```
#VAISHNAVI SOLANKAR
#MY PROGRAM
import numpy as np
import pandas as pd
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input,  Dense
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

```
#Generate synthetic data
data=np.random.normal(0,1,(1000,20))

#introduce some anomalies
anomalies = np.random.normal(0,5,(50, 20))
data_with_anomalies = np.vstack([data,anomalies])
```

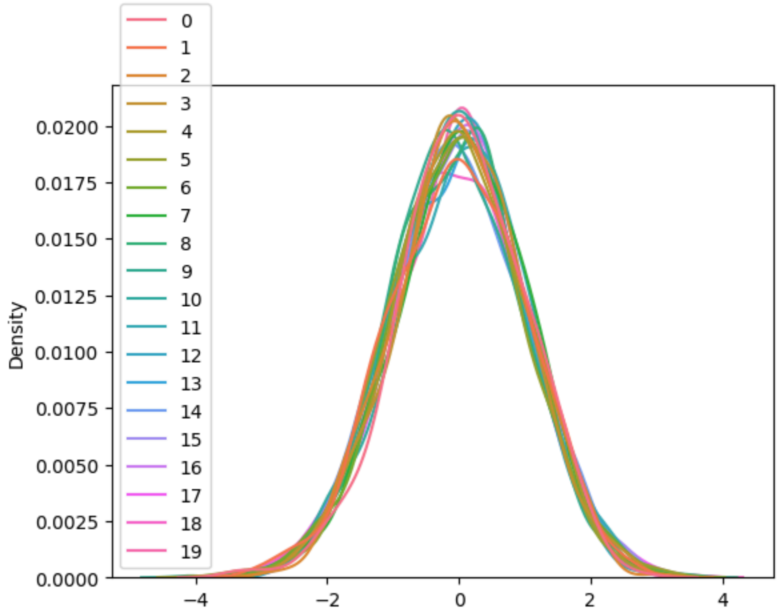
```
data_with_anomalies.shape
```

```
(1050, 20)
```

```
import seaborn as sns
```

```
sns.kdeplot(data)
```

```
<Axes: ylabel='Density'>
```




```
#normalize the data
scaler=MinMaxScaler()
data_scaled=scaler.fit_transform(data_with_anomalies)

#split the data into trainning and testing set
x_train,x_test=train_test_split(data_scaled,test_size=0.2,random_state=0)
```

```
x_test.shape
```

```
(210, 20)
```

```
x_train.shape
```

```
(840, 20)
```

```
#DEfine\ the input layer
input_layer=Input(shape=(x_train.shape[1],))

#Encoder
encoded=Dense(16,activation='relu')(input_layer)
encoded=Dense(8,activation='relu')(encoded)
encoded=Dense(4,activation='relu')(encoded
                                   )

#Decoder
decoded=Dense(8,activation='relu')(encoded)
decoded=Dense(16,activation='relu')(decoded)
decoded=Dense(x_train.shape[1],activation='sigmoid')(decoded)

#Autoencoder
autoencoder=Model(inputs=input_layer,outputs=decoded)
autoencoder.compile(optimizer='adam',loss='mse')


autoencoder.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 20)	0
dense_9 (Dense)	(None, 16)	336
dense_10 (Dense)	(None, 8)	136
dense_11 (Dense)	(None, 4)	36
dense_12 (Dense)	(None, 8)	40
dense_13 (Dense)	(None, 16)	144
dense_14 (Dense)	(None, 20)	340

```
Total params: 3,098 (12.11 KB)
Trainable params: 1,032 (4.03 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 2,066 (8.07 KB)
```

```
#compile the model
autoencoder.compile(optimizer='adam',loss='mse',metrics=['accuracy'])

#train the model
history=autoencoder.fit(x_train,x_train,epochs=50,batch_size=32,validation_data=(x_test,x_test),validation_split=0.1)
#
```

```
Epoch 1/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - accuracy: 0.4509 - loss: 0.0033 - val_accuracy: 0.4524 - val_loss: 0.0033
Epoch 2/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4462 - loss: 0.0037 - val_accuracy: 0.4381 - val_loss: 0.0033
Epoch 3/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.3942 - loss: 0.0029 - val_accuracy: 0.4476 - val_loss: 0.0033
Epoch 4/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4205 - loss: 0.0036 - val_accuracy: 0.4429 - val_loss: 0.0033
Epoch 5/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4450 - loss: 0.0036 - val_accuracy: 0.4476 - val_loss: 0.0033
Epoch 6/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4346 - loss: 0.0039 - val_accuracy: 0.4476 - val_loss: 0.0033
Epoch 7/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.4132 - loss: 0.0041 - val_accuracy: 0.4476 - val_loss: 0.0033
Epoch 8/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4565 - loss: 0.0037 - val_accuracy: 0.4476 - val_loss: 0.0033
Epoch 9/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4311 - loss: 0.0040 - val_accuracy: 0.4476 - val_loss: 0.0033
Epoch 10/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4369 - loss: 0.0034 - val_accuracy: 0.4476 - val_loss: 0.0033
Epoch 11/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4695 - loss: 0.0035 - val_accuracy: 0.4429 - val_loss: 0.0033
Epoch 12/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4395 - loss: 0.0042 - val_accuracy: 0.4333 - val_loss: 0.0033
Epoch 13/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4301 - loss: 0.0040 - val_accuracy: 0.4381 - val_loss: 0.0033
Epoch 14/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4362 - loss: 0.0041 - val_accuracy: 0.4333 - val_loss: 0.0033
Epoch 15/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.4052 - loss: 0.0042 - val_accuracy: 0.4429 - val_loss: 0.0033
Epoch 16/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4580 - loss: 0.0041 - val_accuracy: 0.4333 - val_loss: 0.0033
Epoch 17/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4173 - loss: 0.0038 - val_accuracy: 0.4333 - val_loss: 0.0033
Epoch 18/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4363 - loss: 0.0033 - val_accuracy: 0.4619 - val_loss: 0.0033
```

```
Epoch 19/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4409 - loss: 0.0036 - val_accuracy: 0.4571 - val_loss: 0.0033
Epoch 20/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4418 - loss: 0.0038 - val_accuracy: 0.4381 - val_loss: 0.0032
Epoch 21/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4344 - loss: 0.0040 - val_accuracy: 0.4381 - val_loss: 0.0033
Epoch 22/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.4159 - loss: 0.0039 - val_accuracy: 0.4333 - val_loss: 0.0033
Epoch 23/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4672 - loss: 0.0041 - val_accuracy: 0.4381 - val_loss: 0.0033
Epoch 24/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4489 - loss: 0.0037 - val_accuracy: 0.4429 - val_loss: 0.0033
Epoch 25/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4407 - loss: 0.0041 - val_accuracy: 0.4286 - val_loss: 0.0033
Epoch 26/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.4114 - loss: 0.0038 - val_accuracy: 0.4381 - val_loss: 0.0033
Epoch 27/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4351 - loss: 0.0037 - val_accuracy: 0.4524 - val_loss: 0.0033
Epoch 28/50
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.4522 - loss: 0.0034 - val_accuracy: 0.4333 - val_loss: 0.0033
Epoch 29/50
```

```
#prredict the reconstruction on test data
x_test_pred=autoencoder.predict(x_test)
#calculate the mean squared erro
mse=mean_squared_error(x_test,x_test_pred,multioutput='raw_values')
#define a threshold for anomaly detection
threshold=np.percentile(mse,90)

#identify anomalies
anomalies=mse>threshold
print(f"Number of anamolies detected :{np.sum(anomalies)}")
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
Number of anamolies detected :2
```

```
mse
```

```
array([0.00206743, 0.00350612, 0.00428984, 0.00333435, 0.00141242,
       0.00416099, 0.0032832 , 0.00288451, 0.00420482, 0.00338601,
       0.00248417, 0.00365894, 0.00318676, 0.00442963, 0.00483045,
       0.00372291, 0.00270032, 0.00270721, 0.00201273, 0.00291589])
```

```
np.mean((x_test - x_test_pred)**2, axis=0)
```

```
array([0.00206743, 0.00350612, 0.00428984, 0.00333435, 0.00141242,
       0.00416099, 0.0032832 , 0.00288451, 0.00420482, 0.00338601,
       0.00248417, 0.00365894, 0.00318676, 0.00442963, 0.00483045,
       0.00372291, 0.00270032, 0.00270721, 0.00201273, 0.00291589])
```

```
threshold
```

```
0.004303819269351191
```

```
mse > threshold
```

```
array([False, False, False, False, False, False, False, False, False,
       False, False, False, False,  True,  True, False, False, False,
       False, False])
```

```
True + True
```

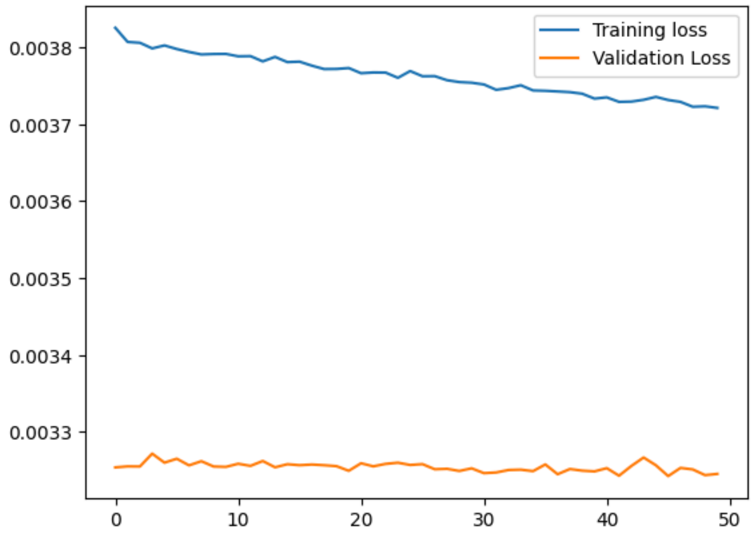
```
2
```

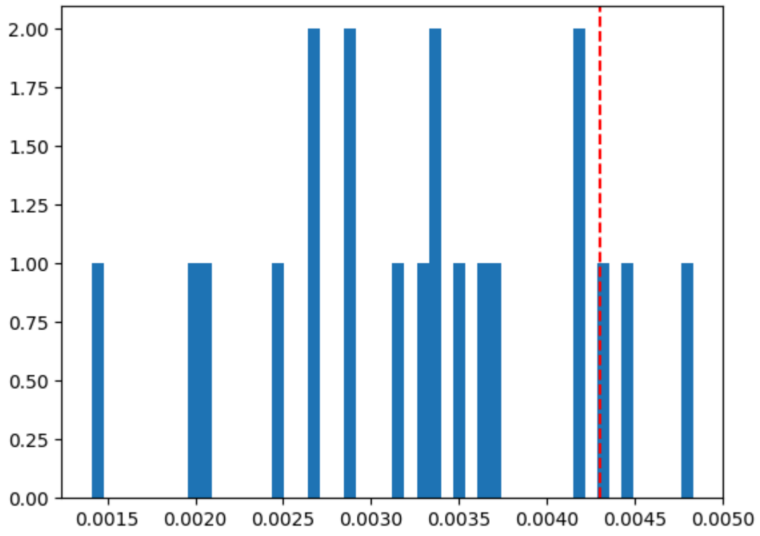
```
import matplotlib.pyplot as plt

#plot the loss over epochs
plt.plot(history.history['loss'],label='Training loss')
plt.plot(history.history['val_loss'],label='Validation Loss')

plt.legend()
plt.show()

#plot mse histogram
plt.hist(mse,bins=50)
plt.axvline(threshold,color='r',linestyle='--',label='Threshold')
plt
plt.show()
```





Start coding or generate with AI.