
20000
10000
0
−7.5
−5.0
−2.5
0.0
2.5
5.0
Feature 21 Generated Histogram
Count
70000
60000
50000
40000
30000
20000
10000
0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
Feature 22 Original Histogram
Count
800000
600000
400000
200000
0
−40
−20
0
20
40
Feature 22 Generated Histogram
Count
700000
600000
500000
400000
300000
200000
100000
0
−60
−40
−20
0
20
40

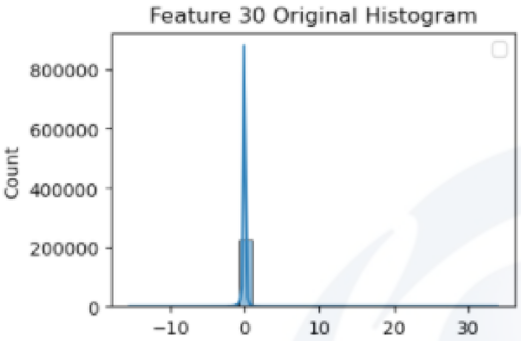

Feature 23 Original Histogram
Count
500000
400000
300000
200000
100000
0
−30
−20
−10
0
10
20
30
Feature 23 Generated Histogram
Count
600000
500000
400000
300000
200000
100000
0
−30
−20
−10
0
10
20
Feature 24 Original Histogram
Count
80000
60000
40000
20000
0
−10
−5
0
5
10
Feature 24 Generated Histogram
Count
70000
60000
50000
40000
30000
20000
10000
0
−10
−5
0
5

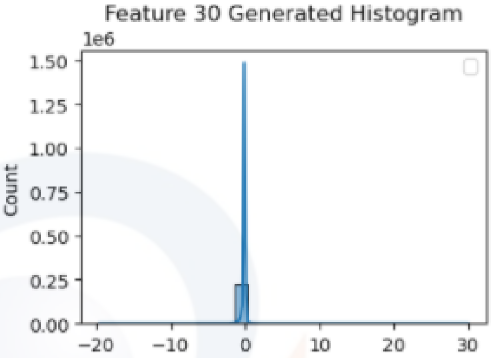

Feature 25 Original Histogram
Count
700000
600000
500000
400000
300000
200000
100000
0
−40
−20
0
20
Feature 25 Generated Histogram
Count
800000
600000
400000
200000
0
−40
−20
0
20
Feature 26 Original Histogram
Count
60000
50000
40000
30000
20000
10000
0
−2
0
2
4
Feature 26 Generated Histogram
Count
100000
80000
60000
40000
20000
0
−2.5
0.0
2.5
5.0
7.5
10.0


Feature 27 Original Histogram
Count
100000
80000
60000
40000
20000
0
−10
−5
0
5
Feature 27 Generated Histogram
Count
100000
80000
60000
40000
20000
0
−10
−5
0
5
Feature 28 Original Histogram
Count
50000
40000
30000
20000
10000
0
−2
−1
0
1
2
3
Feature 28 Generated Histogram
Count
80000
60000
40000
20000
0
−2
0
2
4
6
8
10


Feature 29 Original Histogram
1e6
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Count
−20
−10
0
10
20
30


Feature 29 Generated Histogram
1e6
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Count
−20
−10
0
10
20
30


Feature 30 Original Histogram
800000
600000
400000
200000
0
Count
−10
0
10
20
30


Feature 30 Generated Histogram
1e6
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Count
−20
−10
0
10
20
30

LAMPIRAN B

PEMODELAN MACHINE LEARNING

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Model
from tensorflow.keras.losses import MeanSquaredError, BinaryCrossentropy
from tensorflow.keras.optimizers import Adam
import tensorflow.keras.backend as K
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense, Lambda, concatenate

# Load the credit card dataset (replace 'creditcard.csv' with your dataset file)
df = pd.read_csv('/kaggle/input/creditcardfraud/creditcard.csv')

# Clean the data
df = df.drop_duplicates()
df = df.fillna(0)
df.head()

# Amount and Time are Scaled!

from sklearn.preprocessing import StandardScaler, RobustScaler

# RobustScaler is less prone to outliers.
std_scaler = StandardScaler()
rob_scaler = RobustScaler()

df['scaled_amount'] = rob_scaler.fit_transform(df['Amount'].values.reshape(-1,1))
df['scaled_time'] = rob_scaler.fit_transform(df['Time'].values.reshape(-1,1))
```

```
df.drop(['Time','Amount'], axis=1, inplace=True)

scaled_amount = df['scaled_amount']
scaled_time = df['scaled_time']

df.drop(['scaled_amount', 'scaled_time'], axis=1, inplace=True)
df.insert(0, 'scaled_amount', scaled_amount)
df.insert(1, 'scaled_time', scaled_time)

# Pisahkan data transaksi fraud dan normal
fraud_data = df[df['Class'] == 1]
normal_data = df[df['Class'] == 0]

# Pisahkan fitur dan label
X = df.drop('Class', axis=1)
y = df['Class']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)



# Gunakan seluruh data fraud sebagai data validation
X_fraud = fraud_data.drop('Class', axis=1)
y_fraud = fraud_data['Class']

# Gabungkan data normal train dengan seluruh data fraud sebagai data train
X_train = pd.concat([X_train, X_fraud])
y_train = pd.concat([y_train, y_fraud])

# Shuffle data train
```

```
X_train = X_train.sample(frac=1, random_state=42)
y_train = y_train.sample(frac=1, random_state=42)

from imblearn.over_sampling import SMOTE

# Create SMOTE object
smote = SMOTE(sampling_strategy='auto', random_state=42)

# Resample data
#X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)



# Ubah ke dalam bentuk numpy array
X_train = X_train.values
X_test = X_test.values
y_train = y_train.values
y_test = y_test.values
X_fraud = X_fraud.values
y_fraud = y_fraud.values

#X_train = X_train_resampled.values
#y_train = y_train_resampled.values

import matplotlib.pyplot as plt
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='RdYlGn_r', s=2)
plt.title('test data X_train')
plt.ylabel('mean[1]')
plt.xlabel('mean[0]')
plt.show()

import matplotlib.pyplot as plt
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap='RdYlGn_r', s=2)
plt.title('test data')
plt.ylabel('mean[1]')
plt.xlabel('mean[0]')
plt.show()

# Fungsi untuk membuat encoder dalam Autoencoder
def build_encoder(input_shape, latent_dim):
    inputs = Input(shape=input_shape)
    x = Dense(64, activation='relu')(inputs)
    x = Dense(32, activation='relu')(x)
    z_mean = Dense(latent_dim)(x)
    z_log_var = Dense(latent_dim)(x)
    return Model(inputs, [z_mean, z_log_var])

# Fungsi untuk mengambil sampel dari distribusi normal
def sampling(args):
    z_mean, z_log_var = args
    epsilon = K.random_normal(shape=(K.shape(z_mean)[0], K.shape(z_mean)[1])
)
    return z_mean + K.exp(0.5 * z_log_var) * epsilon

# Fungsi untuk membuat decoder dalam Autoencoder
def build_decoder(input_shape, latent_dim):
    inputs = Input(shape=latent_dim)
    x = Dense(32, activation='relu')(inputs)
    x = Dense(64, activation='relu')(x)
    outputs = Dense(input_shape)(x)
    return Model(inputs, outputs)

# Fungsi untuk membuat discriminator dalam GANs
def build_discriminator(latent_dim):
    inputs = Input(shape=latent_dim)
    x = Dense(32, activation='relu')(inputs)
    x = Dense(1, activation='sigmoid')(x)
    return Model(inputs, x)

from tensorflow.keras.callbacks import EarlyStopping

# Define EarlyStopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Hyperparameter
input_shape = 30  # Jumlah fitur dalam data transaksi
latent_dim = 10  # Jumlah dimensi representasi latent
batch_size = 64
epochs = 500

# Inisialisasi model dan optimizer
encoder = build_encoder(input_shape, latent_dim)
decoder = build_decoder(input_shape, latent_dim)
discriminator = build_discriminator(latent_dim)

autoencoder_input = Input(shape=input_shape)
z_mean, z_log_var = encoder(autoencoder_input)
latent_representation = Lambda(sampling)([z_mean, z_log_var])
reconstructed_data = decoder(latent_representation)

autoencoder = Model(autoencoder_input, reconstructed_data)
autoencoder.compile(optimizer=Adam(), loss=MeanSquaredError())

# Fungsi loss khusus untuk pelatihan discriminator dalam GANs
def discriminator_loss(y_true, y_pred):
    return BinaryCrossentropy()(y_true, y_pred)
```

```
discriminator.compile(optimizer=Adam(), loss=discriminator_loss)

# Pelatihan VAE-GANs
for epoch in range(epochs):
    # Langkah 1: Pelatihan autoencoder (VAE)
    autoencoder.fit(X_train, X_train, batch_size=batch_size, epochs=1,callbacks=[early_stopping])



    # Langkah 2: Pelatihan discriminator (GANs)
    source_representation = encoder.predict(X_train)[0]  # Menggunakan z_mean saja
    target_representation = encoder.predict(X_fraud)[0]  # Menggunakan z_mean saja

    combined_representation = tf.concat([source_representation, target_representation], axis=0)
    labels = tf.concat([tf.ones((len(X_train), 1)), tf.zeros((len(X_fraud), 1))], axis=0)

    discriminator.train_on_batch(combined_representation, labels)

    # Langkah 3: Pelatihan autoencoder dengan adversarial loss
    discriminator.trainable = False
    autoencoder.fit(X_fraud, X_fraud, batch_size=batch_size, epochs=1)
    discriminator.trainable = True

# Calculate histograms for each feature in the original and generated data
original_hists = [np.histogram(X_train[:, i], bins=20, density=True)[0] for i in range(X_train.shape[1])]
generated_samples = loaded_vae_gan.predict(X_train, latent_dim)
```

```
generated_hists = [np.histogram(generated_samples[:, i], bins=20, density=True)[0] for i in range(generated_samples.shape[1])]


import seaborn as sns
from scipy.stats import entropy
# Calculate KL Divergence for each feature
kl_divergences = [entropy(original_hists[i], generated_hists[i]) for i in range(len(original_hists))]

# Plot histograms and KL Divergence values using Seaborn
for i in range(1):
    plt.figure(figsize=(8, 3))

    plt.subplot(1, 2, 1)
    sns.histplot(X_train[:, i], bins=30, kde=True)
    plt.title(f'Feature {i+1} Original Histogram')
    plt.legend()

    plt.subplot(1, 2, 2)
    sns.histplot(generated_samples[:, i], bins=30, kde=True)
    plt.title(f'Feature {i+1} Generated Histogram')
    plt.legend()

    plt.tight_layout()

from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.neural_network import MLPClassifier

# List of parameter values for synthetic samples
parameter_values = [100,500,2000,5000,10000,20000]
for num_samples in parameter_values:
    print("num_samples:",num_samples)
    synthetic_samples = loaded_vae_gan.predict(np.random.normal(size=(num_samples, latent_dim)))
    # Combine synthetic samples with original minority class samples
    X_balanced = np.concatenate([X_train, synthetic_samples])
    y_balanced = np.concatenate([y_train, np.ones(len(synthetic_samples))])
    # Train a Logistic Regression model on the balanced dataset
    lr_model = LogisticRegression()
    lr_model.fit(X_balanced, y_balanced)
    # Evaluate the Logistic Regression model
    y_pred = lr_model.predict(X_test)
    # Evaluasi model
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    print("==================================================")
    print("Training LogisticRegression:")
    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1-score:", f1)
    print("AUC-ROC:", roc_auc)
    print("Classification Report:")
    print(classification_report(y_test, y_pred))
    print("=====================================================
")


    clf = DecisionTreeClassifier(random_state=42)
    # Train the classifier
    clf.fit(X_balanced, y_balanced)
    # Predict on the test data
    y_pred = clf.predict(X_test)
    # Evaluasi model
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    print("=====================================================
")
    print("Training DecisionTreeClassifier:")
    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1-score:", f1)
    print("AUC-ROC:", roc_auc)
    print("Classification Report:")
    print(classification_report(y_test, y_pred))
    print("=======================================================
")

    # Initialize the Random Forest Classifier
    clf = RandomForestClassifier(random_state=42)
    # Train the classifier
    clf.fit(X_balanced, y_balanced)
    # Predict on the test data
    y_pred = clf.predict(X_test)
    # Evaluasi model
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    print("=======================================================
")
    print("Training RandomForestClassifier:")
    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1-score:", f1)
    print("AUC-ROC:", roc_auc)
    print("Classification Report:")
    print(classification_report(y_test, y_pred))
    print("=======================================================
")


    # Initialize the AdaBoost Classifier
    clf = AdaBoostClassifier(random_state=42)
    # Train the classifier
    clf.fit(X_balanced, y_balanced)
    # Predict on the test data
    y_pred = clf.predict(X_test)
    # Evaluasi model
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    print("========================================================
")
    print("Training AdaBoostClassifier:")
    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1-score:", f1)
    print("AUC-ROC:", roc_auc)
    print("Classification Report:")
    print(classification_report(y_test, y_pred))
    print("========================================================
")


    # Initialize the Gradient Boosting Classifier
    clf = GradientBoostingClassifier(random_state=42)
    # Train the classifier
    clf.fit(X_balanced, y_balanced)
    # Predict on the test data
    y_pred = clf.predict(X_test)
    # Evaluasi model
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    print("=========================================================
")
    print("Training GradientBoostingClassifier:")
    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1-score:", f1)
    print("AUC-ROC:", roc_auc)
    print("Classification Report:")
    print(classification_report(y_test, y_pred))
    print("=========================================================
")


    # Initialize the XGBoost Classifier
    clf = XGBClassifier(random_state=42)
    # Train the classifier
    clf.fit(X_balanced, y_balanced)
    # Predict on the test data
    y_pred = clf.predict(X_test)
    # Evaluasi model
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    print("=========================================================
")
    print("Training XGBClassifier:")
```

```
   print("Accuracy:", accuracy)
   print("Precision:", precision)
   print("Recall:", recall)
   print("F1-score:", f1)
   print("AUC-ROC:", roc_auc)
   print("Classification Report:")
   print(classification_report(y_test, y_pred))
   print("==========================================================
")

   clf = MLPClassifier(hidden_layer_sizes=(100,), random_state=42)
   # Train the classifier
   clf.fit(X_balanced, y_balanced)

   # Predict on the test data
   y_pred = clf.predict(X_test)

   # Calculate evaluation metrics
   accuracy = accuracy_score(y_test, y_pred)
   precision = precision_score(y_test, y_pred)
   recall = recall_score(y_test, y_pred)
   roc_auc = roc_auc_score(y_test, y_pred)
   f1 = f1_score(y_test, y_pred)
   print("==========================================================
")
   print("Training MLPClassifier:")
   print("Accuracy:", accuracy)
   print("Precision:", precision)
   print("Recall:", recall)
   print("F1-score:", f1)
   print("AUC-ROC:", roc_auc)
   print("Classification Report:")
print(classification_report(y_test, y_pred))
```