Generation of synthetic tabular data using VAE (Data 3: Berka Dataset)

[https://www.researchgate.net/post/How-to-decide-the-number-of-hidden-layers-and-nodes-in-a-hidden-layer (https://www.researchgate.net/post/How-to-decide-the-number-of-hidden-layers-and-nodes-in-a-hidden-layer)](https://www.researchgate.net/post/How-to-decide-the-number-of-hidden-layers-and-nodes-in-a-hidden-layer)

Data:

In [126]:
```
%reset
```

```
Once deleted, variables cannot be recovered. Proceed (y/[n])? y
```

In [127]:
```
%load_ext watermark
%watermark -p tensorflow,pandas -z -v -n -m -w
```

```
The watermark extension is already loaded. To reload it, use:
  %reload_ext watermark
Python implementation: CPython
Python version       : 3.10.5
IPython version      : 8.4.0

tensorflow: 2.9.1
pandas    : 1.4.2

Compiler    : MSC v.1929 64 bit (AMD64)
OS          : Windows
Release     : 10
Machine     : AMD64
Processor   : Intel64 Family 6 Model 142 Stepping 9, GenuineIntel
CPU cores   : 4
Architecture: 64bit

Watermark: 2.3.1
```

In [128]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [129]:
```
#Calculating the computing time
import time
start = time.time()
import datetime
print("Start Time:" ,datetime.datetime.fromtimestamp(start).strftime('%Y-%m-%d %H:%M:%S'))
```

```
Start Time: 2022-08-08 21:18:43
```

In [130]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch import nn, optim
from torch.autograd import Variable

import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

import seaborn as sns
```

In [131]:
```
pd.set_option("display.max_colwidth", None)
pd.set_option("display.expand_frame_repr", None)
```

In [132]:
```
path = r"C:\Users\Home\Jupyter\Datasets\Original\Berka.csv"
device = torch.device('cpu')
```

In [133]:
```
data_original = pd.read_csv(path, sep =",")
data_original = pd.DataFrame(data_original)
data_original
```

Out[133]:

	trans_id	account_id	date	type	operation	amount	balance	k_symbol	bank	account
0	695247	2378	930101	PRIJEM	VKLAD	700.0	700.0	NaN	NaN	NaN
1	171812	576	930101	PRIJEM	VKLAD	900.0	900.0	NaN	NaN	NaN
2	207264	704	930101	PRIJEM	VKLAD	1000.0	1000.0	NaN	NaN	NaN
3	1117247	3818	930101	PRIJEM	VKLAD	600.0	600.0	NaN	NaN	NaN
4	579373	1972	930102	PRIJEM	VKLAD	400.0	400.0	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...
1048570	1106561	3779	981219	VYDAJ	VYBER	12200.0	59783.7	NaN	NaN	NaN
1048571	1109169	3787	981219	VYDAJ	VYBER	2600.0	81497.4	NaN	NaN	NaN
1048572	1109971	3789	981219	VYBER	VYBER	4900.0	44784.0	NaN	NaN	NaN
1048573	1110516	3791	981219	VYDAJ	VYBER	23500.0	60146.1	NaN	NaN	NaN
1048574	1110008	3789	981219	VYDAJ	VYBER	11100.0	33684.0	NaN	NaN	NaN

1048575 rows × 10 columns

In [134]:
```
data_original.columns
```

Out[134]:
```
Index(['trans_id', 'account_id', 'date', 'type', 'operation', 'amount',
       'balance', 'k_symbol', 'bank', 'account'],
      dtype='object')
```

In [135]:
```
data_original.dropna(axis =0, inplace = True)
data_original.drop_duplicates(inplace =True)
data_original.shape
data_original
```

Out[135]:

	trans_id	account_id	date	type	operation	amount	balance	k_symbol	bank	account
15	637742	2177	930105	PRIJEM	PREVOD Z UCTU	5123.0	5923.0	DUCHOD	YZ	62457513.0
24	579374	1972	930107	PRIJEM	PREVOD Z UCTU	5298.0	5698.0	DUCHOD	UV	14132887.0
46	1049882	3592	930110	PRIJEM	PREVOD Z UCTU	6007.0	6607.0	DUCHOD	MN	73166322.0
49	171813	576	930111	PRIJEM	PREVOD Z UCTU	6207.0	7107.0	DUCHOD	YZ	30300313.0
53	689828	2357	930112	PRIJEM	PREVOD Z UCTU	6434.0	7234.0	DUCHOD	OP	34144538.0
...	...	...	...	...	...	...	...	...	...	...
1047131	2592237	8564	981214	VYDAJ	PREVOD NA UCET	7170.0	94170.1	SIPO	UV	59670215.0
1047135	492988	1681	981214	VYDAJ	PREVOD NA UCET	8550.0	60559.8	SIPO	UV	8172750.0
1047136	489972	1672	981214	VYDAJ	PREVOD NA UCET	3898.0	52169.8	SIPO	ST	9213483.0
1047137	519257	1773	981214	PRIJEM	PREVOD Z UCTU	4316.0	17215.9	DUCHOD	CD	77385341.0
1047140	517067	1767	981214	VYDAJ	PREVOD NA UCET	2266.0	73784.4	SIPO	ST	54714965.0

230465 rows × 10 columns

In [136]:
```
encoded_data = data_original.copy()

catergorical_columns = ['type', 'operation', 'k_symbol', 'bank']
encoded = "_encoded"
for column in catergorical_columns:
    print("Applied dummies for:",column)
    encoded_data[column] = encoded_data[column].astype('category')
    encoded_data.dtypes
    column_encoded = column + encoded
    encoded_data[column_encoded] = encoded_data[column].cat.codes.astype(np.int64)
    encoded_data.drop( labels=column, axis=1, inplace =True)
#     catergorical_columns_encodings  = pd.get_dummies(df_get_dummies[column], prefix=column )
#     df_get_dummies = pd.concat([df_get_dummies, catergorical_columns_encodings.astype(np.int64)], axis=1)
#     df_get_dummies.drop(labels=column, axis= 1, inplace = True)
data_original =encoded_data
data_original
```

```
Applied dummies for: type
Applied dummies for: operation
Applied dummies for: k_symbol
Applied dummies for: bank
```

Out[136]:

	trans_id	account_id	date	amount	balance	account	type_encoded	operation_encoded	k_symbol_encoded	bank_encoded
15	637742	2177	930105	5123.0	5923.0	62457513.0	0	1	1	12
24	579374	1972	930107	5298.0	5698.0	14132887.0	0	1	1	10
46	1049882	3592	930110	6007.0	6607.0	73166322.0	0	1	1	6
49	171813	576	930111	6207.0	7107.0	30300313.0	0	1	1	12
53	689828	2357	930112	6434.0	7234.0	34144538.0	0	1	1	7
...	...	...	...	...	...	...	...	...	...	...
1047131	2592237	8564	981214	7170.0	94170.1	59670215.0	1	0	3	10
1047135	492988	1681	981214	8550.0	60559.8	8172750.0	1	0	3	10
1047136	489972	1672	981214	3898.0	52169.8	9213483.0	1	0	3	9

In [137]:
```
data_original.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 230465 entries, 15 to 1047140
Data columns (total 10 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   trans_id           230465 non-null  int64  
 1   account_id         230465 non-null  int64  
 2   date               230465 non-null  int64  
 3   amount             230465 non-null  float64
 4   balance            230465 non-null  float64
 5   account            230465 non-null  float64
 6   type_encoded       230465 non-null  int64  
 7   operation_encoded  230465 non-null  int64  
 8   k_symbol_encoded   230465 non-null  int64  
 9   bank_encoded       230465 non-null  int64  
dtypes: float64(3), int64(7)
memory usage: 19.3 MB
```

In [138]:
```
data_original.bank_encoded.info()
```

```
<class 'pandas.core.series.Series'>
Int64Index: 230465 entries, 15 to 1047140
Series name: bank_encoded
Non-Null Count   Dtype
--------------   -----
230465 non-null  int64
dtypes: int64(1)
memory usage: 3.5 MB
```

```
columns_list =df_get_dummies.columns
columns_list = np.array(columns_list,dtype = 'str')
#columns_list
data_original = pd.DataFrame(df_get_dummies)
# for columns in columns_list:
#     data_original = data_original.astype(np.int64)    # ({columns:'np.int64'})
#    data_original.columns =data_original.columns.astype(np.int64)

data_original = data_original.astype(np.int64)
print(data_original.dtypes)
```

In [139]:
```
data_original.to_csv('data3(Berka)_original_with_Dummies.csv')
```

In [140]:
```
columns = data_original.columns
data_original.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 230465 entries, 15 to 1047140
Data columns (total 10 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   trans_id           230465 non-null  int64  
 1   account_id         230465 non-null  int64  
 2   date               230465 non-null  int64  
 3   amount             230465 non-null  float64
 4   balance            230465 non-null  float64
 5   account            230465 non-null  float64
 6   type_encoded       230465 non-null  int64  
 7   operation_encoded  230465 non-null  int64  
 8   k_symbol_encoded   230465 non-null  int64  
 9   bank_encoded       230465 non-null  int64  
dtypes: float64(3), int64(7)
memory usage: 19.3 MB
```

Build Data Loader

In [141]:
```
def load_and_standardize_data(path):
    df = data_original
#     df = pd.read_csv(path) # read in from csv
    #del df['Time'] #Only for Data 1, Del the col
    df = df.values.reshape(-1, df.shape[1]).astype('float32')
    X_train, X_test = train_test_split(df, test_size=0.3, random_state=100) # randomly split
    # Standardize features by removing the mean and scaling to unit variance. z = (x - u) / s
    scaler = preprocessing.StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
    return X_train, X_test, scaler
```

In [142]:
```
from torch.utils.data import Dataset, DataLoader
class DataBuilder(Dataset):
    def __init__(self, path, train=True):
        self.X_train, self.X_test, self.standardizer = load_and_standardize_data(path)
        if train:
            self.x = torch.from_numpy(self.X_train)
            self.len=self.x.shape[0]
        else:
            self.x = torch.from_numpy(self.X_test)
            self.len=self.x.shape[0]
        del self.X_train
        del self.X_test
    def __getitem__(self,index):
        return self.x[index]
    def __len__(self):
        return self.len
```

In [143]:
```
#Dataset
data_set=DataBuilder(path)
traindata_set=DataBuilder(path, train=True)
testdata_set=DataBuilder(path, train=False)
#Loader
trainloader=DataLoader(dataset=traindata_set,batch_size=1024)#1024--> original D1
testloader=DataLoader(dataset=testdata_set,batch_size=1024)#1024--> original D1
```

In [144]:
```
type(trainloader.dataset.x), type(testloader.dataset.x)
```

Out[144]:
```
(torch.Tensor, torch.Tensor)
```

In [145]:
```
trainloader.dataset.x.shape, testloader.dataset.x.shape
```

Out[145]:
```
(torch.Size([161325, 10]), torch.Size([69140, 10]))
```

In [146]:
```
trainloader.dataset.x
```

Out[146]:
```
tensor([[-0.1550, -0.1457,  1.0242,  ...,  2.5618, -0.7689,  0.2575],
        [-0.3229, -0.3235,  0.2739,  ..., -0.3903,  0.7289,  0.7910],
        [-0.5705, -0.5767,  1.0020,  ..., -0.3903,  0.7289,  0.7910],
        ...,
        [ 0.3052,  0.3242,  1.0614,  ..., -0.3903, -1.5179, -0.8095],
        [-0.6042, -0.6123, -0.4618,  ...,  2.5618, -0.7689,  0.5242],
        [-0.2309, -0.2250, -1.1604,  ..., -0.3903,  0.7289, -0.0092]])
```

VAE Model

In [147]:
```
class Autoencoder(nn.Module):
    def __init__(self,D_in,H=50,H2=12,latent_dim=3):

        #Encoder
        super(Autoencoder,self).__init__()
        self.linear1=nn.Linear(D_in,H)
        self.lin_bn1 = nn.BatchNorm1d(num_features=H)
        self.linear2=nn.Linear(H,H2)
        self.lin_bn2 = nn.BatchNorm1d(num_features=H2)
        self.linear3=nn.Linear(H2,H2)
        self.lin_bn3 = nn.BatchNorm1d(num_features=H2)

        # Latent vectors mu and sigma
        self.fc1 = nn.Linear(H2, latent_dim)
        self.bn1 = nn.BatchNorm1d(num_features=latent_dim)
        self.fc21 = nn.Linear(latent_dim, latent_dim)
        self.fc22 = nn.Linear(latent_dim, latent_dim)

        # Sampling vector
        self.fc3 = nn.Linear(latent_dim, latent_dim)
        self.fc_bn3 = nn.BatchNorm1d(latent_dim)
        self.fc4 = nn.Linear(latent_dim, H2)
        self.fc_bn4 = nn.BatchNorm1d(H2)

        # Decoder
        self.linear4=nn.Linear(H2,H2)
        self.lin_bn4 = nn.BatchNorm1d(num_features=H2)
        self.linear5=nn.Linear(H2,H)
        self.lin_bn5 = nn.BatchNorm1d(num_features=H)
        self.linear6=nn.Linear(H,D_in)
        self.lin_bn6 = nn.BatchNorm1d(num_features=D_in)

        self.relu = nn.ReLU()

    def encode(self, x):
        lin1 = self.relu(self.lin_bn1(self.linear1(x)))
        lin2 = self.relu(self.lin_bn2(self.linear2(lin1)))
        lin3 = self.relu(self.lin_bn3(self.linear3(lin2)))

        fc1 = F.relu(self.bn1(self.fc1(lin3)))

        r1 = self.fc21(fc1)
        r2 = self.fc22(fc1)

        return r1, r2

    def decode(self, z):
        fc3 = self.relu(self.fc_bn3(self.fc3(z)))
        fc4 = self.relu(self.fc_bn4(self.fc4(fc3)))

        lin4 = self.relu(self.lin_bn4(self.linear4(fc4)))
        lin5 = self.relu(self.lin_bn5(self.linear5(lin4)))
        return self.lin_bn6(self.linear6(lin5))

    def reparameterize(self, mu, logvar):
        if self.training:
            std = logvar.mul(0.5).exp_()
            eps = Variable(std.data.new(std.size()).normal_())
            return eps.mul(std).add_(mu)
        else:
            return mu

    def forward(self, x):
        mu, logvar = self.encode(x)
        z = self.reparameterize(mu, logvar)
        return self.decode(z), mu, logvar
```

In [148]:
```
class customLoss(nn.Module):
    def __init__(self):
        super(customLoss, self).__init__()
        self.mse_loss = nn.MSELoss(reduction="sum")

    def forward(self, x_recon, x, mu, logvar):
        loss_MSE = self.mse_loss(x_recon, x)
        loss_KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp()) #Kullback-Leibler Divergence (KL-divergence)
        return loss_MSE + loss_KLD
#the KL loss is equivalent to the sum of all the KL divergences between the component Xi~N(μi, σi²) in X, and the standard normal[
```

In [149]:
```
D_in = data_set.x.shape[1]
H = 50 #layer 1
H2 = 12 # Layer 2
model = Autoencoder(D_in, H, H2).to(device)
optimizer = optim.Adam(model.parameters(), lr=1e-3) # LR = 0.001
loss_mse = customLoss()
```

In [150]:
```
epochs = 2000
log_interval = 50
val_losses = []
train_losses = []
test_losses = []
```

In [151]:
```
epoch_Data =[]
test_loss_Data =[]
train_loss_Data =[]
average_test_loss_Data =[]
average_train_loss_Data =[]
test_time_Data = []
train_time_Data =[]
```

Notes: When the Bernoulli distribution is modeled, the MSE or the binary cross-entropy could be used. When a normal distribution is modeled, the log-likelihood is often applied.

In [152]:
```
def train(epoch):
    model.train()
    train_loss = 0
    for batch_idx, data in enumerate(trainloader):
        data = data.to(device)
        optimizer.zero_grad()
        reconstructed_data, mu, logvar = model(data)
        loss = loss_mse(reconstructed_data, data, mu, logvar)
        loss.backward()
        train_loss += loss.item()
        optimizer.step()
        ###############
        average_train_loss =train_loss / len(trainloader.dataset)#New
        epoch_Data.append(epoch)#New
        train_loss_Data.append(train_loss)#New
        average_train_loss_Data.append(train_loss / len(trainloader.dataset))#New
        train_time_Data.append(time.time())
        ###############
    if epoch % 100 == 0:
        print('train=> Epoch: {} Average training loss: {:.4f}'.format(
            epoch, train_loss / len(trainloader.dataset)))
        train_losses.append(train_loss / len(trainloader.dataset))
```

In [153]:
```
# Changes in Bloc
def test(epoch):
    with torch.no_grad():
        test_loss = 0
        for batch_idx, data in enumerate(testloader):
            data = data.to(device)
            optimizer.zero_grad()
            reconstructed_data, mu, logvar = model(data)
            loss = loss_mse(reconstructed_data, data, mu, logvar)
            test_loss += loss.item()
            ##############
            average_test_loss =test_loss / len(testloader.dataset)#New
            epoch_Data.append(epoch)#New
            test_loss_Data.append(test_loss)#New
            average_test_loss_Data.append(test_loss / len(testloader.dataset))#New
            test_time_Data.append(time.time())
            ##############
        if epoch % 100 == 0:
            print('test=> Epoch: {} Average test loss: {:.4f}'.format(
                    epoch, test_loss / len(testloader.dataset)))
            test_losses.append(test_loss / len(testloader.dataset))
```

In [154]:
```
train_start_time =time.time()
for epoch in range(1, epochs + 1):
    train(epoch)
    test(epoch)
train_end_time =time.time()
```

```
train=> Epoch: 100 Average training loss: 6.2663
test=> Epoch: 100 Average test loss: 6.2546
train=> Epoch: 200 Average training loss: 6.2105
test=> Epoch: 200 Average test loss: 6.2042
train=> Epoch: 300 Average training loss: 6.1875
test=> Epoch: 300 Average test loss: 6.1763
train=> Epoch: 400 Average training loss: 6.1714
test=> Epoch: 400 Average test loss: 6.1734
train=> Epoch: 500 Average training loss: 6.1685
test=> Epoch: 500 Average test loss: 6.1560
train=> Epoch: 600 Average training loss: 6.1567
test=> Epoch: 600 Average test loss: 6.1516
train=> Epoch: 700 Average training loss: 6.1526
test=> Epoch: 700 Average test loss: 6.1470
train=> Epoch: 800 Average training loss: 6.1504
test=> Epoch: 800 Average test loss: 6.1481
train=> Epoch: 900 Average training loss: 6.1590
test=> Epoch: 900 Average test loss: 6.1459
train=> Epoch: 1000 Average training loss: 6.1555
```

In [166]:
```
len(train_time_Data), len(epoch_Data), len(test_loss_Data), len(average_test_loss_Data)
```

Out[166]:
```
(316000, 452000, 136000, 136000)
```

In [168]:
```
# #Train Data
# dataframe_train = {'time':train_time_Data,'epoch': epoch_Data, 'test_loss': test_loss_Data, 'average_test_loss':average_test_los
# dataframe_train= pd.DataFrame(data = dataframe_train)
# #sns.pairplot(dataframe_train)
# dataframe_train.to_csv("train_result_data 3.csv")

# # #Test Data
# # dataframe_test = {'time':test_time_Data,'epoch': epoch, 'test_loss': test_loss_Data, 'average_test_loss':average_test_loss_Dat
# # dataframe_test= pd.DataFrame(data = dataframe_test)
# # sns.pairplot(dataframe_test)
# # dataframe_test.to_csv("test_result_data 3.csv")
```

In [169]:
```
train_time = train_end_time - train_start_time
print("Total Training Time for",epochs, "epochs:", round(train_time) , "seconds")
```

```
Total Training Time for 2000 epochs: 5633 seconds
```

In [170]:
```
scaler = trainloader.dataset.standardizer
```

In [171]:
```
with torch.no_grad():
    for batch_idx, data in enumerate(testloader):
        data = data.to(device)
        optimizer.zero_grad()
        reconstructed_data, mu, logvar = model(data) # a vector of means, μ, and another vector of standard deviations, σ.
```

In [172]:
```
reconstructed_data.size()
```

Out[172]:
```
torch.Size([532, 10])
```

In [173]:
```
sigma = torch.exp(logvar/2)
```

In [174]:
```
mu[1], sigma[1]
```

Out[174]:
```
(tensor([-0.7053, -0.4228, -0.3559]), tensor([0.7094, 0.1745, 0.4548]))
```

In [175]:
```
mu.mean(axis=0)
```

Out[175]:
```
tensor([-0.0008, -0.0057,  0.0079])
```

In [176]:
```
 sigma.mean(axis=0)
```

Out[176]:
```
tensor([0.7156, 0.1818, 0.4224])
```

In [177]:
```
# sample z from q
synthetic_data_size = 284807
q = torch.distributions.Normal(mu.mean(axis=0), sigma.mean(axis=0))
z = q.rsample(sample_shape=torch.Size([synthetic_data_size])) # q-->latent matrix | Sampling z in VAE
```

In [178]:
```
z.shape,z
```

Out[178]:
```
(torch.Size([284807, 3]),
 tensor([[-0.0022,  0.2897,  0.0950],
         [-1.3041, -0.0706, -0.1575],
         [ 0.3256, -0.2474, -0.1322],
         ...,
         [ 0.1041,  0.2391, -0.1605],
         [-0.5000,  0.0035,  0.2166],
         [ 0.1258,  0.0412, -0.3201]]))
```

In [179]:
```
with torch.no_grad():pred = model.decode(z).cpu().numpy()
pred # predicted values
```

Out[179]:
```
array([[ 1.5424129 ,  1.512798  ,  0.00782269, ...,  1.9011494 ,
         0.26974756, -0.16690125],
       [-0.3826072 , -0.3844935 , -0.4612161 , ..., -0.34830615,
         0.71095926, -1.1435223 ],
       [-0.25680742, -0.25622344,  0.14264421, ..., -0.40213773,
        -1.5361156 ,  0.22210172],
       ...,
       [-0.3602102 , -0.35860384,  0.5226393 , ...,  3.2392166 ,
        -0.73298943, -0.10087059],
       [-0.37644583, -0.37546676,  0.709481  , ..., -0.3743603 ,
         0.58353096, -0.76592606],
       [-0.33916867, -0.33697093, -0.9781519 , ..., -0.35150895,
         0.66486186,  0.16139537]], dtype=float32)
```

In [180]:
```
synthetic_data = scaler.inverse_transform(pred)
synthetic_data.shape
```

Out[180]:
```
(284807, 10)
```

In [181]:
```
synthetic_data = pd.DataFrame(synthetic_data, columns = columns)
synthetic_data.head(20)
```

Out[181]:

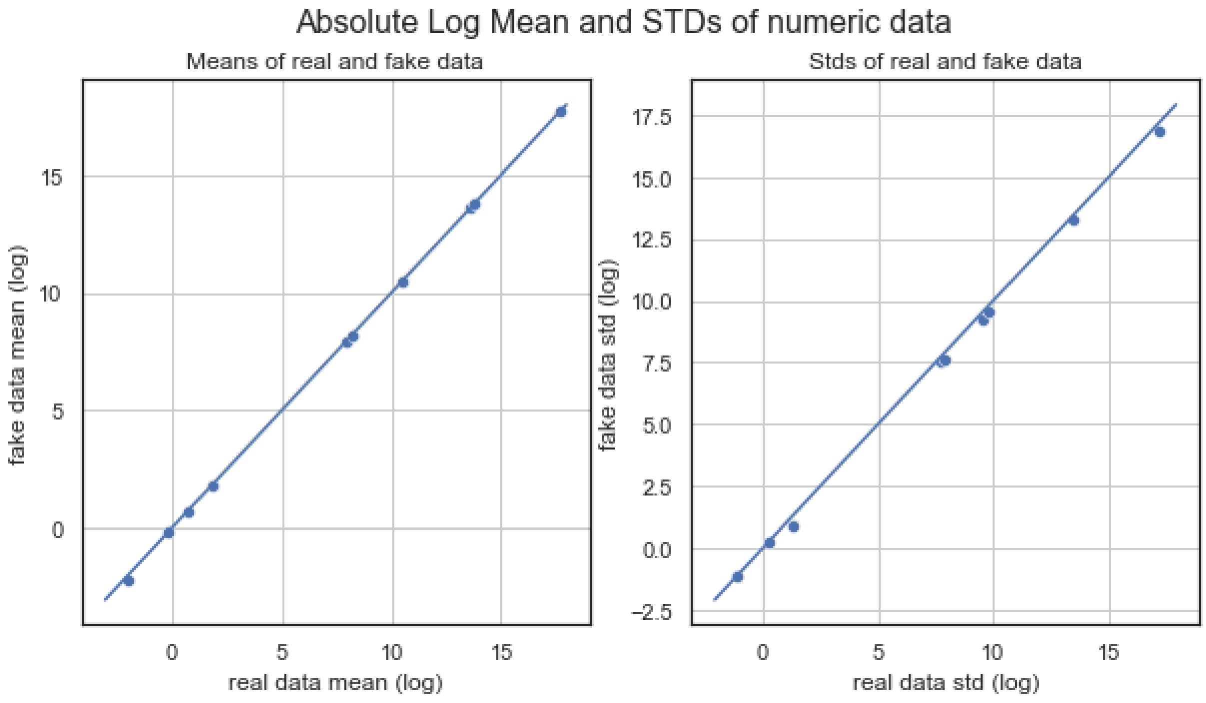
	trans_id	account_id	date	amount	balance	account	type_encoded	operation_encoded	k_symbol_encoded	bank_encoded
0	1.893603e+06	6272.414551	966592.7500	4443.313477	24104.875000	27465290.0	0.223791	0.776208	2.386926	5.408965
1	5.416495e+05	1843.711792	960217.5625	3525.596436	25382.693359	69885768.0	0.985758	0.014241	2.976062	1.747665
2	6.299995e+05	2143.122803	968425.2500	1850.156738	24884.314453	52236880.0	1.003997	-0.003994	-0.024384	6.867317
3	6.679832e+05	2266.719238	964865.9375	1794.346069	35766.082031	8923333.0	1.006320	-0.006309	-0.067164	5.700200
4	8.456946e+05	2869.657715	973192.8125	1960.037964	81713.898438	73779520.0	0.992664	0.007342	-0.020522	3.067994
5	5.502711e+05	1882.722900	971349.3125	1974.176147	22585.460938	34013564.0	0.998054	0.001942	2.824395	4.767800
6	6.535631e+05	2237.844971	974768.7500	2472.256592	77018.289062	33889916.0	1.000068	-0.000071	2.719403	9.184162
7	5.525154e+05	1884.843262	946675.0625	5077.654785	33902.921875	27327288.0	0.960367	0.039629	2.916970	4.194983
8	5.475146e+05	1866.004272	972270.3750	5685.538086	27839.410156	55351056.0	-0.028392	1.028388	1.035408	6.009065
9	5.867245e+05	2003.447021	976215.5625	2428.482910	32782.207031	27775046.0	0.997629	0.002366	2.683357	6.304288
10	6.214750e+05	2114.303223	967827.5625	1990.668213	25992.718750	54010584.0	1.004320	-0.004317	-0.054147	2.727832
11	6.875268e+05	2355.573242	963598.3125	6535.298828	27761.240234	44473856.0	0.982841	0.017154	2.961886	8.206357

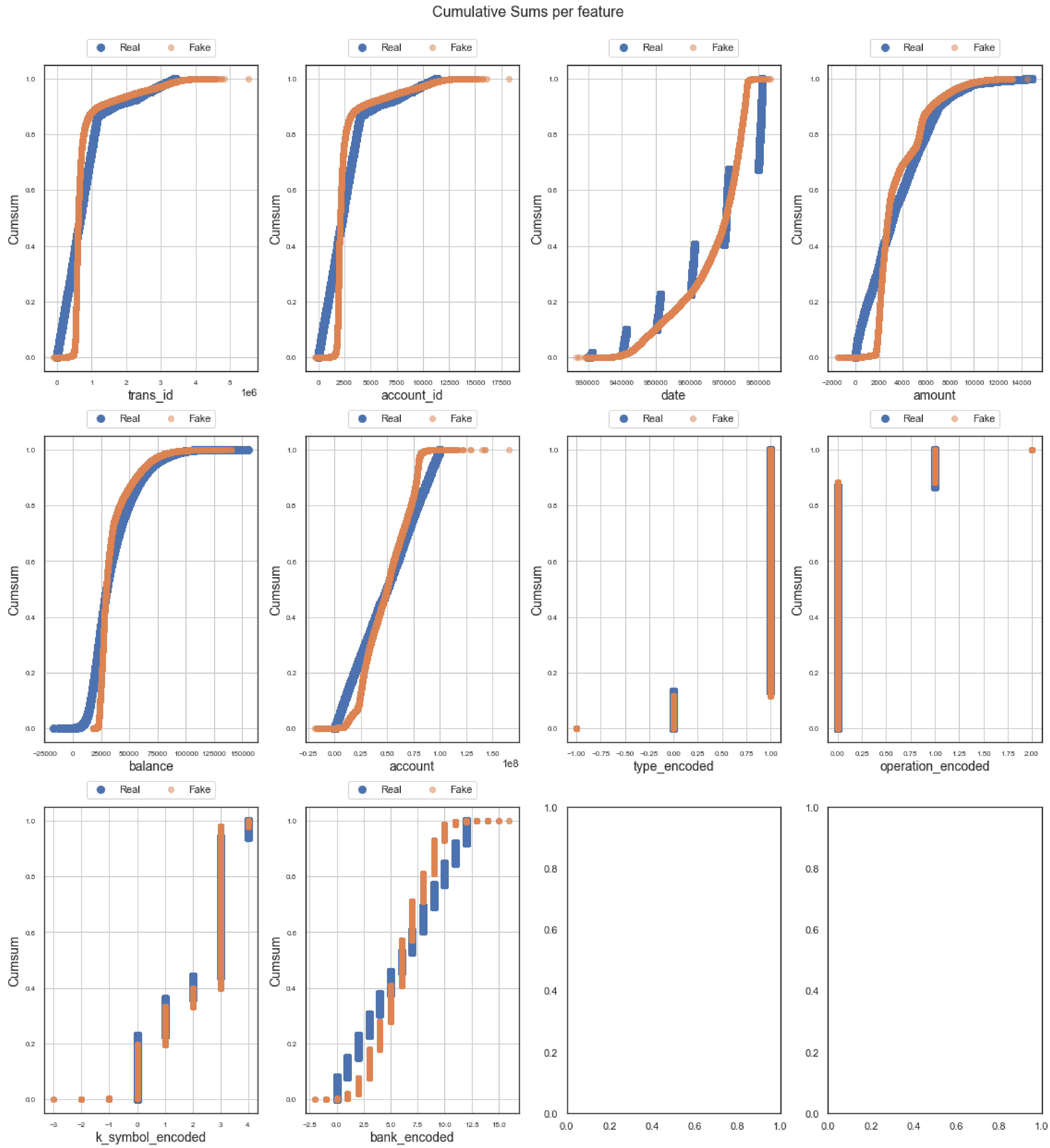
In [182]:
```
synthetic_data_rounded =synthetic_data.round(decimals=0)
```

In [184]:

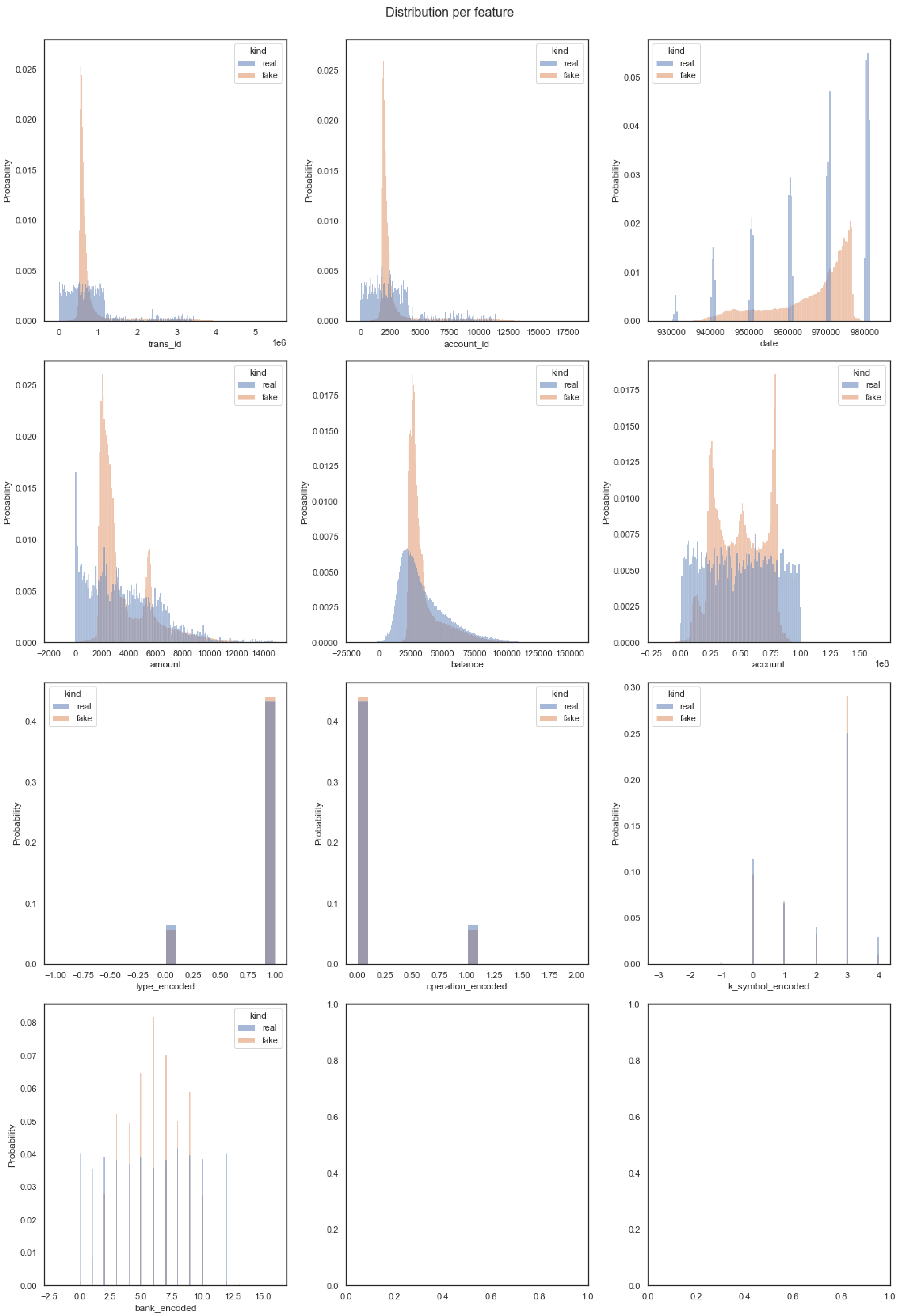
```
from table_evaluator import load_data, TableEvaluator

print(len(data_original), len(synthetic_data_rounded))
table_evaluator =  TableEvaluator(data_original, synthetic_data_rounded)
table_evaluator.visual_evaluation()
```

```
230465 284807
```

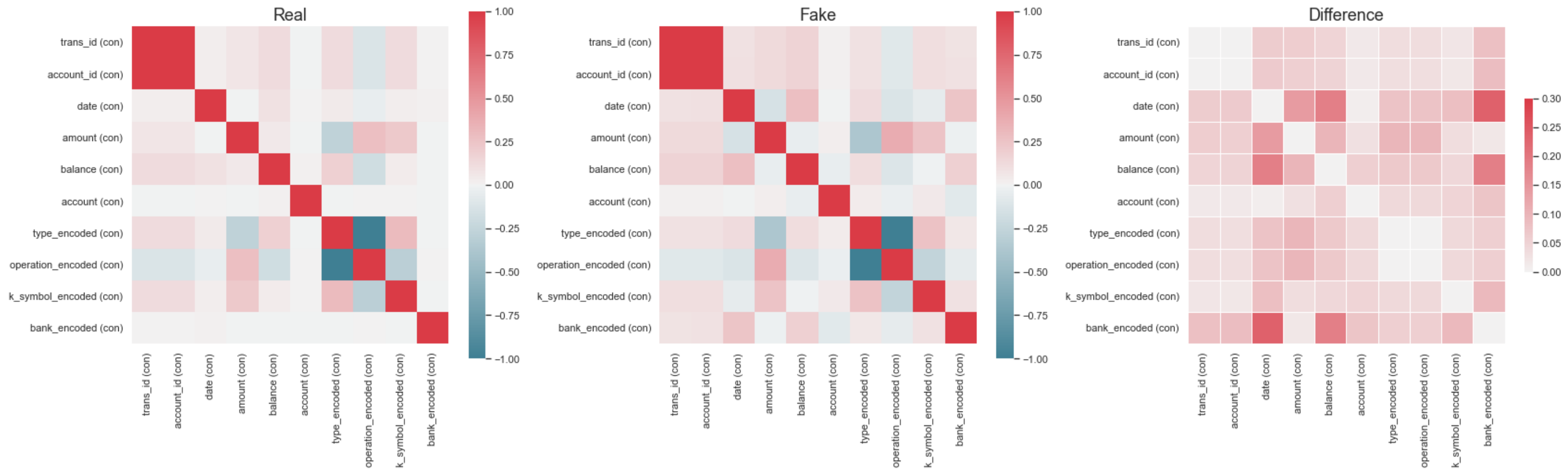


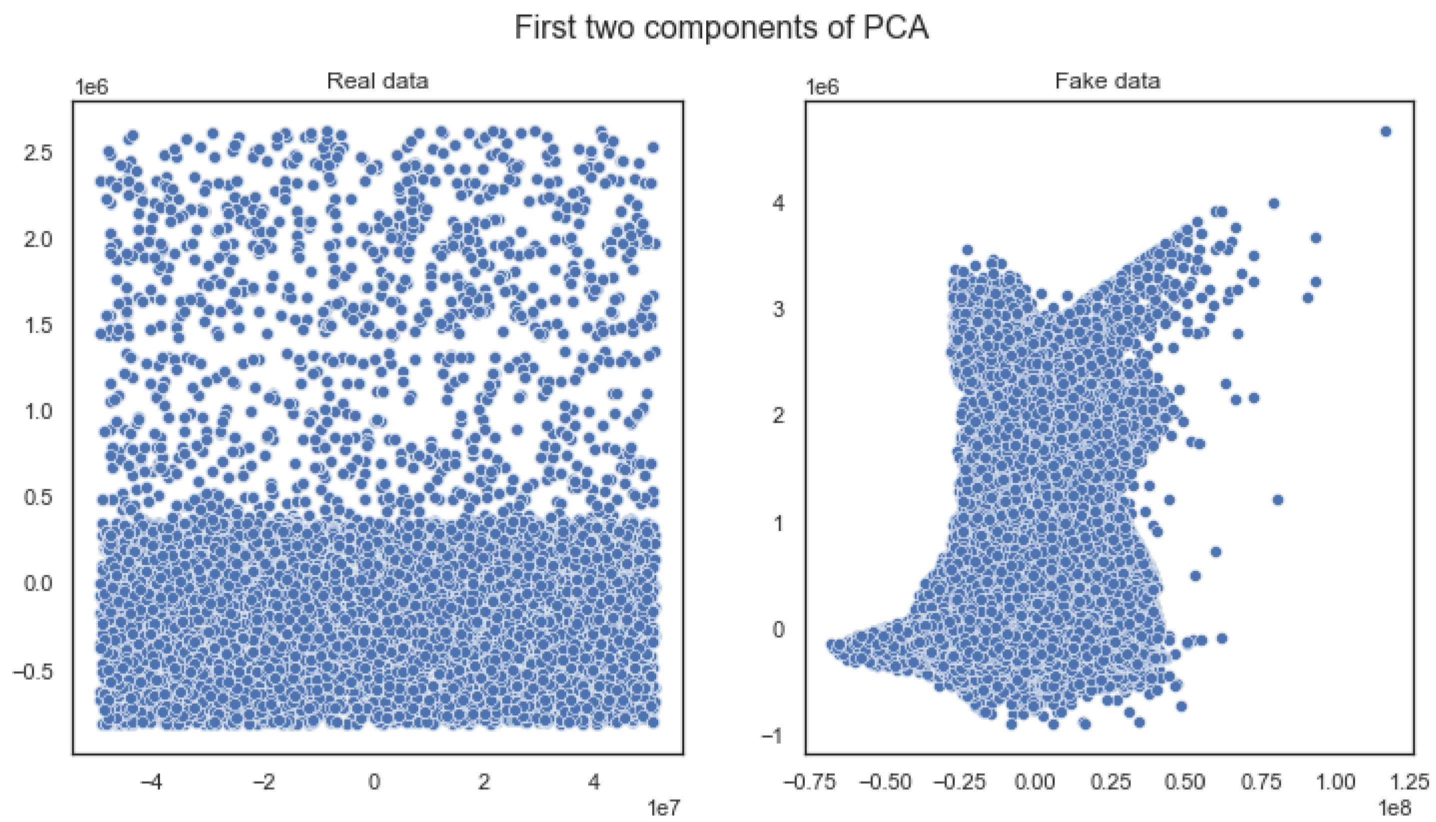


Cumulative Sums per feature
Real
Fake
Cumsum
trans_id
account_id
date
amount
balance
account
type_encoded
operation_encoded
k_symbol_encoded
bank_encoded
1e6
1e8



Distribution per feature
kind
real
fake
Probability
trans_id
1e6
account_id
date
amount
balance
account
1e8
type_encoded
operation_encoded
k_symbol_encoded
bank_encoded






```
In [185]: VAE_Synthetic_Data = datetime.datetime.now().strftime("VAE_Synthetic_Data3 %d-%m-%Y(%H.%M Hrs).csv")
          synthetic_data.to_csv(VAE_Synthetic_Data)
```

In [186]:
```
end = time.time()
total_time = end - start
print("End Time:" ,datetime.datetime.fromtimestamp(end).strftime('%Y-%m-%d %H:%M:%S'))
print("Total Run Time:", round(total_time/3600) , "Hours")
```

```
End Time: 2022-08-09 01:01:20
Total Run Time: 4 Hours
```

```
model.save('model_keras_example')
```

```
checkpoint_model = ModelCheckpoint(os.path.join(save_path, "model.h5"), verbose=1)
```

+==+

Notes:

https://harvard-iacs.github.io/2019-CS109B/labs/lab10/VAE/ (https://harvard-iacs.github.io/2019-CS109B/labs/lab10/VAE/)

https://jhui.github.io/2017/03/06/Variational-autoencoders/ (https://jhui.github.io/2017/03/06/Variational-autoencoders/)

https://medium.com/@olivia.liang032/how-to-measure-statistical-similarity-on-tabular-data-demonstrated-using-synthetic-data-66a1aa60084d (https://medium.com/@olivia.liang032/how-to-measure-statistical-similarity-on-tabular-data-demonstrated-using-synthetic-data-66a1aa60084d)

https://towardsdatascience.com/visualising-high-dimensional-datasets-using-pca-and-t-sne-in-python-8ef87e7915b (https://towardsdatascience.com/visualising-high-dimensional-datasets-using-pca-and-t-sne-in-python-8ef87e7915b)

It is highly recommended to use another dimensionality reduction method (e.g. PCA for dense data or TruncatedSVD for sparse data) to reduce the number of dimensions to a reasonable amount (e.g. 50) if the number of features is very high.

https://colab.research.google.com/github/tvhahn/Manufacturing-Data-Science-with-Python/blob/master/Metal%20Machining/1.B_building-vae.ipynb#scrollTo=t6mNH0b6RnlU (https://colab.research.google.com/github/tvhahn/Manufacturing-Data-Science-with-Python/blob/master/Metal%20Machining/1.B_building-vae.ipynb#scrollTo=t6mNH0b6RnlU)

In [ ]: