Importing and Transforming data

In [1]:
```
import pandas as pd
import numpy as np


real_data = pd.read_csv(r"C:\Users\Home\Jupyter\Datasets\creditcard.csv")
#del real_data['Time']
data_synthetic = pd.read_csv(r"C:\Users\Home\Jupyter\VAE_Synthetic_Data.csv")
#del data_synthetic['Time']
```

In [2]:
```
real_data.drop(labels='Time', axis=1, inplace=True)
data_synthetic.drop(labels='Unnamed: 0', axis=1, inplace=True)
data_synthetic = pd.read_csv(r"C:\Users\Home\Jupyter\VAE_Synthetic_Data.csv")
data_synthetic.drop(columns = "Unnamed: 0",axis = 1,inplace = True)

data_synthetic =data_synthetic.round(3)
data_synthetic.Class =data_synthetic.Class.round(0)
data_synthetic[data_synthetic.Class < 0] = 0
data_synthetic.Class =data_synthetic.Class.astype(np.int64)
data_synthetic
```

Out[2]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0.979	-0.058	0.641	0.362	-0.829	-0.210	-0.085	0.173	0.119	0.140	...	-0.078	-0.160	0.138	-0.019	0.240	0.051	-0.080	-0.068	60.163	0
1	-1.222	0.544	0.658	-0.467	0.512	-0.404	0.878	0.192	-0.536	-0.363	...	-0.074	0.039	-0.104	0.050	-0.235	-0.043	0.109	0.047	63.899	0
2	1.898	-0.577	-1.049	0.386	-0.251	-0.641	-0.197	0.005	0.447	0.205	...	-0.060	0.088	-0.014	0.129	0.197	0.050	-0.014	0.008	89.232	0
3	-0.624	-0.271	-0.086	0.009	2.587	4.570	-0.497	1.057	0.344	-0.083	...	0.024	-0.469	-0.482	1.450	0.251	0.062	-0.084	-0.042	146.501	0
4	-1.328	0.483	1.084	0.293	1.898	1.573	-0.145	0.402	-0.362	0.186	...	-0.095	-0.145	0.192	0.175	-0.081	0.057	0.129	0.350	63.017	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
68742	0.175	0.658	0.711	2.123	0.310	0.441	0.126	0.314	-0.090	0.223	...	-0.034	0.029	0.112	0.037	0.003	0.113	0.148	0.072	48.818	0
68743	1.266	-1.158	0.680	-0.230	-1.453	0.371	-1.120	0.223	0.182	0.317	...	-0.027	0.236	0.151	-0.041	0.160	-0.067	0.111	0.044	117.371	0
68744	-0.117	0.378	0.722	-0.368	0.115	-0.454	0.392	0.177	-0.156	-0.133	...	-0.047	-0.024	0.142	0.010	-0.016	-0.000	-0.008	0.030	45.800	0
68745	1.054	-0.185	0.382	0.218	-0.501	-0.403	-0.293	0.082	0.221	-0.021	...	-0.139	-0.195	0.199	-0.006	0.201	0.064	0.023	0.036	49.084	0
68746	1.491	0.565	-0.216	3.558	0.206	0.661	0.153	0.169	-0.664	1.003	...	-0.016	0.127	0.315	-0.105	0.041	0.114	-0.013	-0.049	34.188	0

68747 rows × 30 columns

In [3]:
```
data_synthetic.Amount.max()
```

Out[3]:
```
7754.646
```

Cleaning data

1. Drop columns

In [4]:
```
# real_data = real_data.loc[:, ~real_data.columns.str.contains('^Unnamed|id|ID')]
# data_synthetic = data_synthetic.loc[:, ~data_synthetic.columns.str.contains('^Unnamed|ID|id')]
# real_data = real_data.dropna()
# data_synthetic = data_synthetic.dropna()
```

2. Encode categorical columns and normalize numerical columns

In [5]:
```
# Categorical columns that are string types
cat_list = ['Class']
# Categorical columns that are numerical types
numcat_list = [ ]
# Numerical columns
num_list = ['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11',
       'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V21',
       'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount']
```

In [6]:
```
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
le = LabelEncoder()
oe = OneHotEncoder(sparse=False)

from sklearn.preprocessing import StandardScaler, MinMaxScaler
scaler = MinMaxScaler(feature_range=(-1, 1))

def transform_cat(df):
    df_2 = df.apply(le.fit_transform)
    df_oe = oe.fit_transform(df_2)
    df_oe = pd.DataFrame(df_oe)
    return df_oe
```

```
def transform_numcat(df):
    df_oe = oe.fit_transform(df)
    df_oe = pd.DataFrame(df_oe)
    return df_oe

def transform_num(df):
    df_2 = scaler.fit_transform(df)
    df_2 = pd.DataFrame(df_2)
    return df_2
```

In [7]:
```
numcat_train = transform_numcat(real_data[numcat_list])
num_train = transform_num(real_data[num_list])
cat_train = transform_cat(real_data[cat_list])

numcat_test = transform_numcat(data_synthetic[numcat_list])
num_test = transform_num(data_synthetic[num_list])
cat_test = transform_cat(data_synthetic[cat_list])
```

In [8]:
```
# Integrate datasets
x_train = pd.concat([numcat_test, cat_train], axis=1, sort=False)
x_train = pd.concat([x_train, num_train], axis=1, sort=False)

x_test = pd.concat([numcat_test, cat_test], axis=1, sort=False)
x_test = pd.concat([x_test, num_test], axis=1, sort=False)
```

3. Reshape for modeling

In [9]:
```
x_train = np.array(x_train)
x_test = np.array(x_test)
# Flatten the data into vectors
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))
print(x_train.shape)
print(x_test.shape)
```

```
(284807, 31)
(68747, 31)
```

Autoencoder

Autoencoder Structure Building

A single fully-connected neural layer as encoder and decoder

In [10]:
```
from keras.layers import Input, Dense
from keras.models import Model
def modeling_autoencoder(latent_dim, x_train):
    original_dim= x_train.shape[1]

    # this is our input placeholder
    input_data = Input(shape=(original_dim,))
    # "encoded" is the encoded representation of the input
    encoded = Dense(latent_dim, activation='relu')(input_data)
    # "decoded" is the lossy reconstruction of the input
    decoded = Dense(original_dim, activation='sigmoid')(encoded)

    # this model maps an input to its reconstruction (Define a model that would turn input_data into decoded output)
    autoencoder = Model(input_data, decoded)

    #### Create a separate encoder model ####
    # this model maps an input to its encoded representation
    encoder = Model(input_data, encoded)

    #### as well as the decoder model ####
    # create a placeholder for an encoded (assigned # of dimensions) input
    encoded_input = Input(shape=(latent_dim,))
    # retrieve the last layer of the autoencoder model
    decoder_layer = autoencoder.layers[-1]
    # create the decoder model
    decoder = Model(encoded_input, decoder_layer(encoded_input))

    #### Autoencoder model training ####
    autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')

    autoencoder.fit(x_train, x_train,
                epochs=50,
                batch_size=256,
                shuffle=True,
                validation_split = 0.2)

    return encoder, decoder
```

Autoencoder Model Inference

In [11]:
```
trained_encoder = modeling_autoencoder(1, x_train)[0]
encoded_testdata = trained_encoder.predict(x_test)
encoded_traindata = trained_encoder.predict(x_train)
```

```
Epoch 1/50
891/891 [==============================] - 2s 2ms/step - loss: 0.6927 - val_loss: 0.6922
Epoch 2/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6916 - val_loss: 0.6909
Epoch 3/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6901 - val_loss: 0.6893
Epoch 4/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6884 - val_loss: 0.6874
Epoch 5/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6864 - val_loss: 0.6853
Epoch 6/50
891/891 [==============================] - 2s 2ms/step - loss: 0.6843 - val_loss: 0.6831
Epoch 7/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6820 - val_loss: 0.6807
Epoch 8/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6795 - val_loss: 0.6782
Epoch 9/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6769 - val_loss: 0.6755
Epoch 10/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6742 - val_loss: 0.6727
Epoch 11/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6714 - val_loss: 0.6697
Epoch 12/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6684 - val_loss: 0.6667
Epoch 13/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6654 - val_loss: 0.6636
Epoch 14/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6622 - val_loss: 0.6604
Epoch 15/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6590 - val_loss: 0.6571
Epoch 16/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6556 - val_loss: 0.6537
Epoch 17/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6522 - val_loss: 0.6502
Epoch 18/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6488 - val_loss: 0.6467
Epoch 19/50
891/891 [==============================] - 2s 2ms/step - loss: 0.6452 - val_loss: 0.6431
Epoch 20/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6416 - val_loss: 0.6394
Epoch 21/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6379 - val_loss: 0.6357
Epoch 22/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6342 - val_loss: 0.6319
Epoch 23/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6304 - val_loss: 0.6280
Epoch 24/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6266 - val_loss: 0.6241
Epoch 25/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6227 - val_loss: 0.6202
Epoch 26/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6188 - val_loss: 0.6162
Epoch 27/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6148 - val_loss: 0.6122
Epoch 28/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6108 - val_loss: 0.6081
Epoch 29/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6068 - val_loss: 0.6041
Epoch 30/50
891/891 [==============================] - 1s 2ms/step - loss: 0.6027 - val_loss: 0.5999
Epoch 31/50
891/891 [==============================] - 2s 2ms/step - loss: 0.5986 - val_loss: 0.5958
Epoch 32/50
891/891 [==============================] - 1s 2ms/step - loss: 0.5944 - val_loss: 0.5916
Epoch 33/50
891/891 [==============================] - 1s 2ms/step - loss: 0.5902 - val_loss: 0.5873
Epoch 34/50
891/891 [==============================] - 1s 2ms/step - loss: 0.5861 - val_loss: 0.5831
Epoch 35/50
891/891 [==============================] - 1s 2ms/step - loss: 0.5818 - val_loss: 0.5788
Epoch 36/50
891/891 [==============================] - 1s 2ms/step - loss: 0.5776 - val_loss: 0.5745
Epoch 37/50
891/891 [==============================] - 2s 2ms/step - loss: 0.5733 - val_loss: 0.5702
Epoch 38/50
891/891 [==============================] - 2s 2ms/step - loss: 0.5690 - val_loss: 0.5659
Epoch 39/50
891/891 [==============================] - 2s 2ms/step - loss: 0.5647 - val_loss: 0.5615
Epoch 40/50
891/891 [==============================] - 2s 2ms/step - loss: 0.5604 - val_loss: 0.5572
Epoch 41/50
891/891 [==============================] - 2s 2ms/step - loss: 0.5561 - val_loss: 0.5528
Epoch 42/50
891/891 [==============================] - 2s 2ms/step - loss: 0.5517 - val_loss: 0.5484
Epoch 43/50
891/891 [==============================] - 2s 2ms/step - loss: 0.5474 - val_loss: 0.5440
Epoch 44/50
891/891 [==============================] - 2s 2ms/step - loss: 0.5430 - val_loss: 0.5396
Epoch 45/50
891/891 [==============================] - 1s 2ms/step - loss: 0.5386 - val_loss: 0.5352
Epoch 46/50
891/891 [==============================] - 1s 2ms/step - loss: 0.5342 - val_loss: 0.5307
Epoch 47/50
891/891 [==============================] - 2s 2ms/step - loss: 0.5299 - val_loss: 0.5263
Epoch 48/50
```

```
891/891 [==============================] - 2s 2ms/step - loss: 0.5255 - val_loss: 0.5218
Epoch 49/50
891/891 [==============================] - 1s 2ms/step - loss: 0.5211 - val_loss: 0.5174
Epoch 50/50
891/891 [==============================] - 1s 2ms/step - loss: 0.5166 - val_loss: 0.5129
2149/2149 [==============================] - 2s 787us/step
8901/8901 [==============================] - 7s 804us/step
```

Calculate Similarity Score

In [12]:
```
bins = np.arange(0,8000,20)

real_inds = pd.DataFrame(np.digitize(encoded_traindata, bins), columns = ['Amount'])
syn_inds = pd.DataFrame(np.digitize(encoded_testdata, bins), columns = ['Amount'])
```

In [13]:
```
def identify_probs(table,column):
    counts = table[column].value_counts()
    freqs = {counts.index[i]: counts.values[i] for i in range(len(counts.index))}
    for i in range(1, len(bins)+1):
        if i not in freqs.keys():
            freqs[i] = 0
    sorted_freqs = {}
    for k in sorted(freqs.keys()):
        sorted_freqs[k] = freqs[k]
    probs = []
    for k,v in sorted_freqs.items():
        probs.append(v/len(table[column]))
    return sorted_freqs, np.array(probs)
```

In [14]:
```
from scipy.spatial import distance

real_p = identify_probs(real_inds,'Amount')[1]
syn_p = identify_probs(syn_inds,'Amount')[1]
def cos_similarity(p,q):
    return 1 - distance.cosine(p, q)
cos_similarity(real_p,syn_p)
```

Out[14]:
```
1
```

Dimension Reduction Visualization

Extract 5-dimensional data from autoencoder

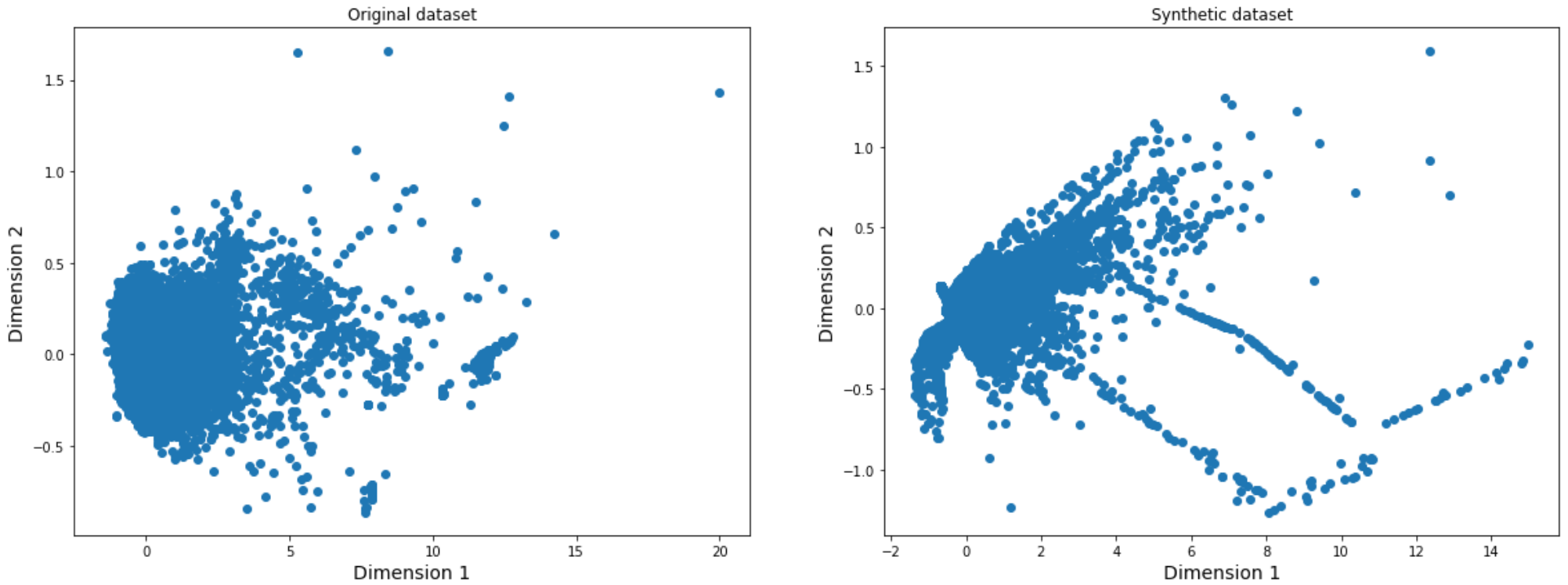
In [15]:
```
trained_encoder = modeling_autoencoder(5, x_train)[0]
encoded_testdata = trained_encoder.predict(x_test)
encoded_traindata = trained_encoder.predict(x_train)
```

```
Epoch 1/50
891/891 [==============================] - 2s 2ms/step - loss: 0.7113 - val_loss: 0.7091
Epoch 2/50
891/891 [==============================] - 2s 2ms/step - loss: 0.7065 - val_loss: 0.7036
Epoch 3/50
891/891 [==============================] - 2s 2ms/step - loss: 0.7006 - val_loss: 0.6972
Epoch 4/50
891/891 [==============================] - 2s 2ms/step - loss: 0.6939 - val_loss: 0.6902
Epoch 5/50
891/891 [==============================] - 2s 2ms/step - loss: 0.6866 - val_loss: 0.6827
Epoch 6/50
891/891 [==============================] - 2s 2ms/step - loss: 0.6789 - val_loss: 0.6746
Epoch 7/50
891/891 [==============================] - 2s 2ms/step - loss: 0.6707 - val_loss: 0.6661
Epoch 8/50
891/891 [==============================] - 2s 2ms/step - loss: 0.6619 - val_loss: 0.6569
Epoch 9/50
891/891 [==============================] - 2s 2ms/step - loss: 0.6524 - val_loss: 0.6469
Epoch 10/50
891/891 [==============================] - 2s 2ms/step - loss: 0.6419 - val_loss: 0.6358
Epoch 11/50
891/891 [==============================] - 2s 2ms/step - loss: 0.6303 - val_loss: 0.6234
Epoch 12/50
891/891 [==============================] - 2s 2ms/step - loss: 0.6171 - val_loss: 0.6092
Epoch 13/50
891/891 [==============================] - 2s 2ms/step - loss: 0.6022 - val_loss: 0.5931
Epoch 14/50
891/891 [==============================] - 2s 2ms/step - loss: 0.5850 - val_loss: 0.5746
Epoch 15/50
891/891 [==============================] - 2s 2ms/step - loss: 0.5654 - val_loss: 0.5533
Epoch 16/50
891/891 [==============================] - 2s 2ms/step - loss: 0.5429 - val_loss: 0.5289
Epoch 17/50
891/891 [==============================] - 2s 2ms/step - loss: 0.5171 - val_loss: 0.5011
Epoch 18/50
891/891 [==============================] - 2s 2ms/step - loss: 0.4877 - val_loss: 0.4694
Epoch 19/50
891/891 [==============================] - 2s 2ms/step - loss: 0.4544 - val_loss: 0.4337
Epoch 20/50
891/891 [==============================] - 2s 2ms/step - loss: 0.4171 - val_loss: 0.3938
Epoch 21/50
891/891 [==============================] - 2s 2ms/step - loss: 0.3757 - val_loss: 0.3496
Epoch 22/50
891/891 [==============================] - 2s 2ms/step - loss: 0.3302 - val_loss: 0.3014
Epoch 23/50
891/891 [==============================] - 2s 2ms/step - loss: 0.2808 - val_loss: 0.2494
Epoch 24/50
891/891 [==============================] - 2s 2ms/step - loss: 0.2280 - val_loss: 0.1939
Epoch 25/50
891/891 [==============================] - 2s 2ms/step - loss: 0.1721 - val_loss: 0.1356
Epoch 26/50
891/891 [==============================] - 2s 2ms/step - loss: 0.1137 - val_loss: 0.0750
Epoch 27/50
891/891 [==============================] - 2s 2ms/step - loss: 0.0532 - val_loss: 0.0124
Epoch 28/50
891/891 [==============================] - 2s 2ms/step - loss: -0.0089 - val_loss: -0.0519
Epoch 29/50
891/891 [==============================] - 2s 2ms/step - loss: -0.0725 - val_loss: -0.1179
Epoch 30/50
891/891 [==============================] - 2s 2ms/step - loss: -0.1379 - val_loss: -0.1860
Epoch 31/50
891/891 [==============================] - 2s 2ms/step - loss: -0.2056 - val_loss: -0.2568
Epoch 32/50
891/891 [==============================] - 2s 2ms/step - loss: -0.2762 - val_loss: -0.3312
Epoch 33/50
891/891 [==============================] - 2s 2ms/step - loss: -0.3506 - val_loss: -0.4101
Epoch 34/50
891/891 [==============================] - 2s 2ms/step - loss: -0.4300 - val_loss: -0.4948
Epoch 35/50
891/891 [==============================] - 2s 2ms/step - loss: -0.5155 - val_loss: -0.5864
Epoch 36/50
891/891 [==============================] - 2s 2ms/step - loss: -0.6084 - val_loss: -0.6864
Epoch 37/50
891/891 [==============================] - 2s 2ms/step - loss: -0.7100 - val_loss: -0.7960
Epoch 38/50
891/891 [==============================] - 2s 2ms/step - loss: -0.8217 - val_loss: -0.9169
Epoch 39/50
891/891 [==============================] - 2s 2ms/step - loss: -0.9450 - val_loss: -1.0504
Epoch 40/50
891/891 [==============================] - 2s 2ms/step - loss: -1.0814 - val_loss: -1.1983
Epoch 41/50
891/891 [==============================] - 2s 2ms/step - loss: -1.2326 - val_loss: -1.3623
Epoch 42/50
891/891 [==============================] - 2s 2ms/step - loss: -1.4003 - val_loss: -1.5444
Epoch 43/50
891/891 [==============================] - 2s 2ms/step - loss: -1.5865 - val_loss: -1.7466
Epoch 44/50
891/891 [==============================] - 2s 2ms/step - loss: -1.7934 - val_loss: -1.9712
Epoch 45/50
891/891 [==============================] - 2s 2ms/step - loss: -2.0233 - val_loss: -2.2209
Epoch 46/50
891/891 [==============================] - 2s 2ms/step - loss: -2.2787 - val_loss: -2.4983
Epoch 47/50
891/891 [==============================] - 2s 2ms/step - loss: -2.5627 - val_loss: -2.8067
Epoch 48/50
```

```
891/891 [==============================] - 2s 2ms/step - loss: -2.8783 - val_loss: -3.1495
Epoch 49/50
891/891 [==============================] - 2s 2ms/step - loss: -3.2291 - val_loss: -3.5304
Epoch 50/50
891/891 [==============================] - 2s 2ms/step - loss: -3.6189 - val_loss: -3.9536
2149/2149 [==============================] - 2s 901us/step
8901/8901 [==============================] - 8s 867us/step
```

1. PCA

In [16]:
```python
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# T-sne visualization
pca = PCA(n_components=2, random_state = 0)
pca_train = pca.fit_transform(encoded_traindata)
pca_test = pca.fit_transform(encoded_testdata)
pca_train_df = pd.DataFrame(data = pca_train, columns = ('Dim_1','Dim_2'))
pca_test_df = pd.DataFrame(data = pca_test, columns = ('Dim_1','Dim_2'))

plt.figure(figsize = [20, 7])
plt.subplot(121)
plt.title('Original dataset')
plt.scatter(pca_train_df['Dim_1'],pca_train_df['Dim_2'], marker = 'o')
plt.xlabel('Dimension 1',fontsize=14)
plt.ylabel('Dimension 2',fontsize=14)
# plt.axis([-1.0, 2.0, -0.5, 1.5])

plt.subplot(122)
plt.title('Synthetic dataset')
plt.scatter(pca_test_df['Dim_1'],pca_test_df['Dim_2'], marker = 'o')
plt.xlabel('Dimension 1',fontsize=14)
plt.ylabel('Dimension 2',fontsize=14)
# plt.axis([-1.0, 2.0, -0.5, 1.5])

plt.show()
```

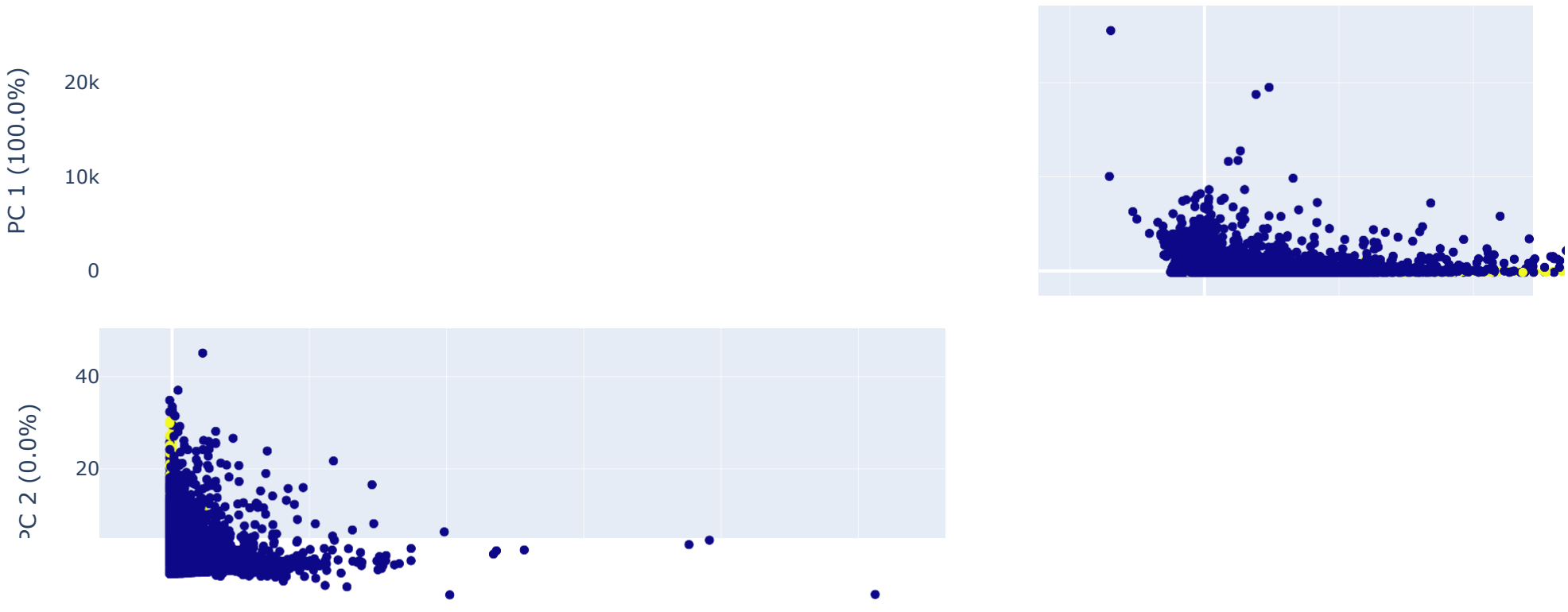


PCA with Plotly

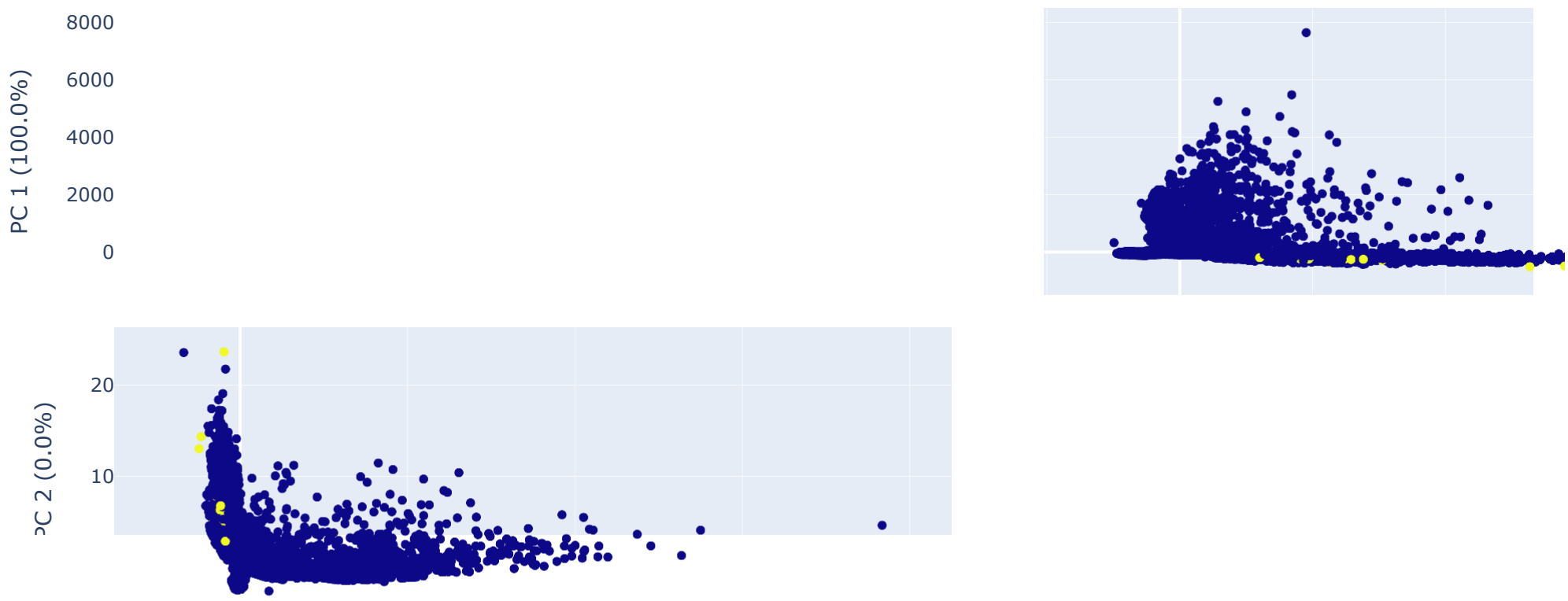
https://plotly.com/python/pca-visualization/

In [17]:
```python
#!pip install plotly
import plotly.express as px
from sklearn.decomposition import PCA

def PCA01(df,features):
        pca = PCA()
        components = pca.fit_transform(df[features])
        labels = {str(i): f"PC {i+1} ({var:.1f}%)"for i, var in enumerate(pca.explained_variance_ratio_ * 100)}

        fig = px.scatter_matrix(components,labels=labels,dimensions=range(2),color=df["Class"])
        fig.update_traces(diagonal_visible=False)
        fig.show()
```

In [18]:
```python
PCA01(real_data,real_data.columns)
```

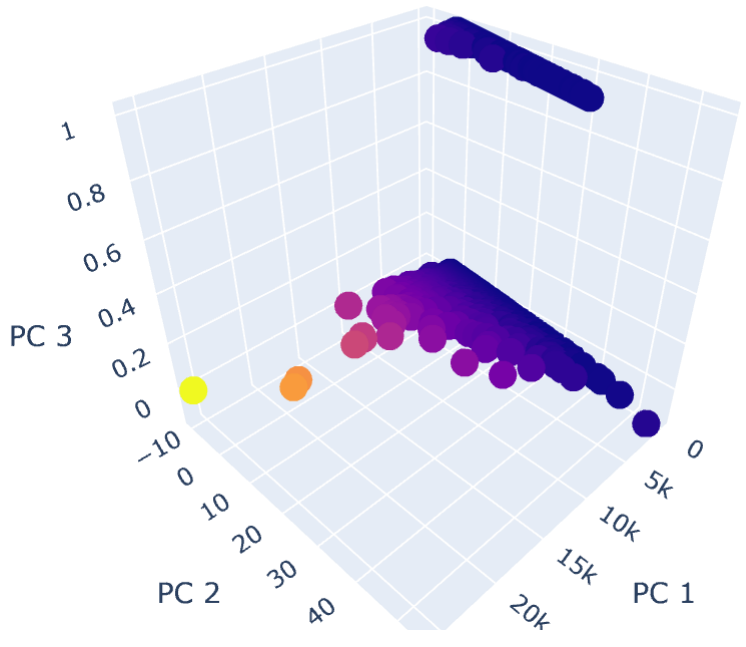


In [19]:
```
PCA01(data_synthetic,data_synthetic.columns)
```



In [20]:
```
#Disard the 3D PCA for Data 1
import plotly.express as px
from sklearn.decomposition import PCA

X= real_data[["Class", 'Amount',"V1"]]

pca = PCA(n_components=3)
components = pca.fit_transform(X)

total_var = pca.explained_variance_ratio_.sum() * 100

fig = px.scatter_3d(components, x=0, y=1, z=2, color=real_data['Amount'],
                    title=f'Total Explained Variance: {total_var:.2f}%',
                    labels={'0': 'PC 1', '1': 'PC 2', '2': 'PC 3'}
)
fig.show()
```

Total Explained Variance: 100.00%



2. T-SNE

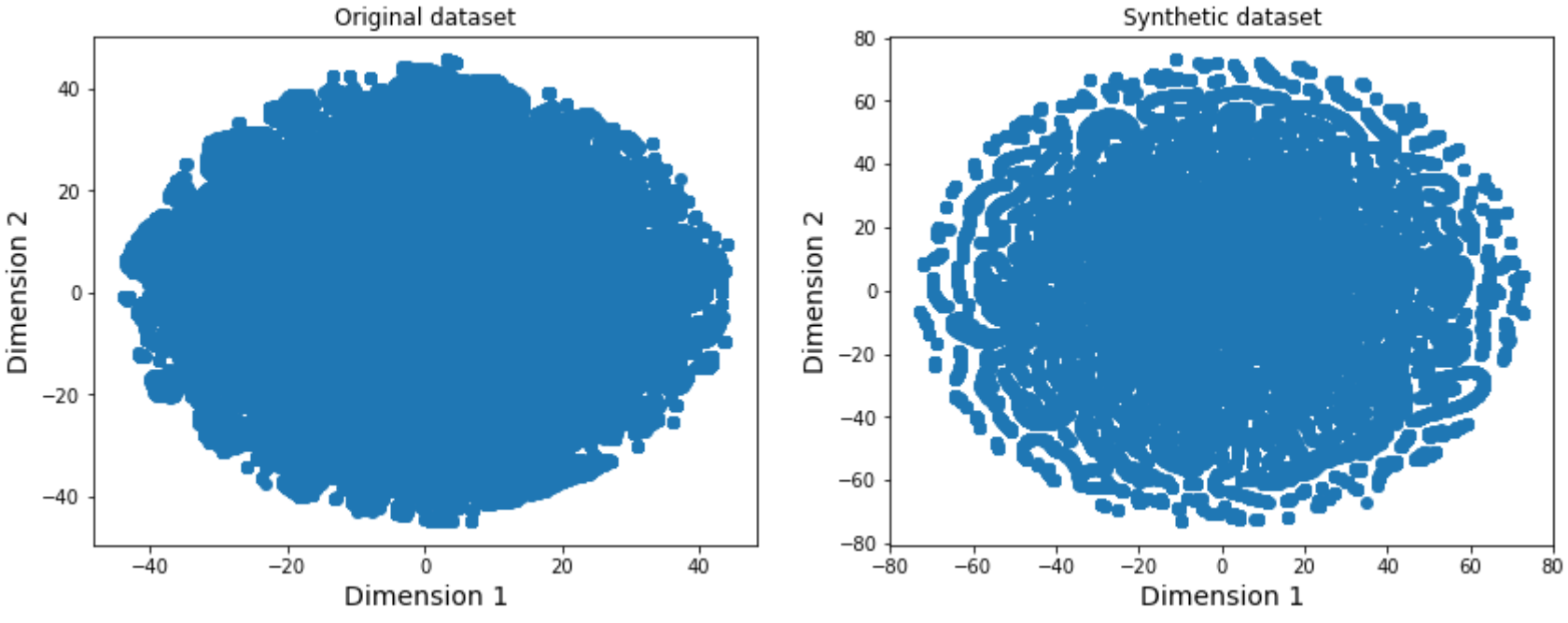
T-SNE and PCA Content

https://www.kaggle.com/code/parulpandey/part1-visualizing-kannada-mnist-with-pca?scriptVersionId=29322090

https://github.com/olekscode/Examples-PCA-tSNE/blob/master/Python/Visualizing%20Iris%20Dataset%20using%20PCA%20and%20t-SNE.ipynb

In [21]:
```
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

# T-sne visualization
tsne = TSNE(n_components = 2, random_state = 0)
tsne_train = tsne.fit_transform(encoded_traindata)
tsne_test = tsne.fit_transform(encoded_testdata)
tsne_train_df = pd.DataFrame(data = tsne_train, columns = ('Dim_1','Dim_2'))
tsne_test_df = pd.DataFrame(data = tsne_test, columns = ('Dim_1','Dim_2'))

plt.figure(figsize = [14, 5])
plt.subplot(121)
plt.title('Original dataset')
plt.scatter(tsne_train_df['Dim_1'],tsne_train_df['Dim_2'], marker = 'o')
plt.xlabel('Dimension 1',fontsize=14)
plt.ylabel('Dimension 2',fontsize=14)
# plt.axis([-30, 40, -40, 40])

plt.subplot(122)
plt.title('Synthetic dataset')
plt.scatter(tsne_test_df['Dim_1'],tsne_test_df['Dim_2'], marker = 'o')
plt.xlabel('Dimension 1',fontsize=14)
plt.ylabel('Dimension 2',fontsize=14)
# plt.axis([-30, 40, -40, 40])

plt.show()
```

```
C:\Users\Home\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\manifold\_t_sne.py:795: FutureWarning:

The default initialization in TSNE will change from 'random' to 'pca' in 1.2.

C:\Users\Home\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\manifold\_t_sne.py:805: FutureWarning:

The default learning rate in TSNE will change from 200.0 to 'auto' in 1.2.

C:\Users\Home\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\manifold\_t_sne.py:795: FutureWarning:

The default initialization in TSNE will change from 'random' to 'pca' in 1.2.

C:\Users\Home\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\manifold\_t_sne.py:805: FutureWarning:

The default learning rate in TSNE will change from 200.0 to 'auto' in 1.2.
```


Original dataset
Dimension 2
Dimension 1
40
20
0
−20
−40
−40
−20
0
20
40
Synthetic dataset
Dimension 2
Dimension 1
80
60
40
20
0
−20
−40
−60
−80
−80
−60
−40
−20
0
20
40
60
80