```
#Step 1 — Importing all required libraries.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Sequential,Model
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns

%matplotlib inline
```

** Use pandas to read in the csv file called anonymized_data.csv . It contains 500 rows and 30 columns of anonymized data along with 1 last column with a classification label, where the columns have been renamed to 4 letter codes.**

```
#Step 2 — Reading our input data.
data = pd.read_csv('/content/anonymized_data.csv')
data.head()
```

	EJWY	VALM	EGXO	HTGR	SKRF	NNSZ	NYLC	GWID	TVUT	CJHI	...	LKKS	UOBF	VBHE	FRWU	NDYZ	QSBO	JDUB	
0	-2.032145	1.019576	-9.658715	-6.210495	3.156823	7.457850	-5.313357	8.508296	3.959194	-5.246654	...	-2.209663	-10.340123	-7.697555	-5.932752	10.872688	0.081321	1.276316	5.2[illegible]
1	8.306217	6.649376	-0.960333	-4.094799	8.738965	-3.458797	7.016800	6.692765	0.898264	9.337643	...	0.851793	-9.678324	-6.071795	1.428194	-8.082792	-0.557089	-7.817282	-8.6[illegible]
2	6.570842	6.985462	-1.842621	-1.569599	10.039339	-3.623026	8.957619	7.577283	1.541255	7.161509	...	1.376085	-8.971164	-5.302191	2.898965	-8.746597	-0.520888	-7.350999	-8.9[illegible]
3	-1.139972	0.579422	-9.526530	-5.744928	4.834355	5.907235	-4.804137	6.798810	5.403670	-7.642857	...	0.270571	-8.640988	-8.105419	-5.079015	9.351282	0.641759	1.898083	3.9[illegible]
4	-1.738104	0.234729	-11.558768	-7.181332	4.189626	7.765274	-2.189083	7.239925	3.135602	-6.211390	...	-0.013973	-9.437110	-6.475267	-5.708377	9.623080	1.802899	1.903705	4.1[illegible]

5 rows × 31 columns

We have 30 feature columns and 1 Label column in our dataset. These feature columns are anonymous.

Step 3 – Checking info of our data.

We can see from the stats below that we don't have any null values in our data.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 31 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   EJWY    500 non-null    float64
 1   VALM    500 non-null    float64
 2   EGXO    500 non-null    float64
 3   HTGR    500 non-null    float64
 4   SKRF    500 non-null    float64
 5   NNSZ    500 non-null    float64
 6   NYLC    500 non-null    float64
 7   GWID    500 non-null    float64
 8   TVUT    500 non-null    float64
 9   CJHI    500 non-null    float64
 10  NVFW    500 non-null    float64
 11  VLBG    500 non-null    float64
 12  IDIX    500 non-null    float64
 13  UVHN    500 non-null    float64
 14  IWOT    500 non-null    float64
 15  LEMB    500 non-null    float64
 16  QMYY    500 non-null    float64
 17  XDGR    500 non-null    float64
 18  ODZS    500 non-null    float64
 19  LNJS    500 non-null    float64
 20  WDRT    500 non-null    float64
 21  LKKS    500 non-null    float64
 22  UOBF    500 non-null    float64
 23  VBHE    500 non-null    float64
 24  FRWU    500 non-null    float64
 25  NDYZ    500 non-null    float64
 26  QSBO    500 non-null    float64
 27  JDUB    500 non-null    float64
 28  TEVK    500 non-null    float64
 29  EZTM    500 non-null    float64
 30  Label   500 non-null    float64
dtypes: float64(31)
memory usage: 121.2 KB
```

Step 4 – Scaling our data for Dimensionality Reduction using Autoencoders.

```
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data.drop('Label',axis=1))
scaled_data.shape
```

```
(500, 30)
```

We are using MinMaxScaler here to scale our data. Also here we are checking the shape of our data. While scaling we dropped the Label column as shown above.

Step 5 – Defining no. of nodes in layers.

```
num_inputs = 30
num_hidden = 2
num_outputs = num_inputs # Must be true for an autoencoder!
```

Step 6 – Building the model for Dimensionality Reduction using Autoencoders.

```
model = Sequential()

model.add(Dense(num_inputs, input_shape=[num_inputs]))
model.add(Dense(num_hidden))
model.add(Dense(num_outputs))

model.compile(optimizer=Adam(0.001), metrics=['accuracy'], loss='mae')

print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 30)                930

 dense_1 (Dense)             (None, 2)                 62

 dense_2 (Dense)             (None, 30)                90

=================================================================
Total params: 1082 (4.23 KB)
Trainable params: 1082 (4.23 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

```
We have a very simple model for this use case.
We have 3 layers.
The first layer is having 30 nodes, 2nd is having 2 nodes and the third is also having 30 nodes.
Our input and output nodes should show the same type of data when building Autoencoders.
```

** Now plot out the reduced dimensional representation of the data. Do you still have clear separation of classes even with the reduction in dimensions? Hint: You definitely should, the classes should still be clearly seperable, even when reduced to 2 dimensions. **

Step 7 – Let's train the model for Dimensionality Reduction using Autoencoders.

```
model.fit(x=scaled_data, y=scaled_data, epochs=1000, batch_size=32)
```

```
16/16 [==============================] - 0s 3ms/step - loss: 0.0744 - accuracy: 0.2300
Epoch 875/1000
16/16 [==============================] - 0s 3ms/step - loss: 0.0745 - accuracy: 0.2200
Epoch 876/1000
16/16 [==============================] - 0s 3ms/step - loss: 0.0744 - accuracy: 0.2340
Epoch 877/1000
16/16 [==============================] - 0s 3ms/step - loss: 0.0744 - accuracy: 0.2360
Epoch 878/1000
16/16 [==============================] - 0s 3ms/step - loss: 0.0743 - accuracy: 0.2320
Epoch 879/1000
16/16 [==============================] - 0s 3ms/step - loss: 0.0743 - accuracy: 0.2340
Epoch 880/1000
16/16 [==============================] - 0s 4ms/step - loss: 0.0743 - accuracy: 0.2480
Epoch 881/1000
16/16 [==============================] - 0s 6ms/step - loss: 0.0743 - accuracy: 0.2300
Epoch 882/1000
16/16 [==============================] - 0s 4ms/step - loss: 0.0744 - accuracy: 0.2280
Epoch 883/1000
16/16 [==============================] - 0s 4ms/step - loss: 0.0744 - accuracy: 0.2400
Epoch 884/1000
16/16 [==============================] - 0s 5ms/step - loss: 0.0744 - accuracy: 0.2380
Epoch 885/1000
16/16 [==============================] - 0s 5ms/step - loss: 0.0743 - accuracy: 0.2340
Epoch 886/1000
16/16 [==============================] - 0s 4ms/step - loss: 0.0743 - accuracy: 0.2280
Epoch 887/1000
16/16 [==============================] - 0s 6ms/step - loss: 0.0744 - accuracy: 0.2280
Epoch 888/1000
16/16 [==============================] - 0s 3ms/step - loss: 0.0745 - accuracy: 0.2260
Epoch 889/1000
16/16 [==============================] - 0s 4ms/step - loss: 0.0744 - accuracy: 0.2340
Epoch 890/1000
16/16 [==============================] - 0s 4ms/step - loss: 0.0744 - accuracy: 0.2520
Epoch 891/1000
16/16 [==============================] - 0s 3ms/step - loss: 0.0743 - accuracy: 0.2360
Epoch 892/1000
16/16 [==============================] - 0s 3ms/step - loss: 0.0743 - accuracy: 0.2320
Epoch 893/1000
16/16 [==============================] - 0s 3ms/step - loss: 0.0744 - accuracy: 0.2360
Epoch 894/1000
16/16 [==============================] - 0s 4ms/step - loss: 0.0743 - accuracy: 0.2360
Epoch 895/1000
16/16 [==============================] - 0s 4ms/step - loss: 0.0744 - accuracy: 0.2320
Epoch 896/1000
16/16 [==============================] - 0s 4ms/step - loss: 0.0743 - accuracy: 0.2380
Epoch 897/1000
16/16 [==============================] - 0s 5ms/step - loss: 0.0743 - accuracy: 0.2400
Epoch 898/1000
16/16 [==============================] - 0s 5ms/step - loss: 0.0744 - accuracy: 0.2320
Epoch 899/1000
16/16 [==============================] - 0s 4ms/step - loss: 0.0743 - accuracy: 0.2280
Epoch 900/1000
16/16 [==============================] - 0s 4ms/step - loss: 0.0744 - accuracy: 0.2360
Epoch 901/1000
16/16 [==============================] - 0s 3ms/step - loss: 0.0745 - accuracy: 0.2320
Epoch 902/1000
16/16 [==============================] - 0s 5ms/step - loss: 0.0746 - accuracy: 0.2400
Epoch 903/1000
16/16 [==============================] - 0s 4ms/step - loss: 0.0744 - accuracy: 0.2360
```

Step 8 – Take the output from the middle layer.

```
intermediate_layer_model = Model(inputs=model.input, outputs=model.get_layer(index=1).output)
intermediate_output = intermediate_layer_model.predict(scaled_data)
```

```
16/16 [==============================] - 0s 2ms/step
```

We can't just directly take the output from our middle layer. That's why we need to create a model, with just 1 layer which will be our middle layer. model.get_layer(index=1) is extracting the middle layer from our original model and .output is used for taking its output. In the second line, we are simply using predict to take the results from the 2nd layer.

```
intermediate_output.shape
```

```
(500, 2)
```

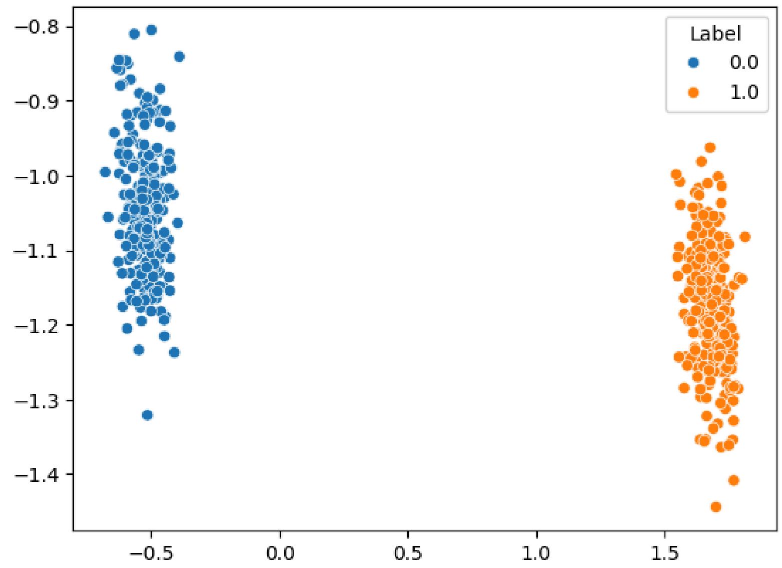
Step 9 – Checking the output shape of our result.

As we can see below that the shape of the intermediate output became 500X2 means 30 columns/features are now suppressed to only 2 columns/features.

Step 10 – Plotting our results for Dimensionality Reduction using Autoencoders.

```
sns.scatterplot(x=intermediate_output[:,0], y=intermediate_output[:,1], hue=data['Label'])
```

```
<Axes: >
```



And BOOM here is the result. The amount of data our 30 features were showing, we are able to show that data precisely using just these 2 dimensions. Both classes are linearly separable, which means our model did a good job of keeping the essence of the data.