Problem Statement 4: Use Autoencoder to implement anomaly detection. Build the model by using a. Import required libraries b. Upload/access the dataset c.Encoder converts it into latent representation d. Decoder networks convert it back to the original input e. Compile the models with Optimizer, Loss, and Evaluation

```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, recall_score, accuracy_score, precision_score

RANDOM_SEED = 2021
TEST_PCT = 0.3
LABELS = ["Normal", "Fraud"]


dataset = pd.read_csv("creditcard.csv")


#check for any nullvalues
print("Any nulls in the dataset ",dataset.isnull().values.any())
print('-------')
print("No. of unique labels", len(dataset['Class'].unique()))
print("Label values ",dataset.Class.unique())
#0 is for normal credit card transaction
```

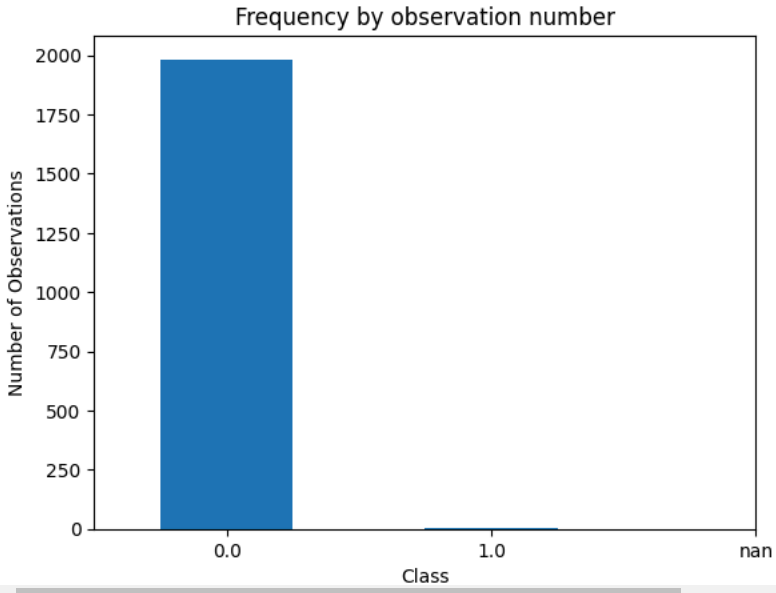
```
#1 is for fraudulent credit card transaction
print('-------')
print("Break down of the Normal and Fraud Transactions")
print(pd.value_counts(dataset['Class'], sort = True))
```

```
Any nulls in the dataset  True
-------
No. of unique labels 3
Label values  [ 0.  1. nan]
-------
Break down of the Normal and Fraud Transactions
Class
0.0    1983
1.0       2
Name: count, dtype: int64
<ipython-input-6-002f82042076>:18: FutureWarning: pandas.value_counts is deprecated and will be removed in a future version. Use pd
  print(pd.value_counts(dataset['Class'], sort = True))
```

```
#Visualizing the imbalanced dataset
count_classes = pd.value_counts(dataset['Class'], sort = True)
count_classes.plot(kind = 'bar', rot=0)
plt.xticks(range(len(dataset['Class'].unique())), dataset.Class.unique())
plt.title("Frequency by observation number")
plt.xlabel("Class")

plt.ylabel("Number of Observations");
```

```
<ipython-input-9-30a6c1fd32bf>:2: FutureWarning: pandas.value_counts is deprecated and will be removed in a future version. Use pd.S
  count_classes = pd.value_counts(dataset['Class'], sort = True)
```



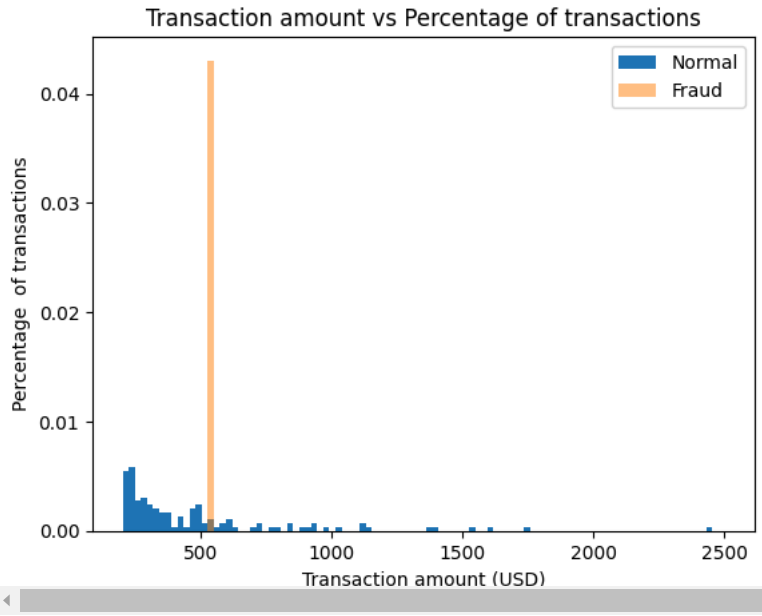

```
# Save the normal and fradulent transactions in separate dataframe normal_dataset = dataset[dataset.Class ==

normal_dataset = dataset [dataset.Class == 0]
fraud_dataset = dataset [dataset.Class == 1]

#Visualize transactionamounts for normal and fraudulent transactions

bins = np.linspace(200, 2500, 100)
plt.hist(normal_dataset.Amount, bins=bins, alpha=1, density=True, label='Normal')
plt.hist(fraud_dataset.Amount, bins=bins, alpha=0.5, density=True, label='Fraud')
plt.legend(loc='upper right')
plt.title("Transaction amount vs Percentage of transactions")

plt.xlabel("Transaction amount (USD)")
plt.ylabel("Percentage  of transactions");
plt.show()
```




```
sc=StandardScaler()
dataset['Time'] = sc.fit_transform(dataset['Time'].values.reshape(-1, 1))
dataset['Amount'] = sc.fit_transform(dataset['Amount'].values .reshape(-1, 1))


raw_data = dataset.values
# if the is represented by a 0 and if fraud then 1
labels = raw_data[:, -1]

#The last element contains transaction is normal which The other data points are the electrocadriogram data
data = raw_data[:, 0:-1]
train_data, test_data, train_labels, test_labels = train_test_split( data, labels, test_size=0.2, random_state=2021)
```

```
min_val = tf.reduce_min(train_data)
max_val = tf.reduce_max(train_data)

train_data = (train_data - min_val) /(max_val - min_val)
test_data = (test_data - min_val) /(max_val - min_val)
train_data = tf.cast(train_data, tf.float32)
test_data = tf.cast(test_data, tf.float32)


train_labels = train_labels.astype(bool)
test_labels = test_labels.astype(bool)

#creating normal and fraud datasets
normal_train_data = train_data[~train_labels]
normal_test_data = test_data[~test_labels]
fraud_train_data = train_data[train_labels]
fraud_test_data = test_data[test_labels]
print(" No. of records in Fraud Train Data=", len(fraud_train_data))
fraud_train_data = print(" No. . of records in Normal Train data=", len(normal_train_data))
print(" No. of records in Fraud Test Data=", len(fraud_test_data))

print(" No. of records in Normal Test data=", len(normal_test_data))
```

```
 No. of records in Fraud Train Data= 3
 No. . of records in Normal Train data= 1585
 No. of records in Fraud Test Data= 0
 No. of records in Normal Test data= 398
```

```
nb_epoch = 50
batch_size = 64
input_dim = normal_train_data.shape[1]

#num of columns, 30
encoding_dim = 14
hidden_dim_1 = int(encoding_dim / 2) #
hidden_dim_2=4
learning_rate = 1e-7


#input Layer
input_layer = tf.keras.layers.Input(shape = (input_dim,))

#Encoder
encoder= tf.keras.layers.Dense(encoding_dim, activation="tanh", activity_regularizer = tf.keras.regularizers.l2(learning_rate)) (input_l
encoder=tf.keras.layers.Dropout(0.2) (encoder)
encoder=tf.keras.layers. Dense(hidden_dim_1, activation = 'relu') (encoder)
encoder= tf.keras.layers. Dense(hidden_dim_2, activation=tf.nn.leaky_relu) (encoder)

#Decoder
decoder = tf.keras.layers.Dense(hidden_dim_1, activation ='relu') (encoder)
decoder=tf.keras.layers.Dropout(0.2) (decoder)
decoder=tf.keras.layers. Dense (encoding_dim, activation= 'relu') (decoder)
decoder = tf.keras.layers.Dense(input_dim, activation='tanh') (decoder)

#Autoencoder
autoencoder = tf.keras.Model(inputs=input_layer, outputs= decoder)
autoencoder.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 30)	0
dense (Dense)	(None, 14)	434
dropout (Dropout)	(None, 14)	0
dense_1 (Dense)	(None, 7)	105
dense_2 (Dense)	(None, 4)	32
dense_3 (Dense)	(None, 7)	35
dropout_1 (Dropout)	(None, 7)	0
dense_4 (Dense)	(None, 14)	112
dense_5 (Dense)	(None, 30)	450